对象存储

SDK 文档

产品文档



腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

SDK 文档
- SDK 概览
- 准备工作
- Android SDK
  - 快速入门
  - Android SDK 常见问题
  - 快速体验
  - 存储桶操作
  - 对象操作
    - 上传对象
    - 下载对象
    - 复制与移动对象
    - 列出对象
    - 删除对象
    - 恢复归档对象
    - 查询对象元数据
    - 生成预签名 URL
    - 预请求跨域配置
    - 服务端加密
    - 单链接限速
    - 检索对象内容
  - 异地容灾
    - 版本控制
    - 跨地域复制
  - 数据管理
    - 生命周期
    - 日志管理
    - 对象标签
    - 存储桶标签
    - 静态网站
    - 清单
  - 访问管理
    - 跨域访问
    - 自定义域名
    - 访问控制

版权所有：腾讯云计算（北京）有限责任公司

防盗链
存储桶策略
数据校验
CRC64 校验
图片处理
图片持久化处理
基础图片处理
图片高级压缩
盲水印
设置自定义头部
设置访问域名（CDN/全球加速）
异常处理
C SDK
快速入门
C SDK
存储桶操作
对象操作
上传对象
复制与移动对象
下载对象
列出对象
删除对象
对象访问 URL
恢复归档对象
查询对象元数据
生成预签名 URL
服务端加密
判断对象是否存在
单链接限速
存储桶管理
数据管理
日志管理
存储桶标签
静态网站
清单
内容审核
视频审核
图片处理

版权所有：腾讯云计算（北京）有限责任公司

- 图片持久化处理
- 基础图片处理
- 图片高级压缩
- 访问管理
- 自定义域名
- 访问控制
- 防盗链
- 数据校验
- CRC64 校验
- 异常处理
- 设置访问域名（CDN/全球加速）

C++ SDK

- 快速入门
- 存储桶操作
- C++ SDK 常见问题
- 对象操作
- 异地容灾
- 版本控制
- 存储桶复制
- 数据管理
- 生命周期
- 日志管理
- 存储桶标签
- 静态网站
- 清单
- 访问管理
- 跨域访问
- 存储桶策略
- 访问控制
- 防盗链
- 预签名 URL
- 对象访问 URL
- 异常处理
- 数据校验
- CRC64 校验
- 设置访问域名（CDN/全球加速）
- 内容审核
- 图片审核

版权所有：腾讯云计算（北京）有限责任公司

视频审核
音频审核
文本审核
文档审核
网页审核
.NET(C#) SDK
快速入门
.NET（C#）SDK 常见问题
存储桶操作
创建存储桶
删除存储桶
查询存储桶列表
检索存储桶
判断存储桶是否存在
对象操作
上传对象
下载对象
复制与移动对象
列出对象
删除对象
判断对象是否存在
恢复归档对象
查询对象元数据
对象访问 URL
生成预签名 URL
预请求跨域配置
服务端加密
单链接限速
检索对象内容
异地容灾
版本控制
存储桶复制
跨地域复制
数据管理
生命周期
日志管理
存储桶标签
对象标签

版权所有：腾讯云计算（北京）有限责任公司

静态网站
清单
访问管理
跨域访问
自定义域名
访问控制
防盗链
图片处理
基础图片处理
盲水印
内容审核
图片审核
视频审核
音频审核
文档审核
文本审核
设置自定义头部
设置访问域名（CDN/全球加速）
异常处理
向下兼容指南
Flutter SDK
快速入门
Flutter SDK 常见问题
快速体验
存储桶操作
对象操作
上传对象
下载对象
列出对象
删除对象
单链接限速
生成预签名链接
异常处理
设置访问域名
Go SDK
快速入门
Go SDK 常见问题
存储桶操作

- 创建存储桶
- 删除存储桶
- 检索存储桶
- 查询存储桶列表
- 判断存储桶是否存在

对象操作
- 上传对象
- 复制与移动对象
- 下载对象
- 列出对象
- 删除对象
- 对象访问 URL
- 恢复归档对象
- 查询对象元数据
- 判断对象是否存在
- 预签名 URL
- 客户端加密
- 服务端加密

异地容灾
- 版本控制
- 存储桶复制

数据管理
- 生命周期
- 日志管理
- 存储桶标签
- 对象标签
- 静态网站
- 清单

访问管理
- 跨域访问
- 存储桶策略
- 自定义域名
- 访问控制
- 全球加速
- 防盗链

数据校验
- CRC64 校验

文件处理

版权所有：腾讯云计算（北京）有限责任公司

文件压缩
文件哈希值计算
文件解压
异常处理
设置访问域名（CDN/全球加速）
图片处理
图片持久化处理
基础图片处理
图片高级压缩
Guetzli 压缩
盲水印
内容审核
图片审核
视频审核
音频审核
文本审核
文档审核
网页审核
iOS SDK
快速入门
iOS SDK 常见问题
快速体验
存储桶操作
对象操作
上传对象
下载对象
列出对象
复制与移动对象
检索对象内容
检查对象是否存在
删除对象
恢复归档对象
查询对象元数据
服务端加密
对象访问 URL
生成预签名链接
预请求跨域配置
异地容灾

版权所有：腾讯云计算（北京）有限责任公司

- 版本控制
- 跨地域复制
- 数据管理
  - 生命周期
  - 日志管理
  - 存储桶标签
  - 静态网站
  - 对象标签
  - 清单
- 访问管理
  - 存储桶策略
  - 跨域访问
  - 自定义域名
  - 访问控制
  - 防盗链
- 图片处理
  - 基础图片处理
- 内容识别
  - 语音识别
    - 队列接口
- 设置自定义头部
- 设置访问域名（CDN/全球加速）
- 异常处理

Java SDK

- 快速入门
- Java SDK 常见问题
- 存储桶操作
- 对象操作
  - 上传对象
  - 下载对象
  - 复制与移动对象
  - 列出对象
  - 删除对象
  - 判断对象是否存在
  - 查询对象元数据
  - 修改对象元数据
  - 对象访问 URL
  - 生成预签名 URL

版权所有：腾讯云计算（北京）有限责任公司

- 恢复归档对象
- 服务端加密
- 客户端加密
- 单链接限速
- 检索对象内容
- 自定义域名上传下载对象

数据管理
- 生命周期
- 日志管理
- 存储桶标签
- 静态网站
- 对象标签
- 清单

异地容灾
- 版本控制
- 存储桶复制

访问管理
- 防盗链
- 跨域访问
- 自定义域名
- 访问控制

图片处理
- 基础图片处理
- 图片持久化处理
- 图片高级压缩

内容审核
- 图片审核
- 视频审核
- 音频审核
- 文档审核
- 文本审核

文件处理
- 文件压缩
- 文件哈希值计算
- 文件解压

媒体处理
- 模板操作
- 工作流操作

版权所有：腾讯云计算（北京）有限责任公司

- 任务操作
- 桶操作
- 队列操作
- 视频截帧
- 视频元信息获取
- AI 内容识别
  - 图片标签
  - 汽车识别
  - 以图搜图
- 异常处理
- 设置访问域名（CDN/全球加速）

JavaScript SDK

- 快速入门
- 存储桶操作
- JavaScript SDK 常见问题
- 对象操作
  - 上传对象
  - 下载对象
  - 列出对象
  - 删除对象
  - 复制与移动对象
  - 恢复归档对象
  - 查询对象元数据
  - 检查对象是否存在
  - 对象访问 URL
  - 生成预签名链接
  - 服务端加密
  - 预请求跨域配置
  - 单链接限速
- 文件处理
  - 文件压缩
  - 文件哈希值计算
  - 文件解压
- 异地容灾
  - 版本控制
  - 跨地域复制
- 数据管理
  - 生命周期

版权所有：腾讯云计算（北京）有限责任公司

存储桶标签
静态网站
对象标签
访问管理
跨域访问
存储桶策略
自定义域名
访问控制
防盗链
数据校验
CRC64 校验
图片处理
基础图片处理
盲水印
内容审核
图片审核
视频审核
音频审核
文本审核
网页审核
设置访问域名（CDN/全球加速）
异常处理
Node.js SDK
快速入门
存储桶操作
对象操作
上传对象
下载对象
列出对象
删除对象
复制与移动对象
恢复归档对象
查询对象元数据
检查对象是否存在
对象访问 URL
生成预签名链接
预请求跨域配置
单链接限速

版权所有：腾讯云计算（北京）有限责任公司

- 服务端加密
- 异地容灾
  - 版本控制
  - 跨地域复制
- 数据管理
  - 生命周期
  - 日志管理
  - 存储桶标签
  - 对象标签
  - 静态网站
  - 清单
- 访问管理
  - 跨域访问
  - 存储桶策略
  - 自定义域名
  - 访问控制
  - 防盗链
- 数据校验
  - CRC64 校验
- 设置访问域名（CDN/全球加速）
- 图片处理
  - 基础图片处理
  - 盲水印
- 异常处理
- 内容审核
  - 视频审核

PHP SDK

- 快速入门
- PHP SDK 常见问题
- 对象操作
  - 上传对象
  - 复制与移动对象
  - 下载对象
  - 列出对象
  - 删除对象
  - 恢复归档对象
  - 生成对象访问 URL
  - 生成预签名 URL

- 判断对象是否存在
- 服务端加密
- 查询对象元数据

存储桶操作
- 创建存储桶
- 删除存储桶
- 查询存储桶列表
- 检索存储桶
- 判断存储桶是否存在

数据管理
- 生命周期
- 日志管理
- 存储桶标签
- 对象标签
- 静态网站
- 清单

访问管理
- 跨域访问
- 自定义域名
- 访问控制
- 防盗链

异常处理

设置访问域名（CDN/全球加速）

图片处理
- 基础图片处理
- 图片样式
- 图片持久化处理
- 盲水印
- 图片高级压缩
- Guetzli 压缩

媒体处理
- 队列接口
- 媒体 bucket 接口
- 私有 M3U8 接口

文档处理
- 文档转码
  - 同步请求接口
  - 异步处理任务接口

版权所有：腾讯云计算（北京）有限责任公司

异步处理队列接口
文件处理
多文件打包压缩
文件哈希值计算
文件解压
任务接口
动图任务接口
截图任务接口
转码任务接口
拼接任务接口
智能封面任务接口
多任务接口
视频增强任务接口
精彩集锦任务接口
人声分离任务接口
模板接口
搜索模板接口
Python SDK
快速入门
Python SDK 常见问题
存储桶操作
对象操作
上传对象
下载对象
复制和移动对象
列出对象
删除对象
判断对象是否存在
查询对象元数据
修改对象元数据
对象访问 URL
生成预签名 URL
恢复归档对象
检索对象内容
服务端加密
对象加密
存储桶加密
客户端加密

版权所有：腾讯云计算（北京）有限责任公司

单链接限速
异地容灾
版本控制
存储桶复制
数据管理
生命周期
日志管理
存储桶标签
对象标签
静态网站
清单
访问管理
跨域访问
存储桶策略
自定义域名
访问控制
防盗链
内容识别
图片二维码
设置访问域名（CDN/全球加速）
异常处理
图片处理
图片持久化处理
图片高级压缩
盲水印
React Native SDK
快速入门
React Native SDK 常见问题
快速体验
存储桶操作
对象操作
上传对象
下载对象
列出对象
删除对象
单链接限速
生成预签名链接
异常处理

- 设置访问域名
- Mini Program SDK
  - 快速入门
  - 小程序 SDK 常见问题
  - 存储桶操作
  - 对象操作
    - 上传对象
    - 下载对象
    - 列出对象
    - 删除对象
    - 复制与移动对象
    - 恢复归档对象
    - 查询对象元数据
    - 检查对象是否存在
    - 对象访问 URL
    - 生成预签名链接
    - 预请求跨域配置
    - 单链接限速
    - 服务端加密
  - 异地容灾
    - 版本控制
    - 跨地域复制
  - 数据管理
    - 生命周期
    - 日志管理
    - 存储桶标签
    - 对象标签
    - 静态网站
    - 清单
  - 访问管理
    - 跨域访问
    - 存储桶策略
    - 自定义域名
    - 防盗链
  - 数据校验
    - CRC64 校验
  - 内容审核
    - 图片审核

版权所有：腾讯云计算（北京）有限责任公司

视频审核
音频审核
文本审核
文档审核
网页审核
设置访问域名（CDN/全球加速）
图片处理
基础图片处理
盲水印
异常处理
错误码

版权所有：腾讯云计算（北京）有限责任公司

SDK 文档

SDK 概览

最近更新时间：2021-07-07 10:28:44

腾讯云对象存储（Cloud Object Storage，COS）除了提供多种 API 接口，还提供了丰富多样的 SDK 供开发者使用，下表展示了 COS 所支持的 SDK 和对应的快速入门文档指引，开发者们可查看对应的 SDK 快速入门文档，完成 SDK 下载、安装、初始化等操作。

SDK	接入文档
Android SDK	Android SDK 快速入门
C SDK	C SDK 快速入门
C++ SDK	C++ SDK 快速入门
.NET SDK	.NET SDK 快速入门
Go SDK	Go SDK 快速入门
iOS SDK	iOS SDK 快速入门
Java SDK	Java SDK 快速入门
JavaScript SDK	JavaScript SDK 快速入门
Node.js SDK	Node.js SDK 快速入门
PHP SDK	PHP SDK 快速入门
Python SDK	Python SDK 快速入门
小程序 SDK	小程序 SDK 快速入门

版权所有：腾讯云计算（北京）有限责任公司

准备工作

最近更新时间：2021-12-27 12:43:13

在使用 SDK 时可能会涉及以下概念和术语，建议您提前了解并准备相关信息：

名称	描述
APPID	开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 API 密钥管理 页面获取
SecretId	开发者拥有的项目身份识别 ID，用于身份认证，可在 API 密钥管理 页面获取
SecretKey	开发者拥有的项目身份密钥，可在 API 密钥管理 页面获取
Bucket	存储桶，COS 中用于存储数据的容器。有关存储桶的进一步说明，请参见 存储桶概述 文档
BucketName-APPID	完整的存储桶名称，以 APPID 为后缀，例如 `examplebucket-1250000000`
Object	对象，COS 中存储的具体文件，是存储的基本实体
ObjectKey	对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 对象概述 文档
Region	地域信息，枚举值可参见 可用地域 文档，例如：ap-beijing、ap-hongkong、eu-frankfurt 等
ACL	访问控制列表（Access Control List），是指特定 Bucket 或 Object 的访问控制信息列表，请参见 ACL概述 文档

说明

如果您在使用 XML 版本 SDK 时遇到函数或方法不存在等错误，请先将 XML 版本 SDK 升级到最新版再重试。

版权所有：腾讯云计算（北京）有限责任公司

Android SDK 快速入门

最近更新时间：2024-01-04 15:52:02

相关资源

SDK 源码下载请参见 XML Android SDK。
示例 Demo 请参见 XML Android SDK Demo。
SDK 接口与参数文档请参见 SDK API 参考。
SDK 文档中的所有示例代码请参见 SDK 代码示例。
SDK 更新日志请参见 ChangeLog。
SDK 常见问题请参见：Android SDK 常见问题。

说明

如果您在使用 XML 版本 SDK 时遇到函数或方法不存在等错误，请先将 XML 版本 SDK 升级到最新版再重试。

准备工作

1. 您需要一个 Android 应用，这个应用可以是您现有的工程，也可以是您新建的一个空的工程。
2. 请确保您的 Android 应用目标为 API 级别 15 (Ice Cream Sandwich) 或更高版本。
3. 您需要一个可以获取腾讯云临时密钥的远程地址，关于临时密钥的有关说明请参见 移动应用直传实践。

第一步：安装 SDK

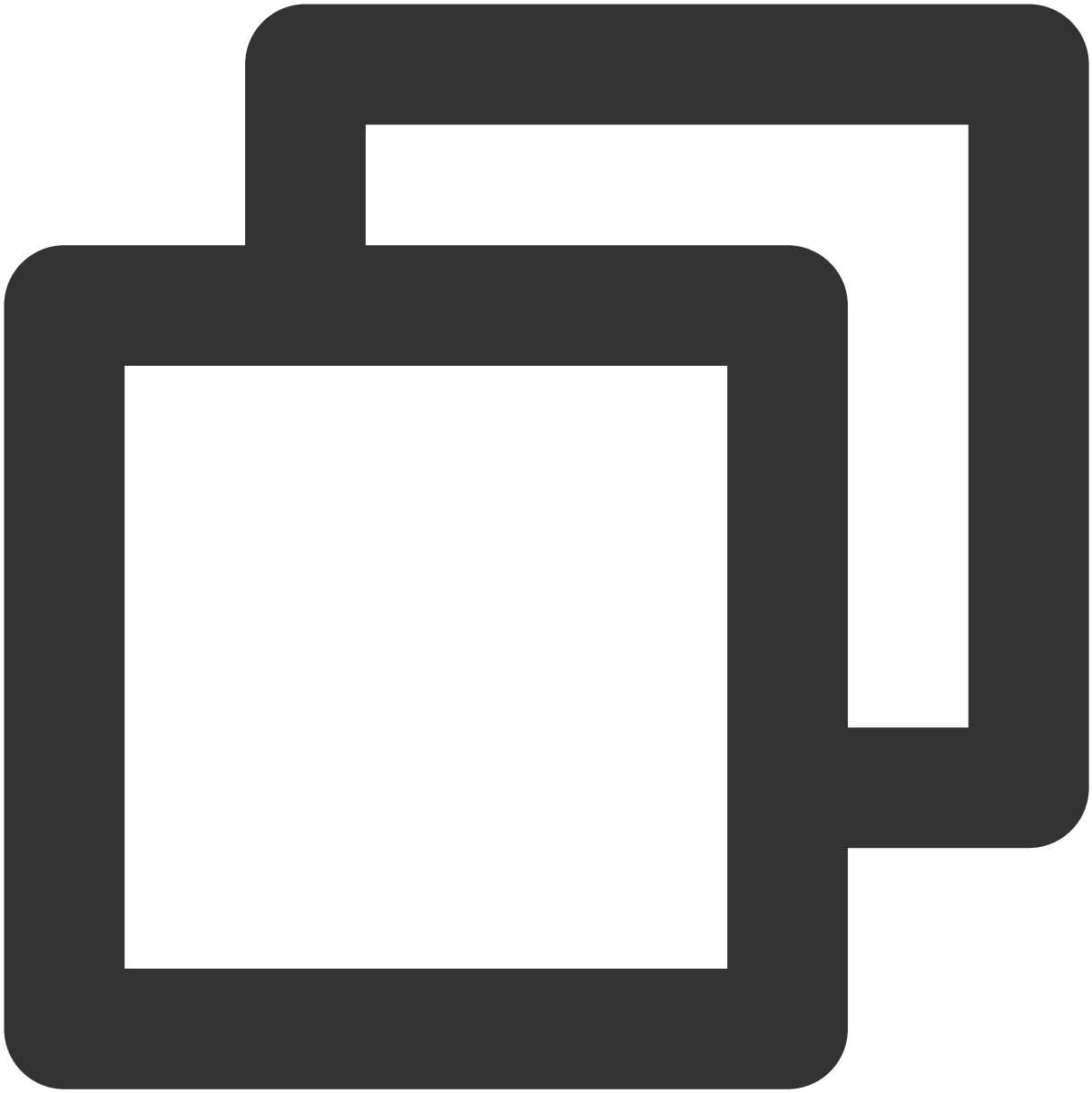
方式一：自动集成（推荐）

说明

bintray 仓库已经下线，COS SDK 已经迁移到 mavenCentral，引用路径和之前不同，您在更新的时候请使用新的引用路径。

使用 mavenCentral 仓库

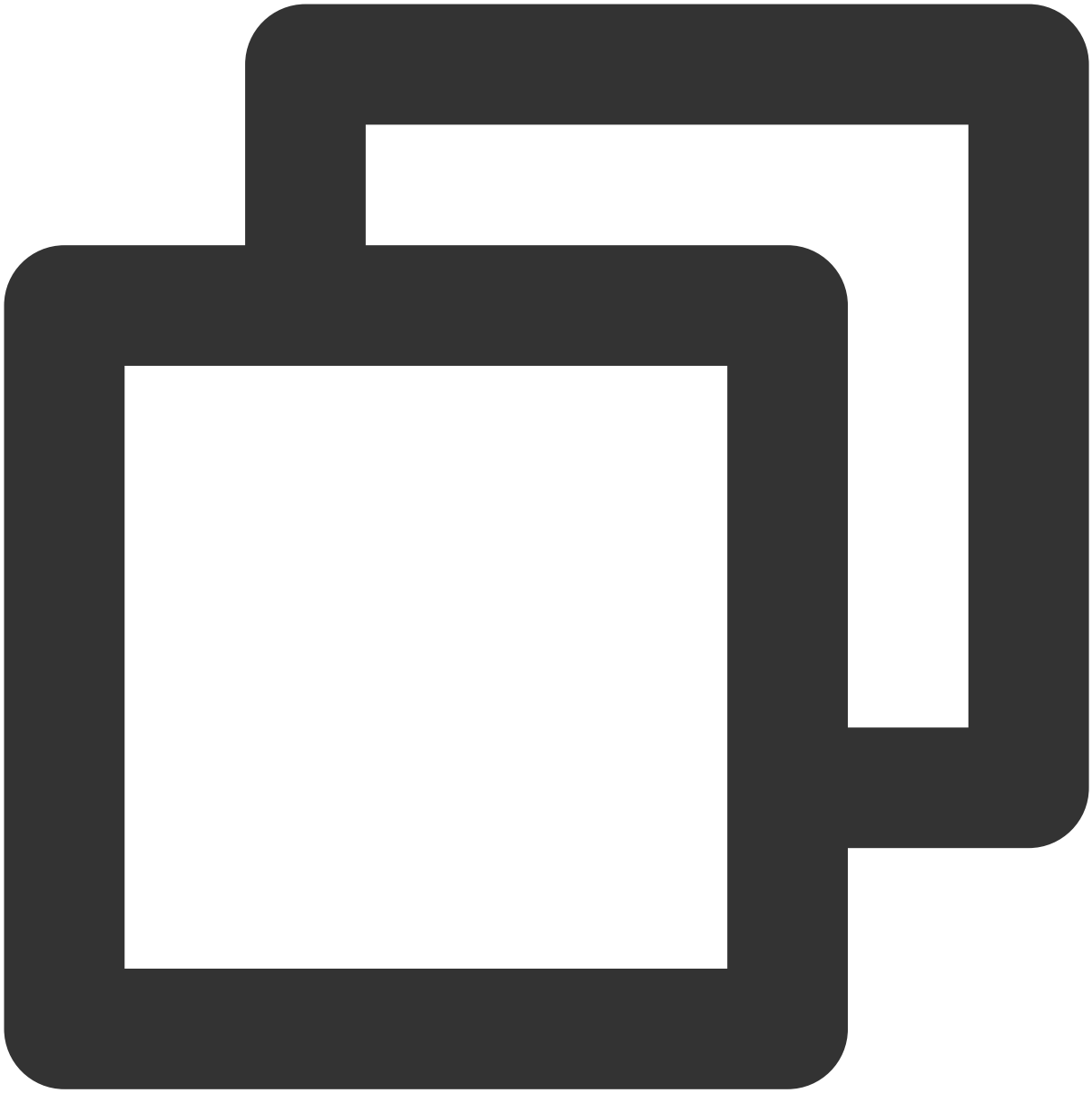
在项目级别（通常是根目录下）的 `build.gradle` 中添加：



```
repositories {
    google()
    // 增加这行
    mavenCentral()
}
```

标准版 SDK

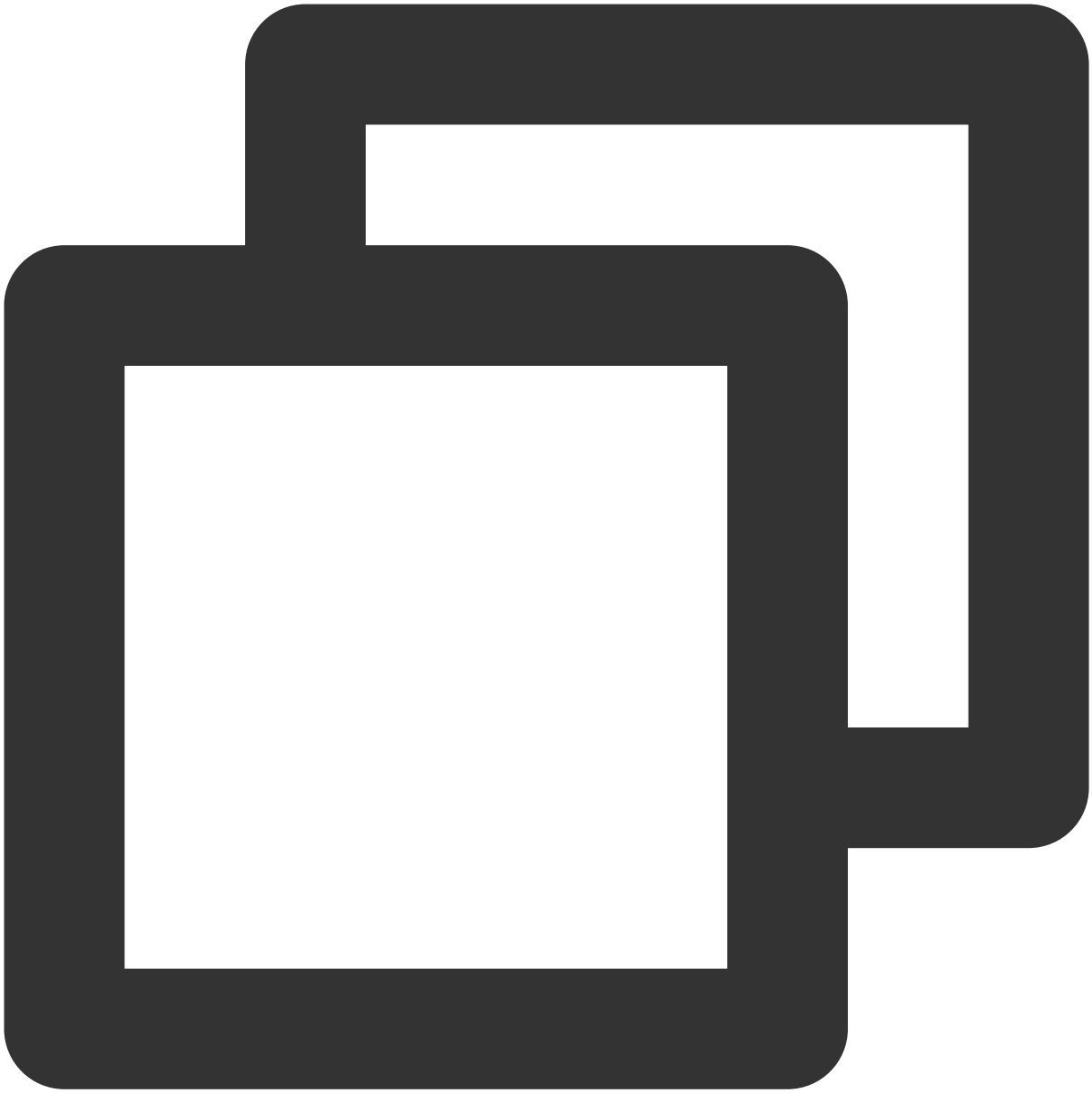
在应用级别（通常是 App 模块下）的 `build.gradle` 中添加依赖：



```
dependencies {
    ...
    // 增加这行
    implementation 'com.qcloud.cos:cos-android:5.9.+'
}
```

精简版 SDK

在应用级别（通常是 App 模块下）的 `build.gradle` 中添加依赖：



```
dependencies {
    ...
    // 增加这行
    implementation 'com.qcloud.cos:cos-android-lite:5.9.+'
}
```

关闭腾讯灯塔上报功能

为了持续跟踪和优化 SDK 的质量，给您带来更好的使用体验，我们在 SDK 中引入了 腾讯灯塔 SDK。

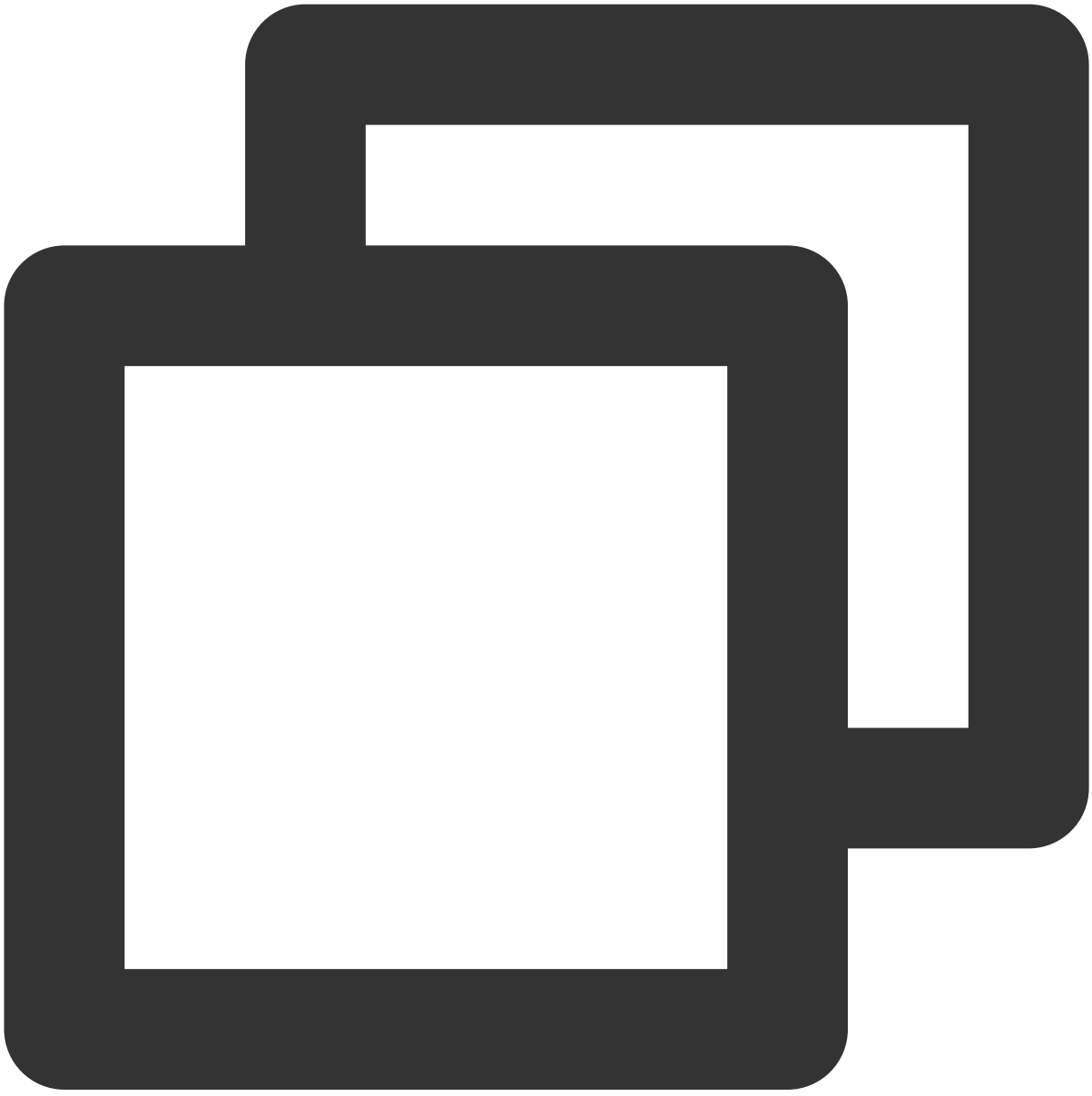
说明

腾讯灯塔只对 COS 侧的请求性能进行监控，不会上报业务侧数据。

若是想关闭该功能，请在应用级别（通常是 App 模块下）的 `build.gradle` 中进行如下操作:

版本为5.8.0以及以上：

修改 cos-android 的依赖为



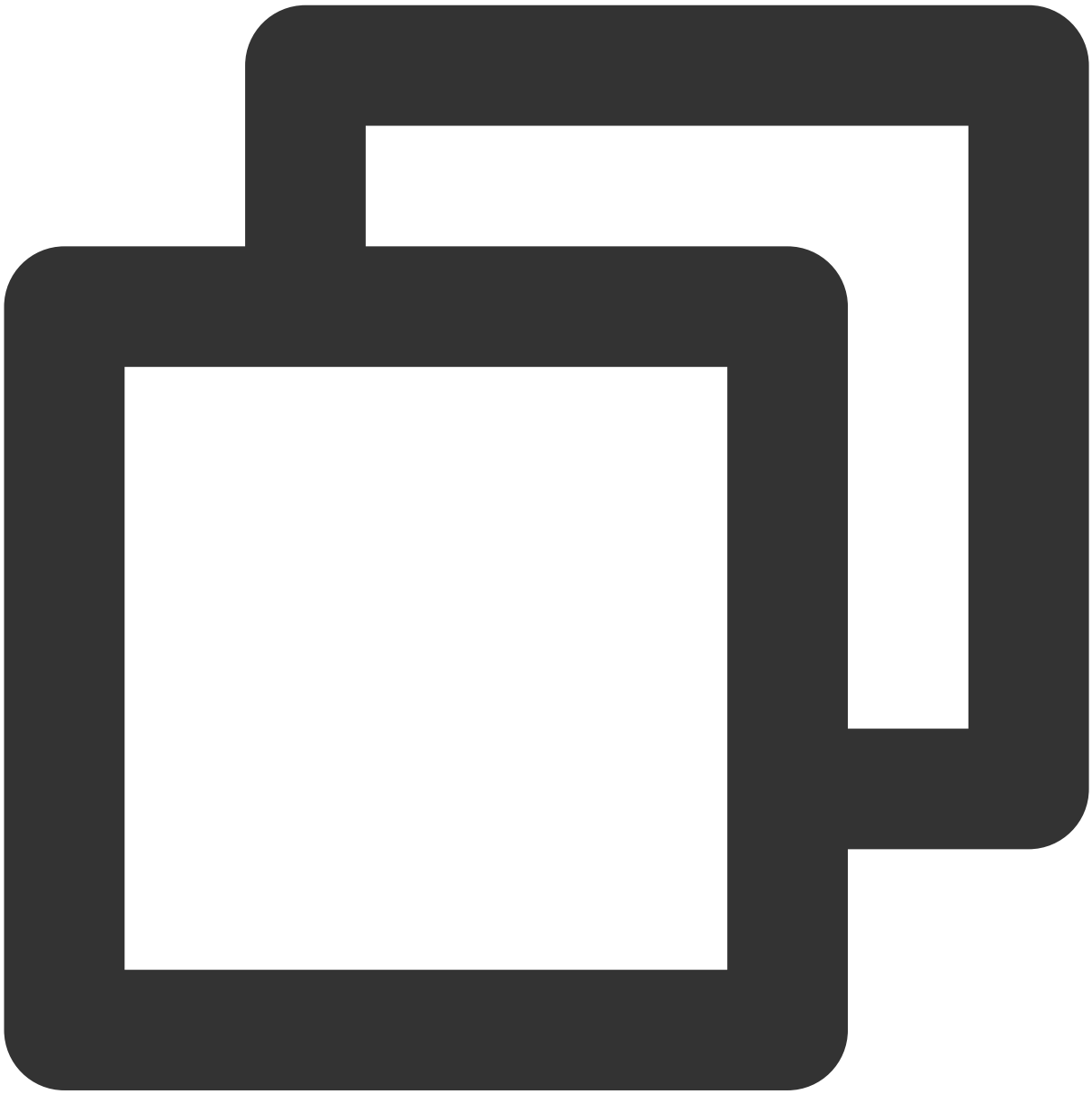
```
dependencies {
    ...
    // 修改为
    implementation 'com.qcloud.cos:cos-android-nobeacon:x.x.x'

    //lite 版本修改为
```

```
    implementation 'com.qcloud.cos:cos-android-lite-nobeacon:x.x.x'
}
```

版本为5.5.8至5.7.9：

添加去掉 beacon 的语句



```
dependencies {
    ...
    implementation ('com.qcloud.cos:cos-android:x.x.x'){
        // 增加这行
        exclude group: 'com.tencent.qcloud', module: 'beacon-android-release'
```

```
        }
    }
```

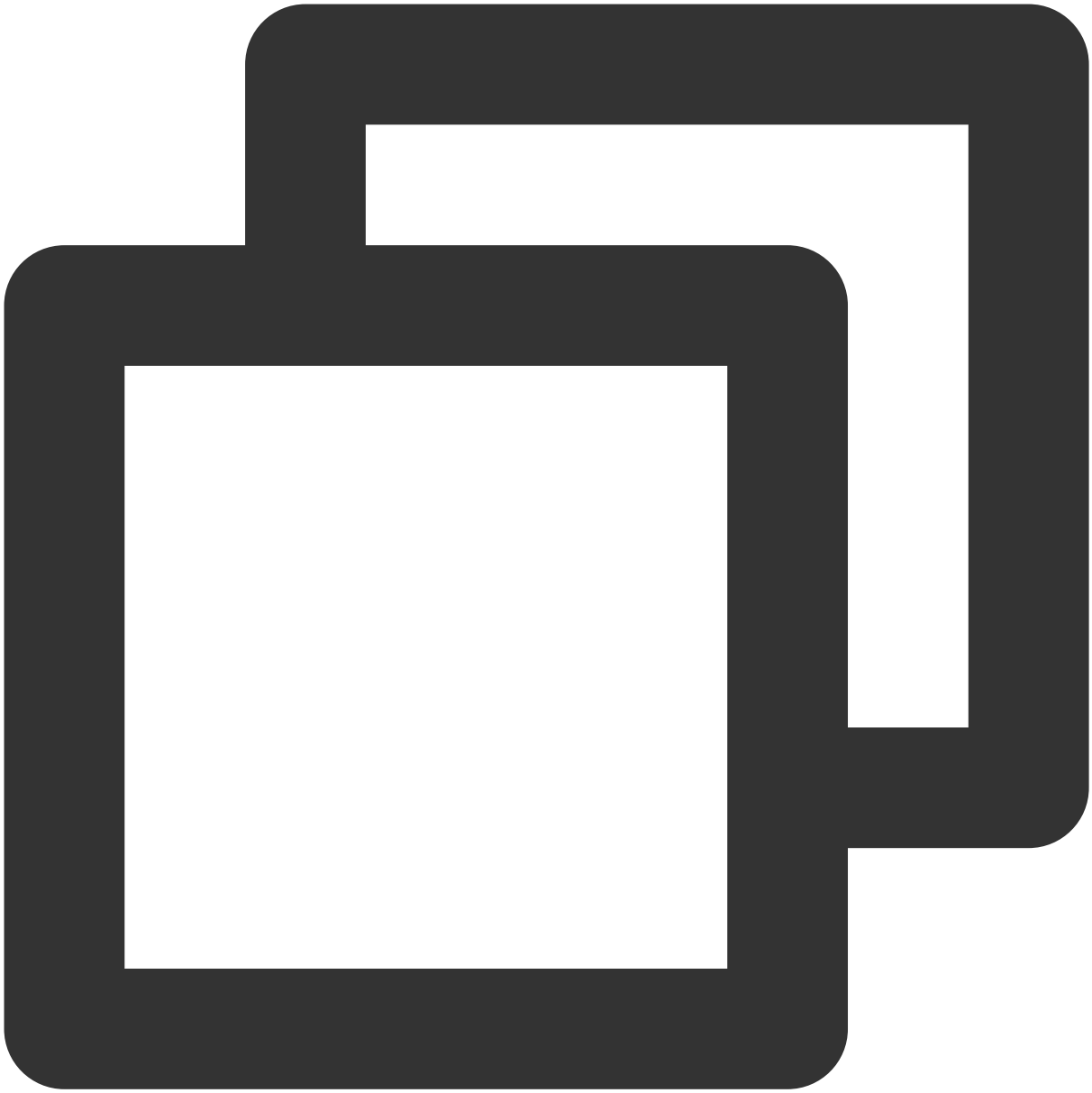
方式二：手动集成

1. 下载归档文件

您可以通过 快速下载地址 下载最新的正式包，也可以在 SDK Releases 里面找到我们所有历史版本的正式包。
下载完成并解压后，您可以看到里面包含了数个 `jar` 或 `aar` 包。下面是对它们的简单说明，请根据需要选择集成的包。
必选的库：
cos-android：COS 协议实现
qcloud-foundation：基础库
bolts-tasks：第三方 Task 库
okhttp：第三方 Networking 库
okio：第三方 IO 库
可选的库：
quic：QUIC 协议，当您使用到 QUIC 协议来传输数据时需要
beacon： beacon 移动分析，用于改进 SDK

2. 集成到项目中

把需要的包放到您应用模块下的 `libs` 文件夹下，并在应用级别（通常是 App 模块下）的 `build.gradle` 文件中添加如下依赖：

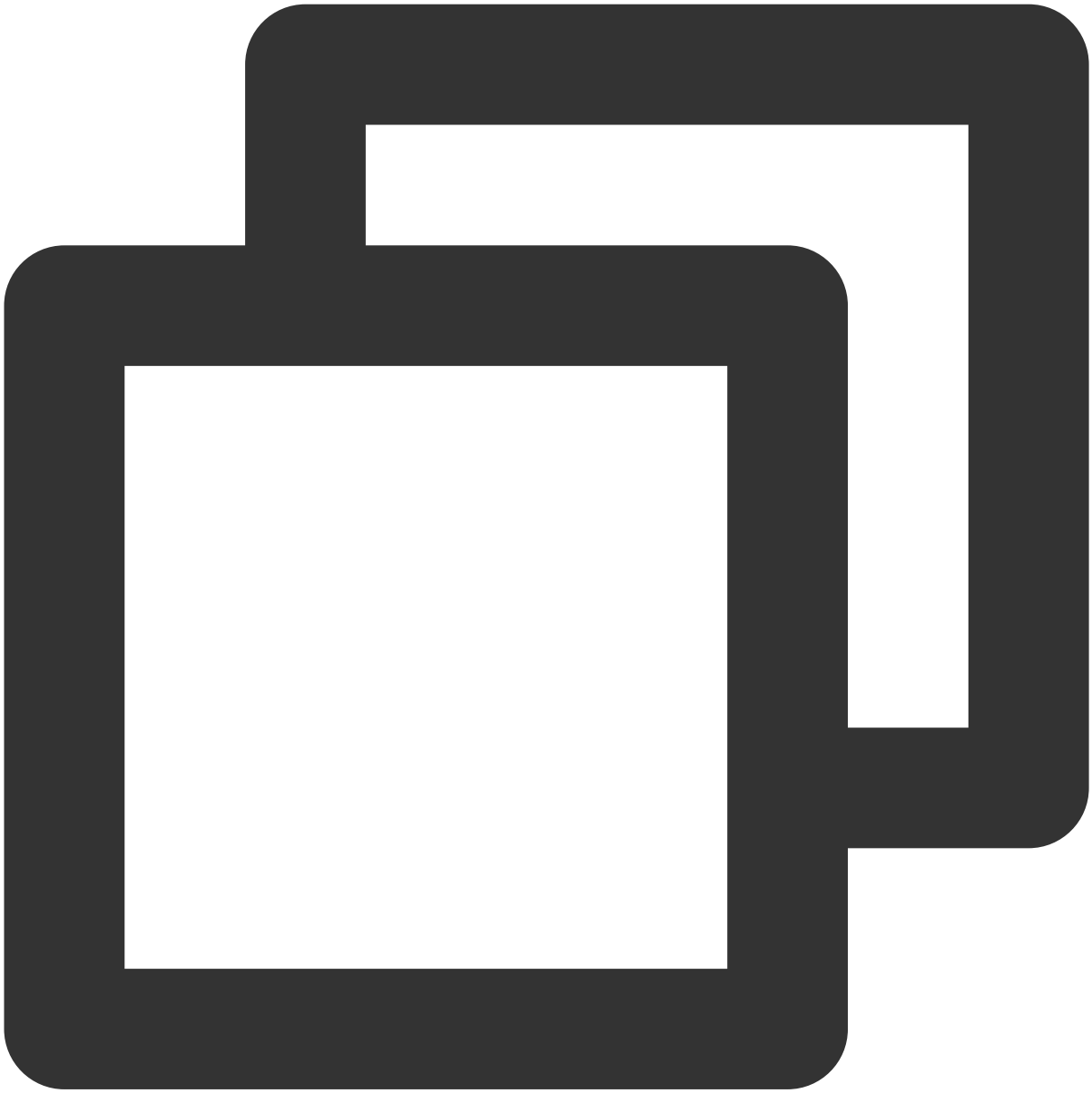


```
dependencies {
    ...
    // 增加这行
    implementation fileTree(dir: 'libs', include: ['*.jar', '*.aar'])
}
```

第二步：配置权限

网络权限

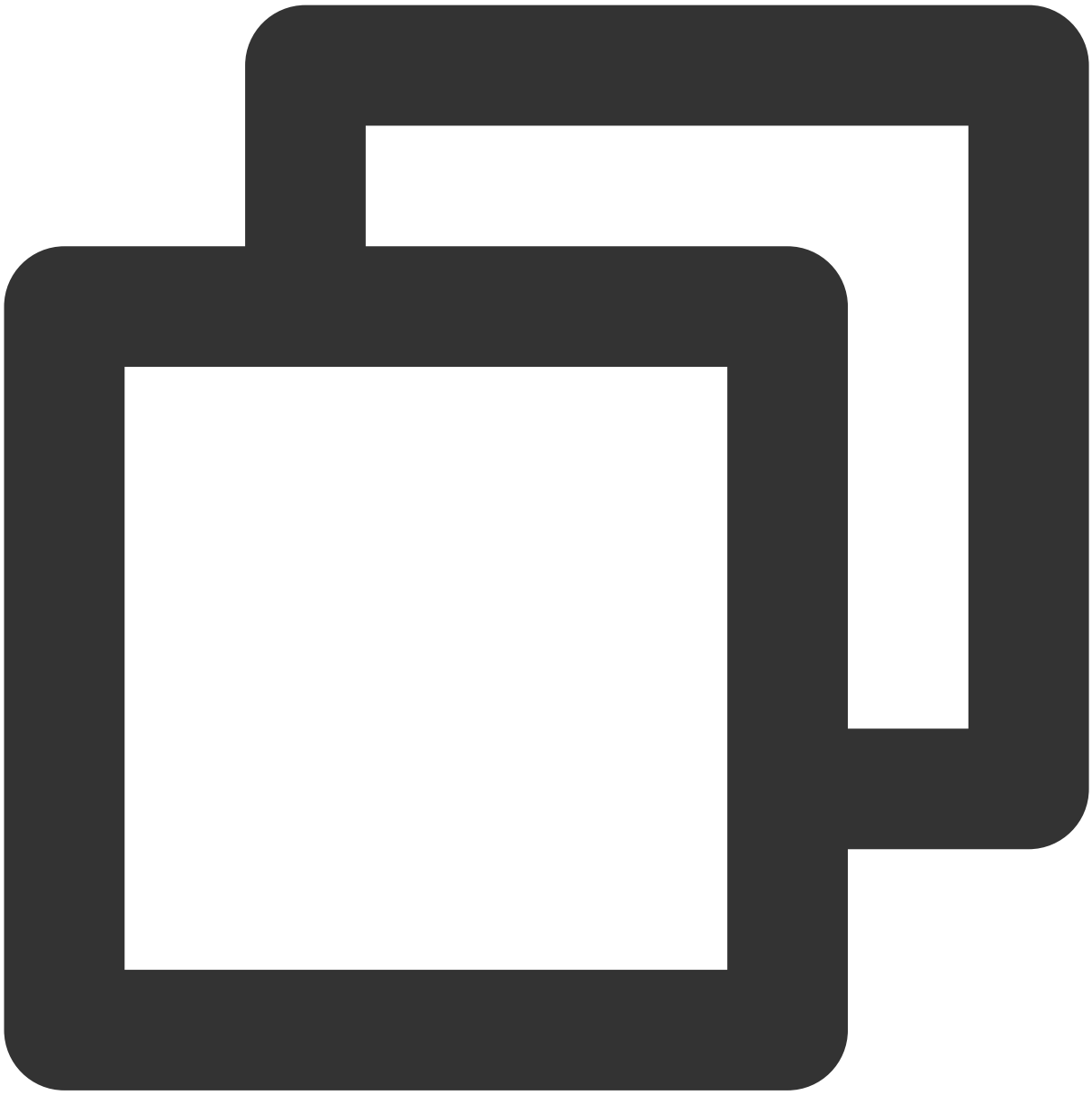
SDK 需要网络权限，用于与 COS 服务器进行通信，请在应用模块下的 `AndroidManifest.xml` 中添加如下权限声明：



```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

存储权限

如果您的应用场景中需要从外部存储中读写文件，请在应用模块下的 `AndroidManifest.xml` 中添加如下权限声明：



```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

注意

在 Android 6.0（API level 23）以上，您需要在运行时动态申请存储权限。

第三步：开始使用

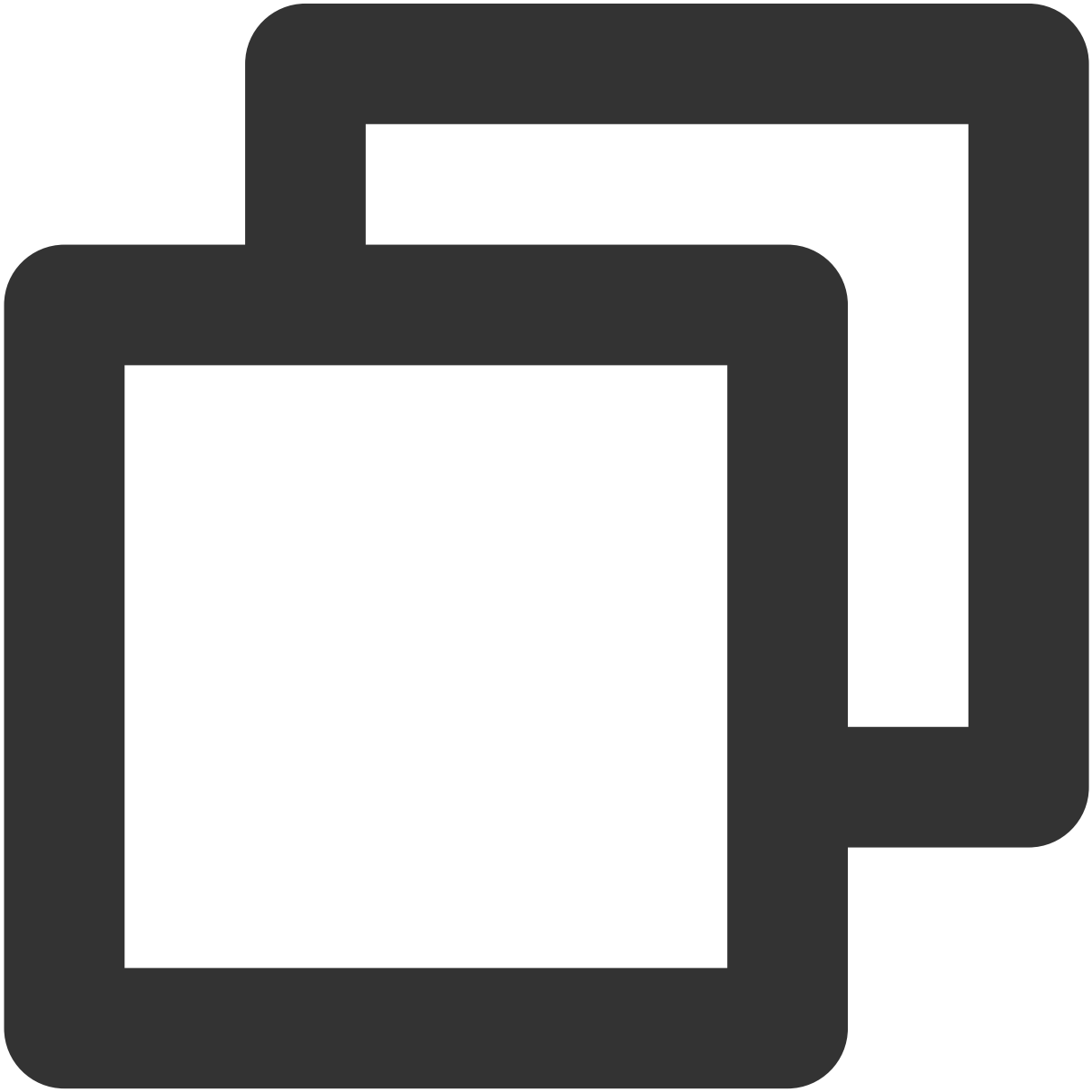
注意

建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。

如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

1. 实现获取临时密钥

实现一个 `BasicLifecycleCredentialProvider` 的子类，实现请求临时密钥并返回结果的过程。



```
public static class MySessionCredentialProvider
        extends BasicLifecycleCredentialProvider {

    @Override
    protected QCloudLifecycleCredentials fetchNewCredentials()
            throws QCloudClientException {

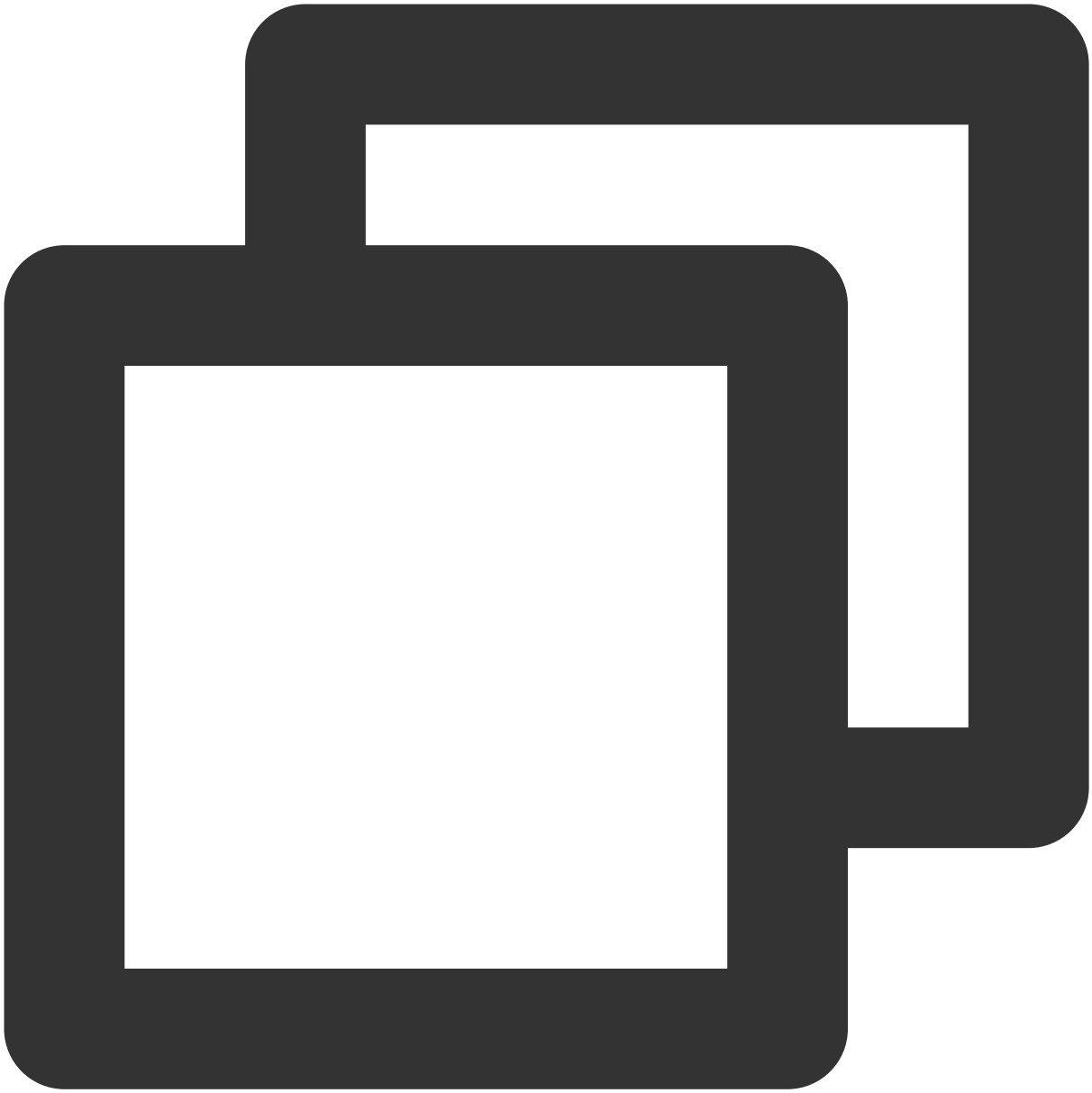
        // 首先从您的临时密钥服务器获取包含了密钥信息的响应

        // 然后解析响应，获取临时密钥信息
        String tmpSecretId = "SECRETID"; // 临时密钥 SecretId
        String tmpSecretKey = "SECRETKEY"; // 临时密钥 SecretKey
        String sessionToken = "SESSIONTOKEN"; // 临时密钥 Token
        long expiredTime = 1556183496L;//临时密钥有效截止时间戳，单位是秒

        //建议返回服务器时间作为签名的开始时间，避免由于用户手机本地时间偏差过大导致请求过期
        // 返回服务器时间作为签名的起始时间
        long startTime = 1556182000L; //临时密钥有效起始时间，单位是秒

        // 最后返回临时密钥信息对象
        return new SessionQCloudCredentials(tmpSecretId, tmpSecretKey,
                sessionToken, startTime, expiredTime);
    }
}
```

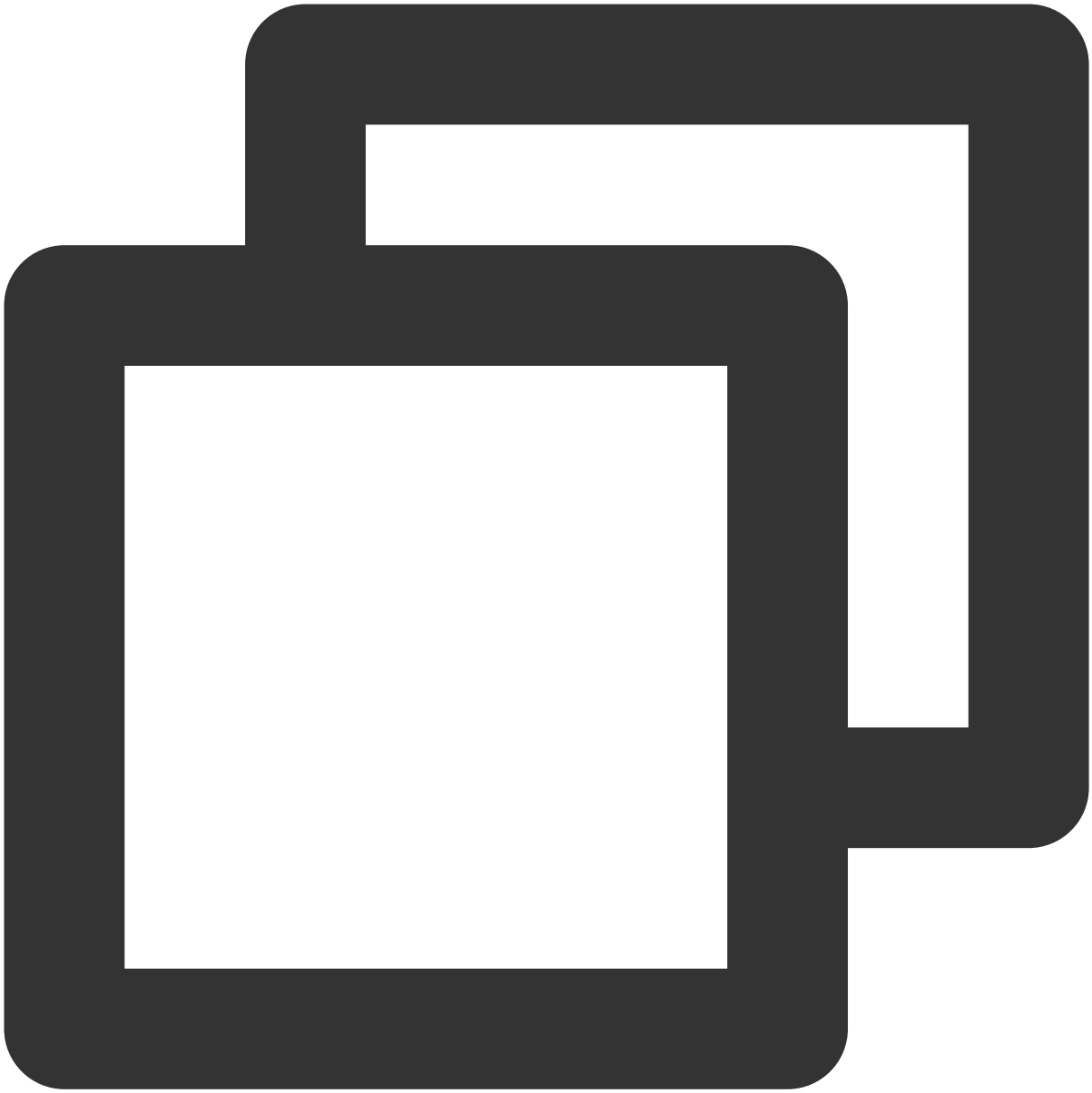
这里假设类名为 `MySessionCredentialProvider` 。初始化一个实例，来给 SDK 提供密钥。



```
QCloudCredentialProvider myCredentialProvider = new MySessionCredentialProvider();
```

使用永久密钥进行本地调试

您可以使用腾讯云的永久密钥来进行开发阶段的本地调试。**由于该方式存在泄漏密钥的风险，请务必在上线前替换为临时密钥的方式。**

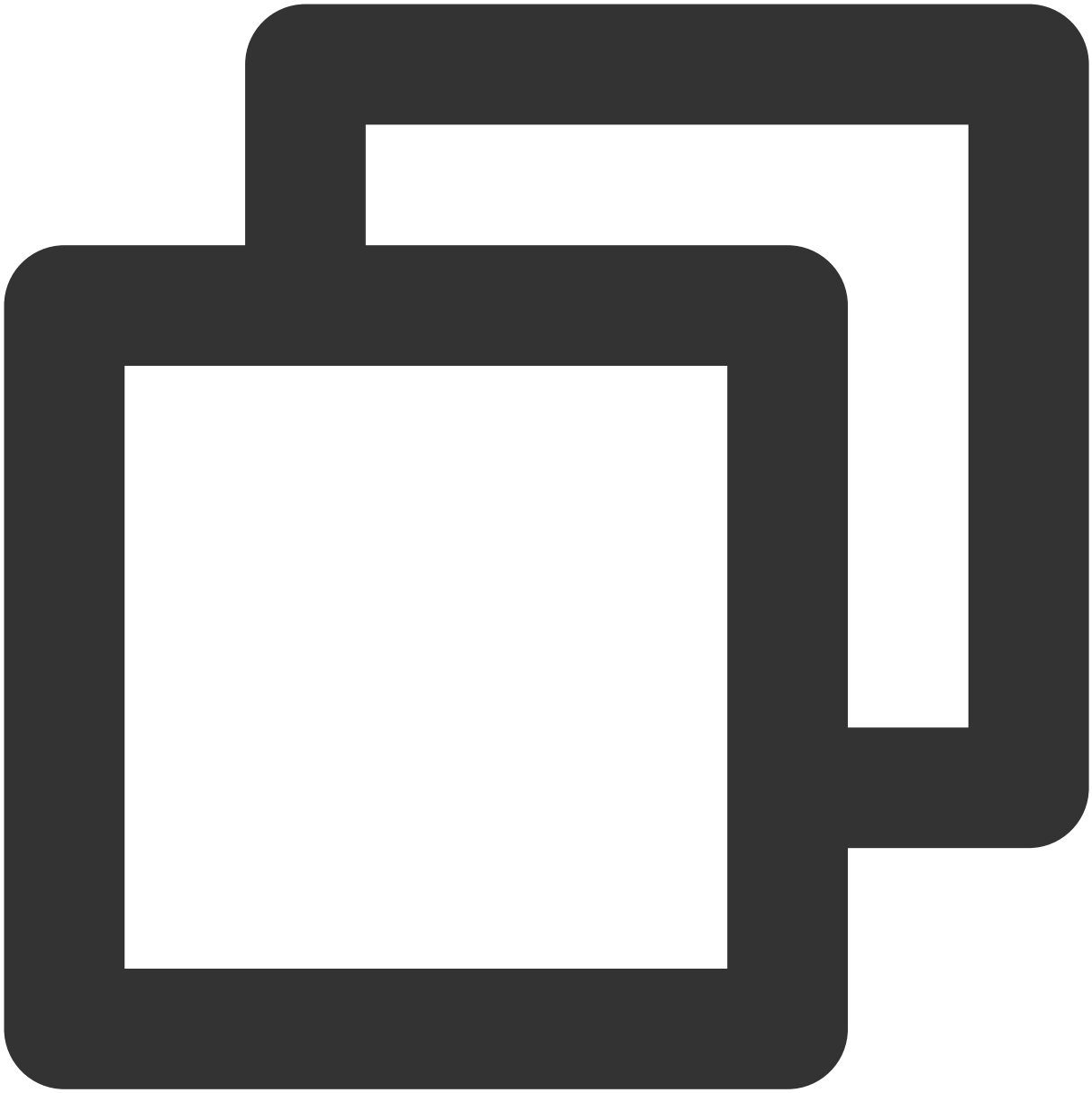


```
String secretId = "SECRETID"; //永久密钥 secretId
String secretKey = "SECRETKEY"; //永久密钥 secretKey

// keyDuration 为请求中的密钥有效期，单位为秒
QCloudCredentialProvider myCredentialProvider =
    new ShortTimeCredentialProvider(secretId, secretKey, 300);
```

使用服务端计算的签名对请求授权

实现一个 `QCloudSelfSigner` 的子类，实现获取服务端签名并加入请求授权。



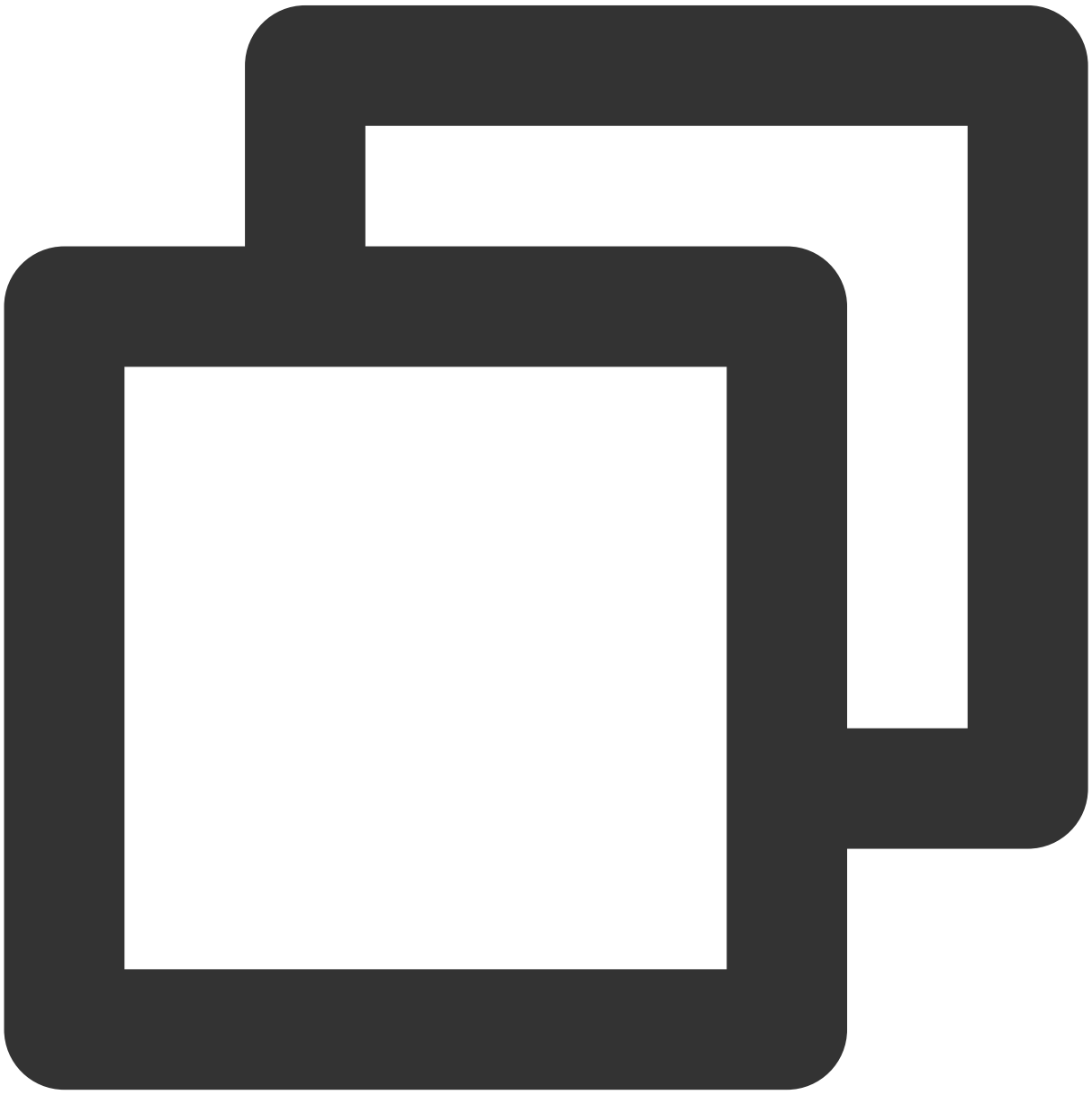
```
QCloudSelfSigner myQCloudSelfSigner = new QCloudSelfSigner() {
    /**
     * 对请求进行签名
     *
     * @param request 需要签名的请求
     * @throws QCloudClientException 客户端异常
     */
    @Override
    public void sign(QCloudHttpRequest request) throws QCloudClientException {
        // 1. 把 request 的请求参数传给服务端计算签名
        String auth = "get auth from server";
```

```
            // 2. 给请求添加签名
            request.addHeader(HttpConstants.Header.AUTHORIZATION, auth);
        }
    });
```

2. 初始化 COS Service

使用您提供密钥的实例 `myCredentialProvider` 或 服务端签名授权实例 `myQCloudSelfSigner`，初始化一个 `CosXmlService` 的实例。

`CosXmlService` 提供了访问 COS 的所有接口，建议作为 **程序单例** 使用。



```
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
String region = "COS_REGION";

// 创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
CosXmlServiceConfig serviceConfig = new CosXmlServiceConfig.Builder()
        .setRegion(region)
        .isHttps(true) // 使用 HTTPS 请求，默认为 HTTP 请求
        .builder();

// 初始化 COS Service，获取实例
CosXmlService cosXmlService = new CosXmlService(context,
    serviceConfig, myCredentialProvider);

// 通过服务端签名授权初始化 COS Service，获取实例
CosXmlService cosXmlService = new CosXmlService(context,
    serviceConfig, myQCloudSelfSigner);
```

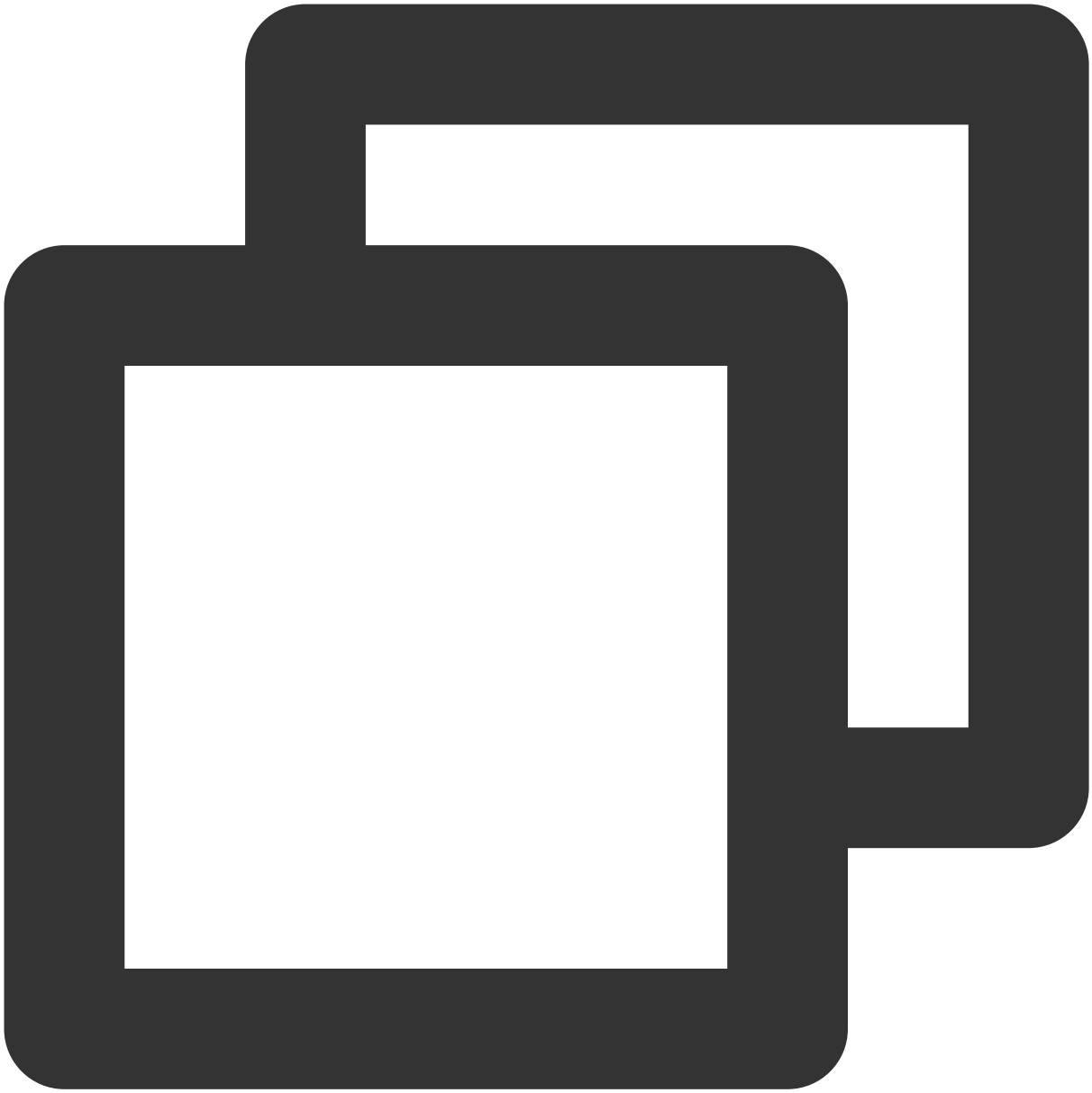
说明

关于存储桶不同地域的简称请参考 地域和访问域名。

第四步：访问 COS 服务

上传对象

SDK 支持上传本地文件、二进制数据、Uri 以及输入流。下面以上传本地文件为例：



```
// 初始化 TransferConfig，这里使用默认配置，如果需要定制，请参考 SDK 接口文档
TransferConfig transferConfig = new TransferConfig.Builder().build();
// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);

// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String srcPath = new File(context.getCacheDir(), "exampleobject")
        .toString(); //本地文件的绝对路径
```

```
//若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传；否则，赋值 null
String uploadId = null;

// 上传文件
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(bucket, cosPath,
        srcPath, uploadId);

//设置初始化分块上传回调(5.9.7版本以及后续版本支持)
cosxmlUploadTask.setInitMultipleUploadListener(new InitMultipleUploadListener() {
    @Override
    public void onSuccess(InitiateMultipartUpload initiateMultipartUpload) {
        //用于下次续传上传的 uploadId
        String uploadId = initiateMultipartUpload.uploadId;
    }
});
//设置上传进度回调
cosxmlUploadTask.setCosXmlProgressListener(new CosXmlProgressListener() {
    @Override
    public void onProgress(long complete, long target) {
        // todo Do something to update progress...
    }
});
//设置返回结果回调
cosxmlUploadTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        COSXMLUploadTask.COSXMLUploadTaskResult uploadResult =
                (COSXMLUploadTask.COSXMLUploadTaskResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
//设置任务状态回调，可以查看任务过程
cosxmlUploadTask.setTransferStateListener(new TransferStateListener() {
    @Override
    public void onStateChanged(TransferState state) {
```

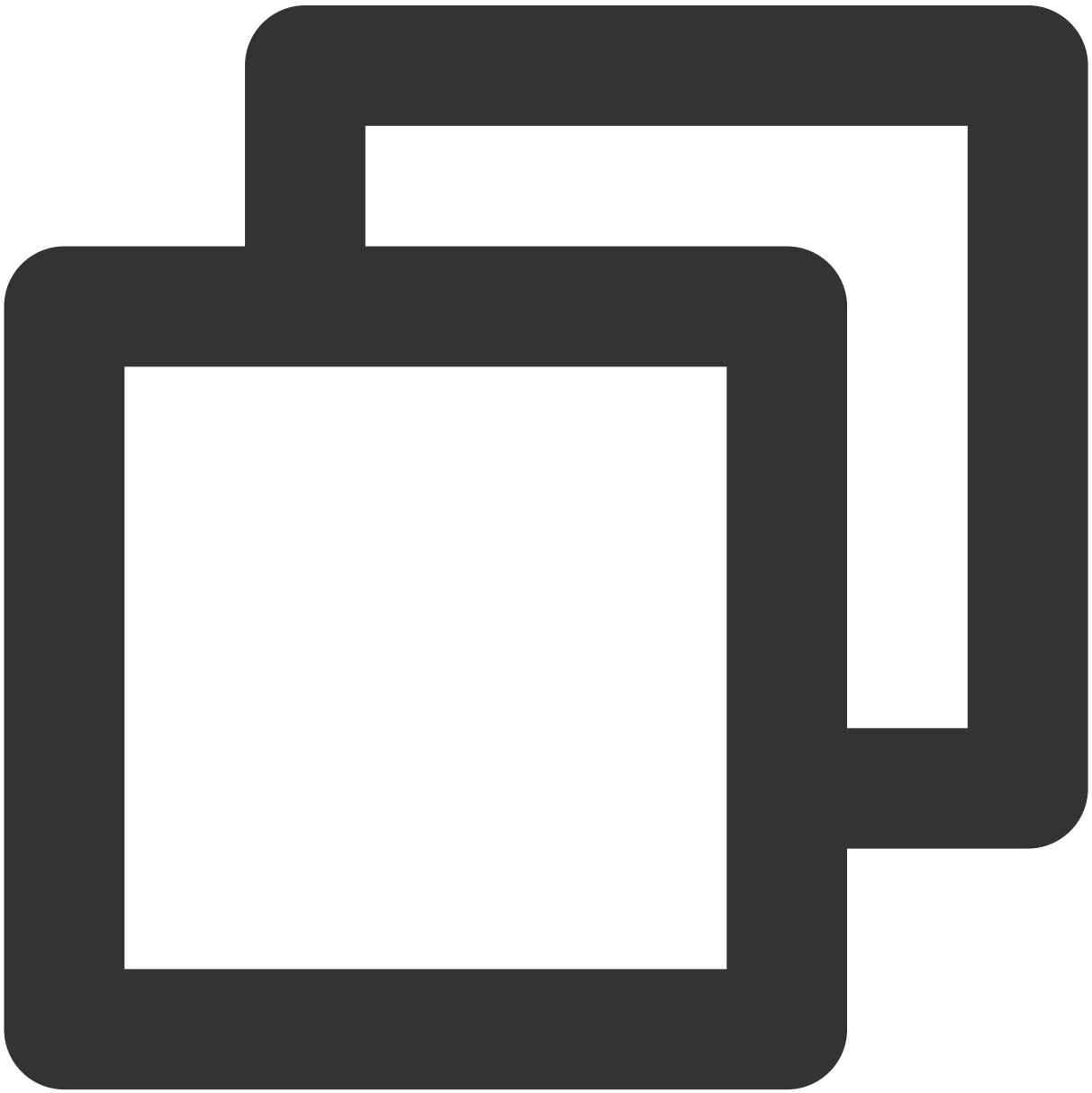
```
            // todo notify transfer state
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 生成预签名链接 文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

下载对象



```
// 高级下载接口支持断点续传，所以会在下载前先发起 HEAD 请求获取文件信息。
// 如果您使用的是临时密钥或者使用子账号访问，请确保权限列表中包含 HeadObject 的权限。

// 初始化 TransferConfig，这里使用默认配置，如果需要定制，请参考 SDK 接口文档
TransferConfig transferConfig = new TransferConfig.Builder().build();
//初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);

// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
//本地目录路径
String savePathDir = context.getExternalCacheDir().toString();
//本地保存的文件名，若不填（null），则与 COS 上的文件名一样
String savedFileName = "exampleobject";

Context applicationContext = context.getApplicationContext(); // application
// context
COSXMLDownloadTask cosxmlDownloadTask =
        transferManager.download(applicationContext,
                bucket, cosPath, savePathDir, savedFileName);

//设置下载进度回调
cosxmlDownloadTask.setCosXmlProgressListener(new CosXmlProgressListener() {
    @Override
    public void onProgress(long complete, long target) {
        // todo Do something to update progress...
    }
});
//设置返回结果回调
cosxmlDownloadTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        COSXMLDownloadTask.COSXMLDownloadTaskResult downloadTaskResult =
                (COSXMLDownloadTask.COSXMLDownloadTaskResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
```

```
                serviceException.printStackTrace();
            }
        }
    });
    //设置任务状态回调，可以查看任务过程
    cosxmlDownloadTask.setTransferStateListener(new TransferStateListener() {
        @Override
        public void onStateChanged(TransferState state) {
            // todo notify transfer state
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

高级下载接口支持断点续传，所以会在下载前先发起 HEAD 请求获取文件信息。如果您使用的是临时密钥或者使用子账号访问，请确保权限列表中包含 HeadObject 的权限。

Android SDK 常见问题

最近更新时间：2024-01-04 15:52:02

客户端网络正常，但是通过 HTTP 访问 COS 非常慢，或者报错 Connection reset，该如何处理？

部分区域的运营商可能会对 COS 的域名进行劫持，因此尽量通过 HTTPS 来访问 COS。

调用完成分块上传接口时没有包含 etag 信息，导致报错 400 BadRequest，该如何处理？

可能是所在的网络过滤了 Etag 头部，SDK 在上传分块后没有解析到对应的参数，导致 SDK 在结束分块上传时报错。

QCloudResultListener 或者其他回调函数没有回调，该如何处理？

如果您是通过日志查看没有回调，可能是输出日志过滤级别太高，或者其他过滤方式将日志过滤了，可以调整过滤规则，或者通过在回调函数中打断点的方式来判断回调情况。

调用接口时报错 NoClassDefFoundError，该如何处理？

SDK 依赖于 bolts 和 okHttp 两个常见的类，如果是这两个类中的方法找不到，那么可能是您自己的项目中也导入了这两个依赖，但是版本较低，尽量使用和 SDK 中相同的版本，或者更高的版本。

SDK 获取手机权限问题，该如何处理？

如果需要在外部存储中上传或者下载文件，那么必须获取网络和外部存储读写权限，其他权限例如定位权限，设备信息权限均不是必须权限，如果对权限问题很敏感，可以不导入 MtaUtils 这个包，或者升级到 5.5.8 及其以上版本。

使用 HTTPS 报错 `java.security.cert.CertPathValidatorException: Trust anchor for certification path not found`，该如何处理？

如果您是通过代理的方式访问 COS，那么首先检查下代理是否支持 HTTPS，否则请 联系我们 处理。

上传进度到了 100%，最终还是回调了 onFailed 接口，该如何处理？

上传进度这里只是代表 SDK 写入到网络中的进度，100% 并不表示上传完成，只有回调 onSuccess 接口才真正上传成功。如果您在最后发送 Complete Multipart Upload 请求时产生了异常，那么会回调 onFailed 接口表示上传失败，您可根据 onFailed 回调的信息查看具体的异常和解决办法。

使用分块上传报错，例如 400 BadRequest、409 Conflict 等错误，该如何处理？

请尽量使用 SDK 提供的高级接口 TransferManager 来上传和下载，不要自己去封装分块上传的接口，否则很容易出错。

通过 TransferManager 上传和下载报错权限问题，该如何处理？

版权所有：腾讯云计算（北京）有限责任公司

TransferManager 下载文件时会先进行 Head 操作，因此需要同时授权 HeadObject 和 GetObject 两个权限，上传时则需要简单上传和分块上传所有接口的权限。

调用接口时，报错 lock timeout 或者 no credential for sign，或者签名过期等错误，该如何处理？

如果您自己实现了 BasicLifecycleCredentialProvider#fetchNewCredentials() 方法，请在这里判断密钥是否及时更新了，或者密钥是否有效，如果是临时密钥则需要带上 token。

上传报错 `java.lang.RuntimeException: Can't create handler inside thread that has not called Looper.prepare()` ，该如何处理？

如果在主线程中调用 TransferManager#upload() 方法进行上传时报该错误，这个是 mta 上报事件误报了，不影响使用。此外您也可以升级到 5.5.8 及其以上版本解决。

在回调中直接操作 ui 导致应用报错，该如何处理？

sdk 回调线程不一定是主线程，请不要直接操作 ui。

上传时报错 calculate md5 error，该如何处理？

可能是您在上传的过程中修改了文件，导致文件的 MD5 值发生了变化，或者网络很差导致服务端收包产生错误。

请求返回 ServerError 错误，该如何处理？

可能是您通过代理访问 COS，但是代理没有做好转发，直接返回了不正确的回包，导致 SDK 解析错误，可以抓包查看客户端接收的回包是否正常。

调用接口报错 403 权限错误，该如何处理？

权限问题一般不是 SDK 的问题，请检查自己授权信息。您也可以 联系我们 处理。

Android SDK 是否支持断点续传？

COS 的 Android SDK 高级接口支持断点续传，您可以参见 上传与复制对象 文档中的高级接口实现。

快速体验

最近更新时间：2024-01-04 15:52:02

背景

移动互联网时代，App 作为移动互联网服务的基础设施，往往需要上传和下载大量的数据，数据的安全性和可靠性尤为重要。现在开发者可以将数据存储相关的问题交给 腾讯云对象存储（Cloud Object Storage，COS）服务，而只需要关心自己应用的业务逻辑即可，可减少很多工作量，提升开发效率。本文主要介绍如何快速搭建一个基于 COS 的应用传输服务，在腾讯云 COS 上实现应用数据的上传下载，在您的服务器上只需要部署您自己的业务、生成和管理临时密钥。
腾讯云 COS 提供 XML 版本的体验 demo，您可以参照本文档来体验 COS 传输实践 demo。

相关资源

本文涉及的 Demo 都存放在 Github 仓库，用户可前往获取。

准备工作

Android 系统版本：4.4 及以上。
腾讯云 APPID、SecretId、SecretKey（可在云 API 密钥 中获取）。

搭建用户客户端

配置客户端

1. 在 Github 仓库 下载项目文件，然后用 IDE 打开。
2. 在环境变量中配置您的 APPID、SecretId、SecretKey。
3. 运行项目，体验 COS 传输实践 demo。

注意

SecretId、SecretKey 明文不要暴露到不安全环境下。
本项目采用的 ShortTimeCredentialProvider 仅仅是为了演示，正式环境中请不要采用此方式，建议采用 通过临时密钥进行授权 的方式。
环境变量更改后，Android Studio 可能需要重启，相关配置才会更新生效。

运行示例 Demo

查询存储桶列表

启动示例 App 后，将展示用户已创建的所有存储桶。

创建存储桶

点击右上角**新建桶**，在配置页面输入桶名称并选择存储桶的所属 地域。

查询对象列表

选择点击某个存储桶，将看到该存储桶内所有文件以及文件夹。

上传文件

在文件列表页面点击右上角的**上传**，然后选择文件进行上传。

下载文件

在对象列表中，点击文件下方的**下载**按钮，即可下载文件。

版权所有：腾讯云计算（北京）有限责任公司

存储桶操作

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于存储桶基本操作的相关 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Service（List Buckets）	查询存储桶列表	查询指定账号下所有的存储桶列表
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

SDK API 参考

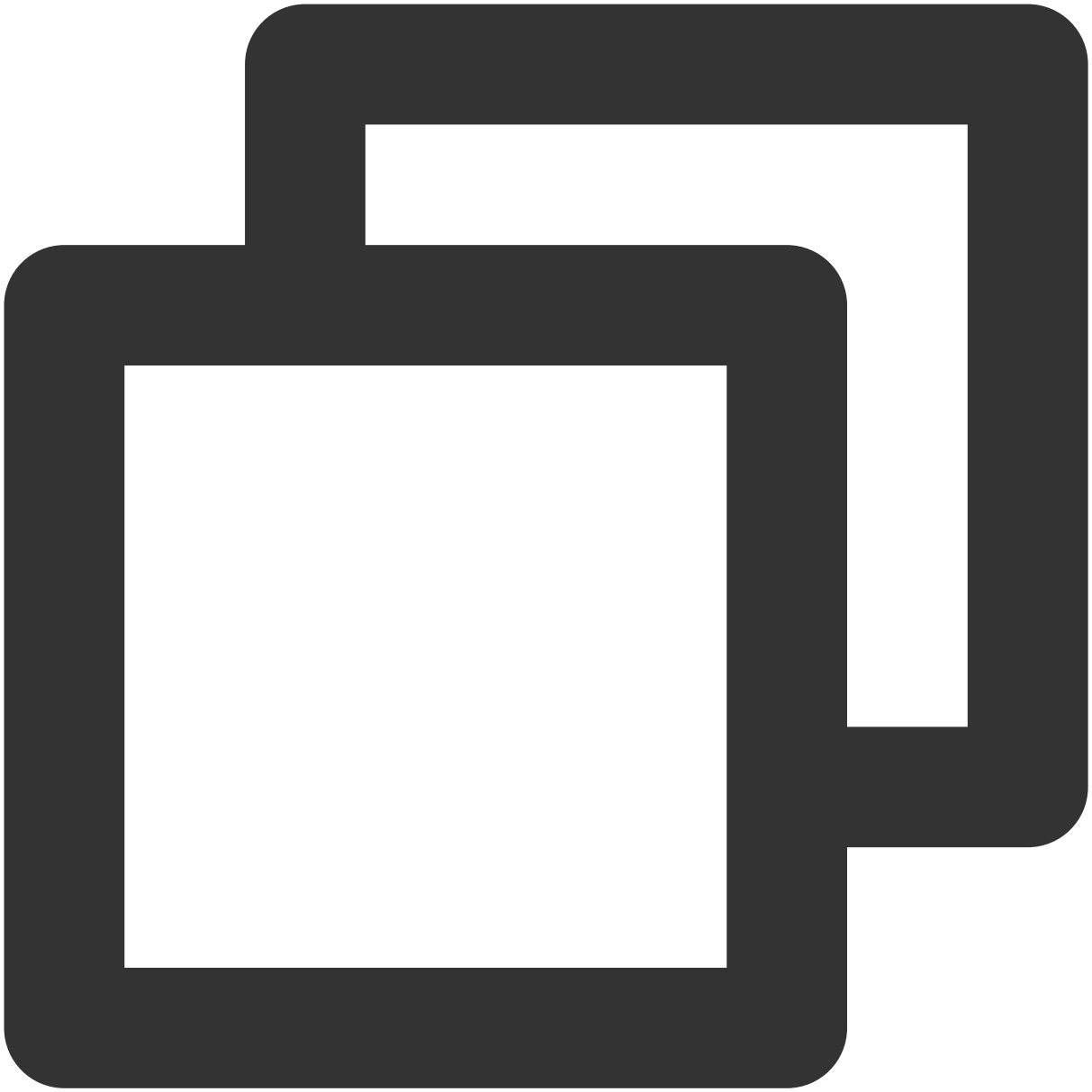
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

查询存储桶列表

功能说明

用于查询指定账号下所有存储桶列表。

示例代码



```
GetServiceRequest getServiceRequest = new GetServiceRequest();
cosXmlService.getServiceAsync(getServiceRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetServiceResult getServiceResult = (GetServiceResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
```

```
                        @Nullable CosXmlClientException clientException,
                        @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

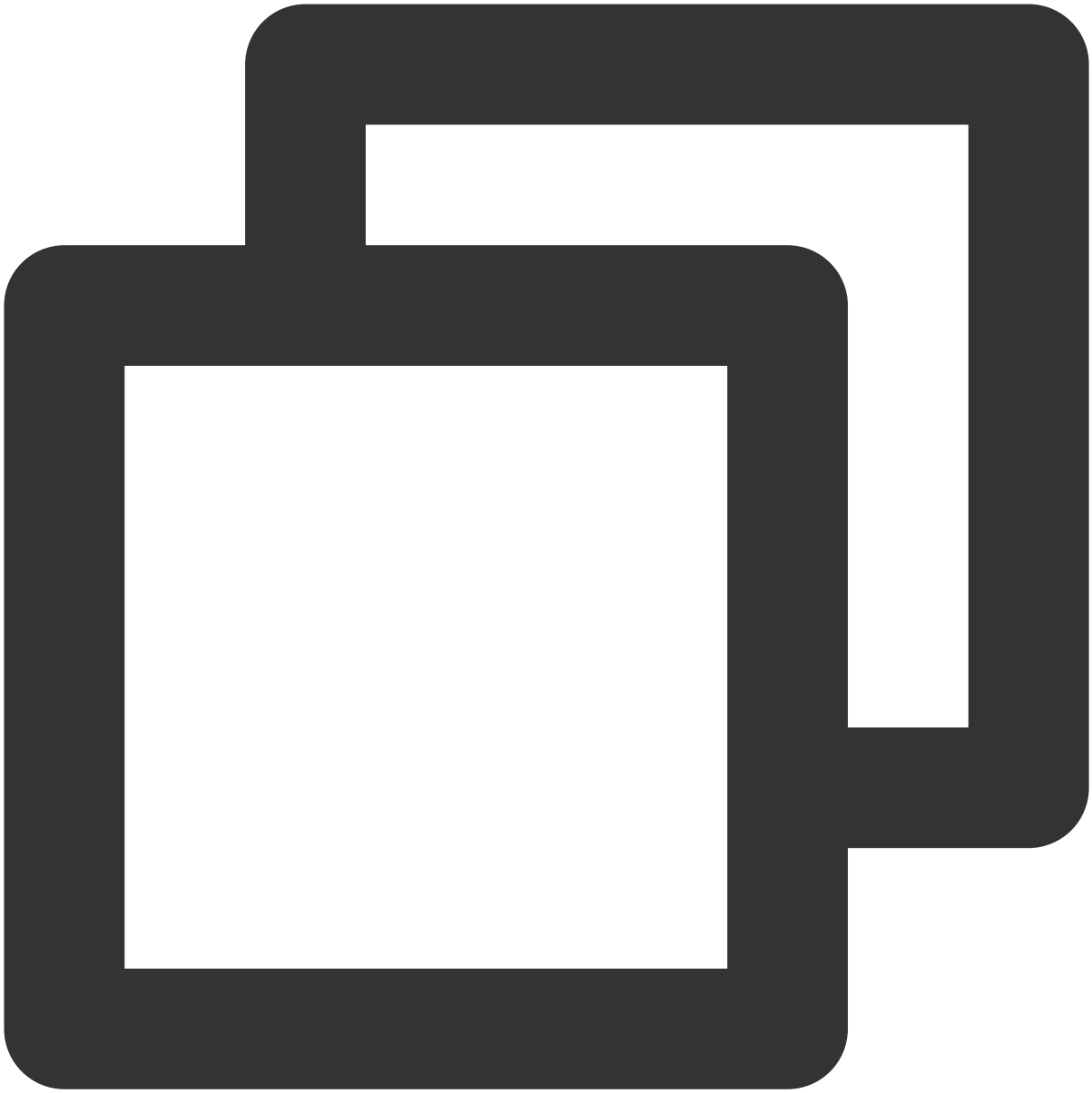
更多完整示例，请前往 GitHub 查看。

创建存储桶

功能说明

创建一个存储桶（PUT Bucket）。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
PutBucketRequest putBucketRequest = new PutBucketRequest(bucket);
cosXmlService.putBucketAsync(putBucketRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutBucketResult putBucketResult = (PutBucketResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
```

```
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

检索存储桶及其权限

功能说明

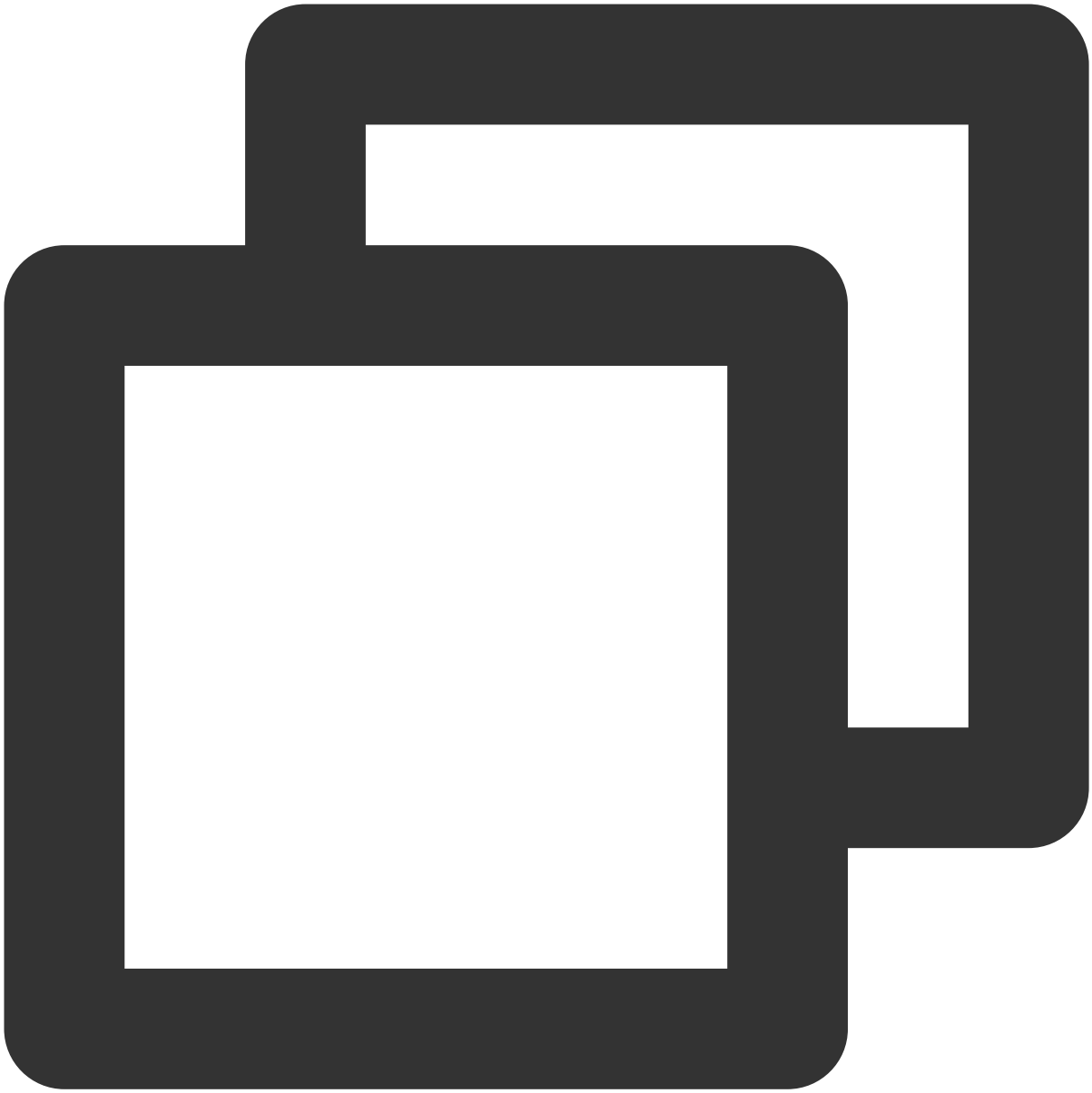
HEAD Bucket 请求可以确认该存储桶是否存在，是否有权限访问。有以下几种情况：

存储桶存在且有读取权限，返回 HTTP 状态码为200。

无存储桶读取权限，返回 HTTP 状态码为403。

存储桶不存在，返回 HTTP 状态码为404。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
HeadBucketRequest headBucketRequest = new HeadBucketRequest(bucket);
cosXmlService.headBucketAsync(headBucketRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        HeadBucketResult headBucketResult = (HeadBucketResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
```

```
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

删除存储桶

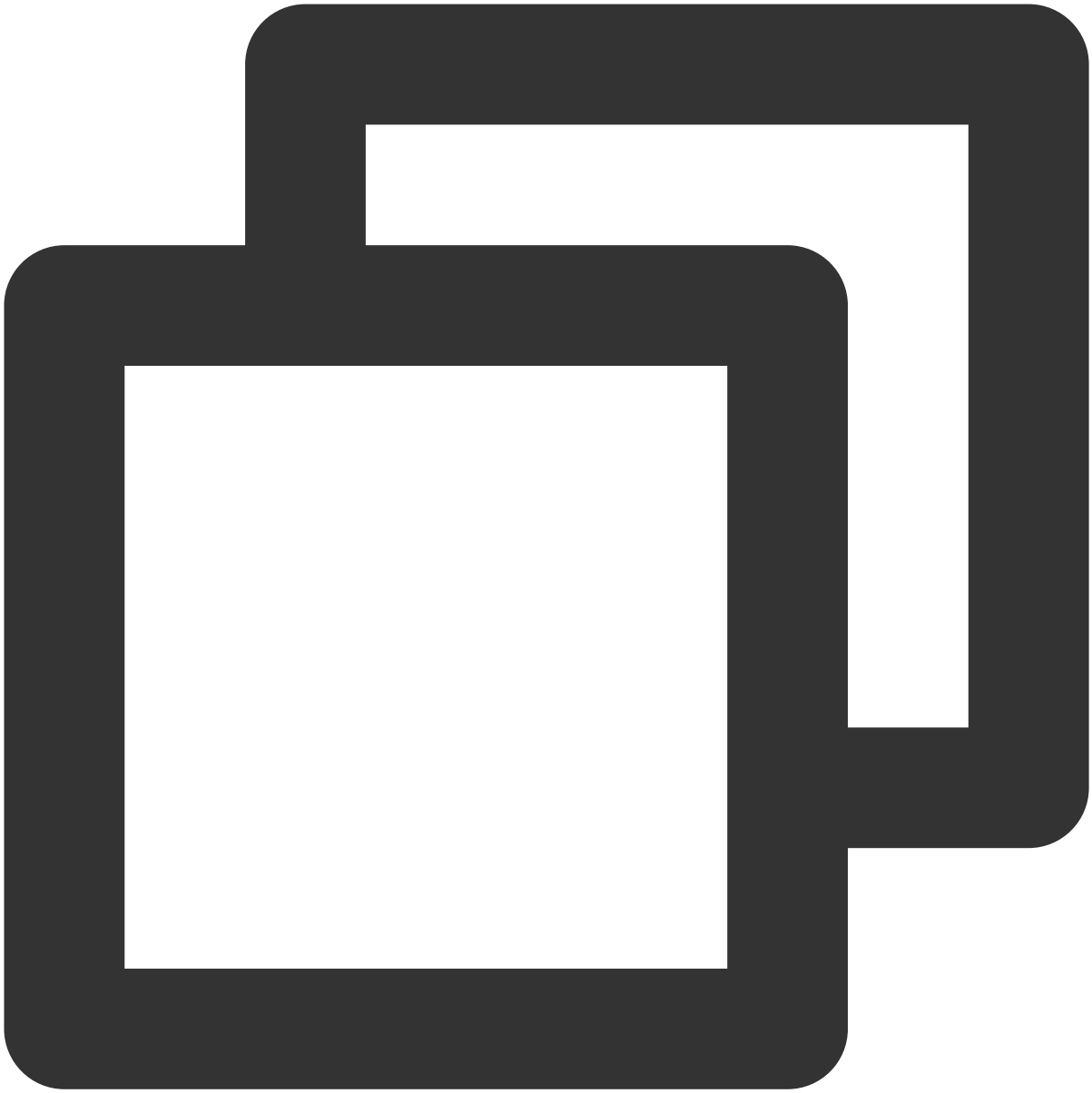
功能说明

删除指定的存储桶（DELETE Bucket）。

注意

删除存储桶前，请确保存储桶内的数据和未完成上传的分块数据已全部清空，否则会无法删除存储桶。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
DeleteBucketRequest deleteBucketRequest = new DeleteBucketRequest(bucket);
cosXmlService.deleteBucketAsync(deleteBucketRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        DeleteBucketResult deleteBucketResult = (DeleteBucketResult) result;
    }

    @Override
```

```
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

对象操作
上传对象

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于对象的上传、复制操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object	简单上传对象	上传一个对象至存储桶

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传操作
Upload Part	上传分块	分块上传对象
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

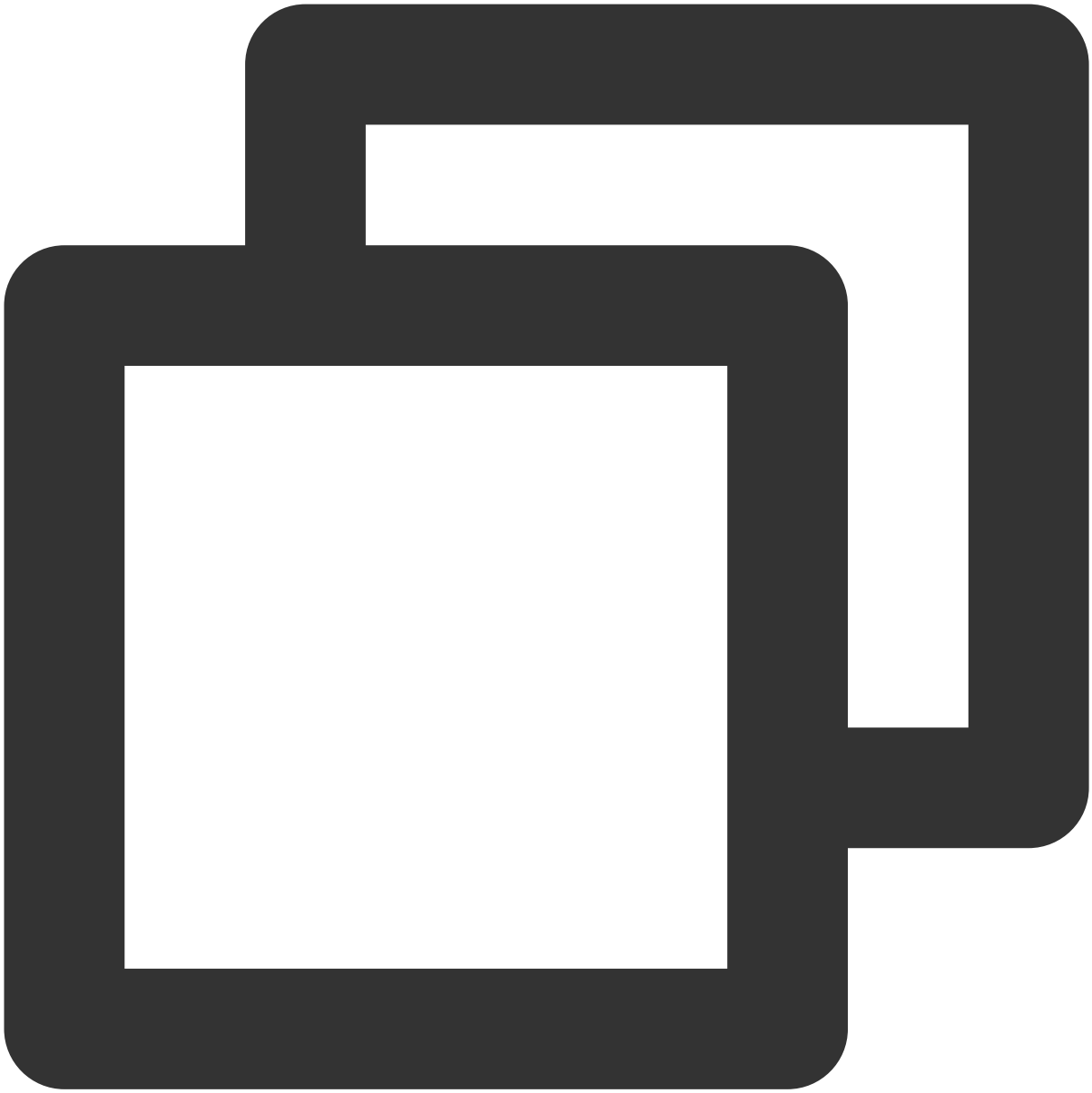
高级接口（推荐）

上传对象

高级接口封装了简单上传、分块上传接口，根据文件大小智能的选择上传方式，同时支持续传功能。

版权所有：腾讯云计算（北京）有限责任公司

示例代码一: 上传本地文件



```
// 初始化 TransferConfig，这里使用默认配置，如果需要定制，请参考 SDK 接口文档
TransferConfig transferConfig = new TransferConfig.Builder().build();
// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);

// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
```

```
String srcPath = new File(context.getCacheDir(), "exampleobject")
        .toString(); //本地文件的绝对路径
//若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传；否则，赋值 null
String uploadId = null;

// 上传文件
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(bucket, cosPath,
        srcPath, uploadId);

//设置初始化分块上传回调(5.9.7版本以及后续版本支持)
cosxmlUploadTask.setInitMultipleUploadListener(new InitMultipleUploadListener() {
    @Override
    public void onSuccess(InitiateMultipartUpload initiateMultipartUpload) {
        //用于下次续传上传的 uploadId
        String uploadId = initiateMultipartUpload.uploadId;
    }
});
//设置上传进度回调
cosxmlUploadTask.setCosXmlProgressListener(new CosXmlProgressListener() {
    @Override
    public void onProgress(long complete, long target) {
        // todo Do something to update progress...
    }
});
//设置返回结果回调
cosxmlUploadTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        COSXMLUploadTask.COSXMLUploadTaskResult uploadResult =
                (COSXMLUploadTask.COSXMLUploadTaskResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
//设置任务状态回调，可以查看任务过程
cosxmlUploadTask.setTransferStateListener(new TransferStateListener() {
```

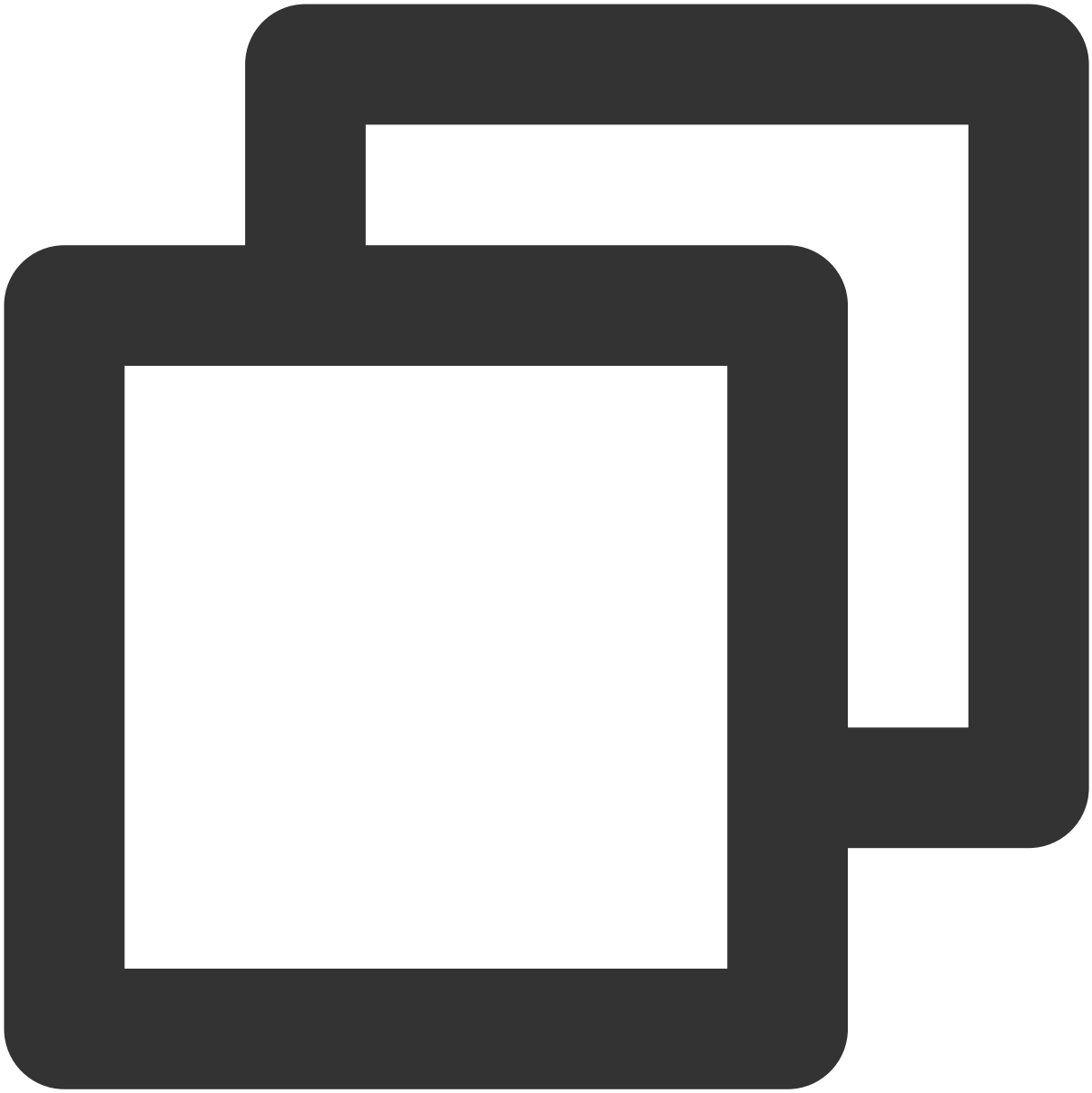
```
    @Override
    public void onStateChanged(TransferState state) {
        // todo notify transfer state
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 生成预签名 URL 文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

示例代码二: 上传二进制数据



```
TransferConfig transferConfig = new TransferConfig.Builder().build();
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);

// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键

// 上传字节数组
byte[] bytes = "this is a test".getBytes(Charset.forName("UTF-8"));
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(bucket, cosPath,
```

```
        bytes);

//设置返回结果回调
cosxmlUploadTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        COSXMLUploadTask.COSXMLUploadTaskResult uploadResult =
                (COSXMLUploadTask.COSXMLUploadTaskResult) result;
    }

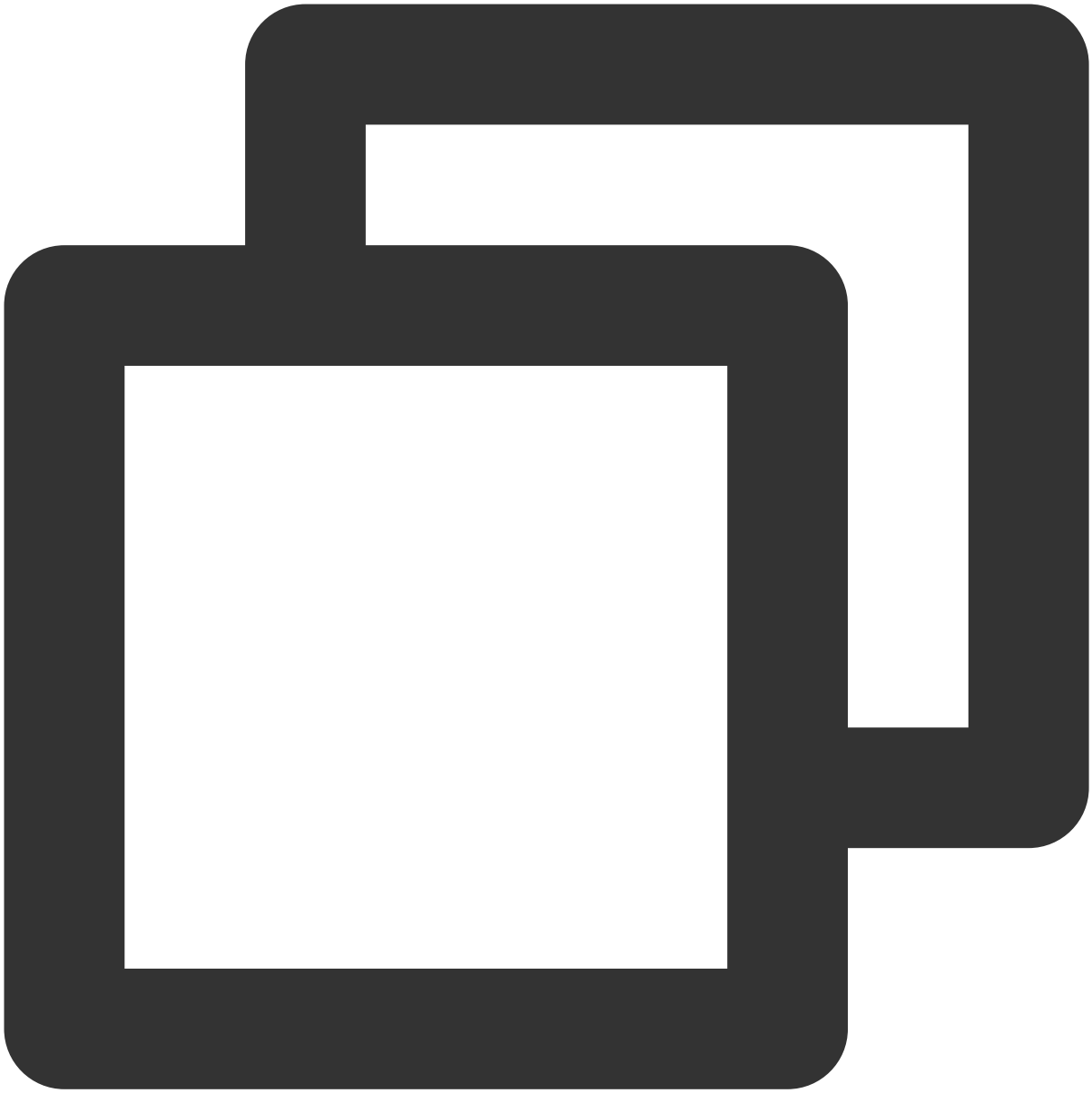
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 生成预签名 URL文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

示例代码三: 流式上传



```
TransferConfig transferConfig = new TransferConfig.Builder().build();
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);

// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键

// 流式上传
InputStream inputStream =
        new ByteArrayInputStream("this is a test".getBytes(Charset.forName(
```

```
                "UTF-8")));
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(bucket, cosPath,
        inputStream);

//设置返回结果回调
cosxmlUploadTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        COSXMLUploadTask.COSXMLUploadTaskResult uploadResult =
                (COSXMLUploadTask.COSXMLUploadTaskResult) result;
    }

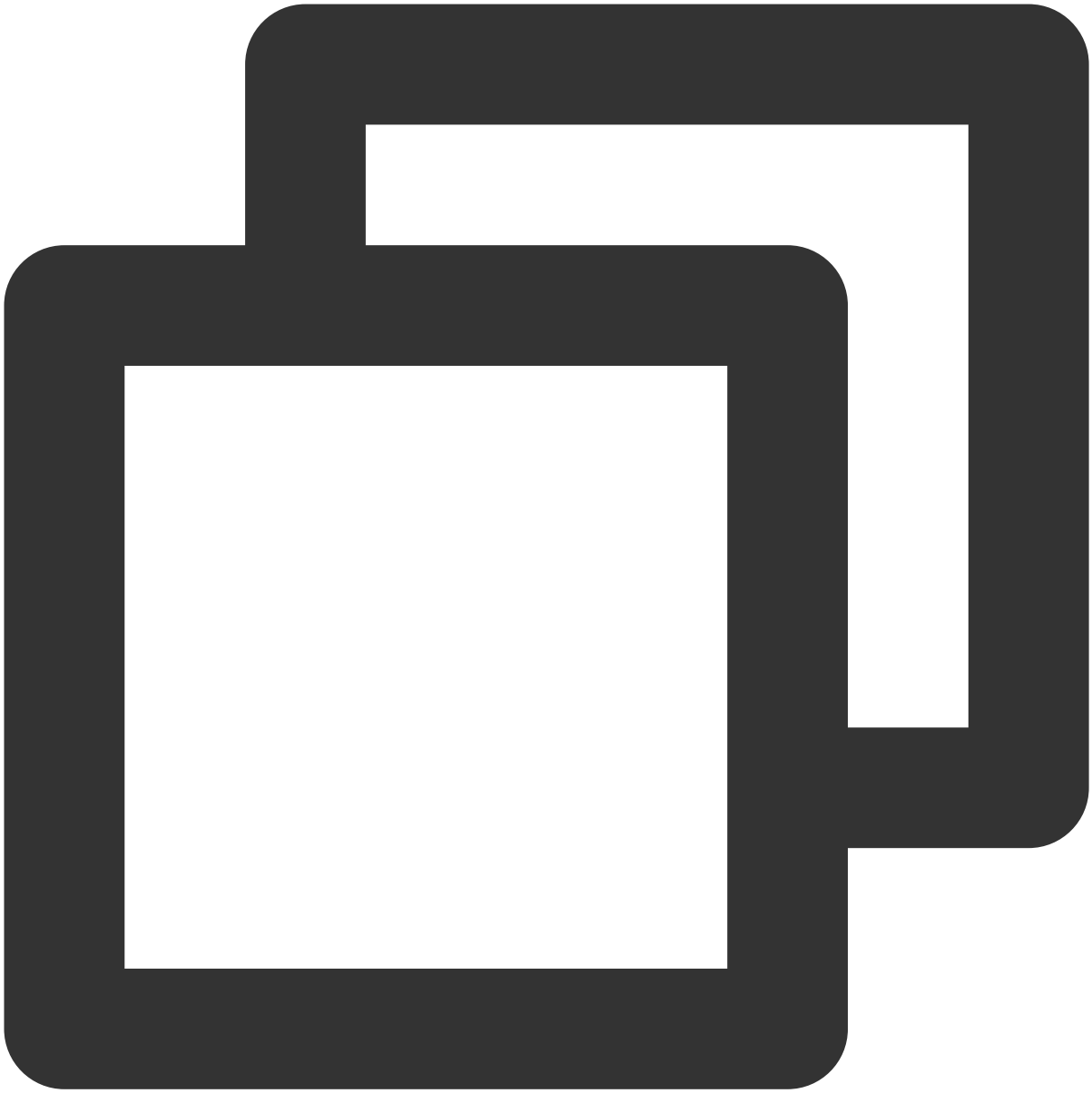
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 生成预签名 URL文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

示例代码四: 通过 URI 上传



```
TransferConfig transferConfig = new TransferConfig.Builder().build();
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);

// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键

// 文件的 Uri
Uri uri = Uri.parse("exampleObject");
// 若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传；否则，赋值 null
```

```
String uploadId = null;
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(bucket, cosPath,
        uri, uploadId);

//设置返回结果回调
cosxmlUploadTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        COSXMLUploadTask.COSXMLUploadTaskResult uploadResult =
                (COSXMLUploadTask.COSXMLUploadTaskResult) result;
    }

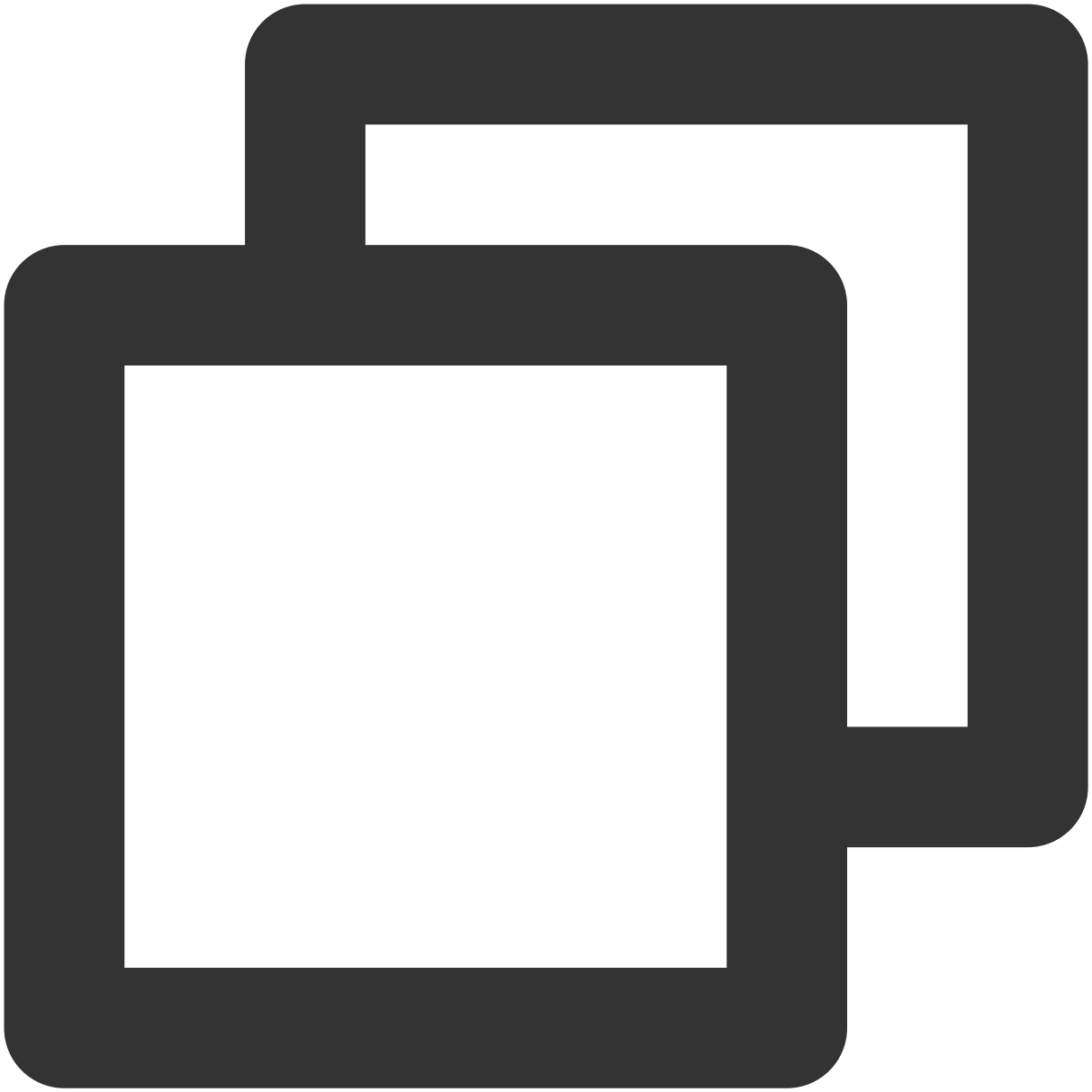
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 生成预签名 URL 文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

示例代码五: 设置智能分块阈值



```
// 默认对大于或等于2M的文件自动进行分块上传，可以通过如下代码修改分块阈值
TransferConfig transferConfig = new TransferConfig.Builder()
        .setDivisionForUpload(2 * 1024 * 1024) // 设置大于等于 2M 的文件进行分块上传
        .build();

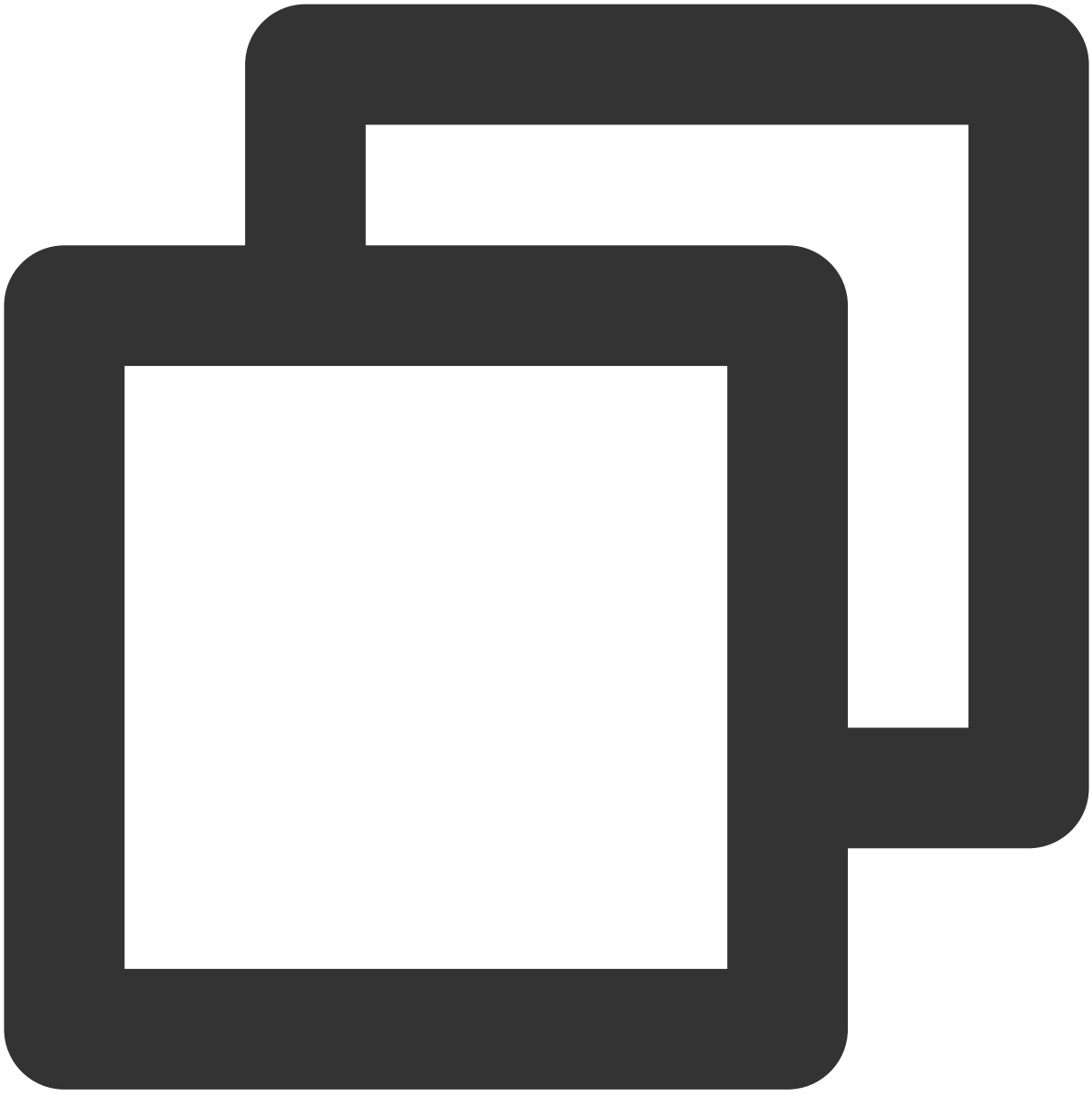
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);
```

说明

更多完整示例，请前往 GitHub 查看。

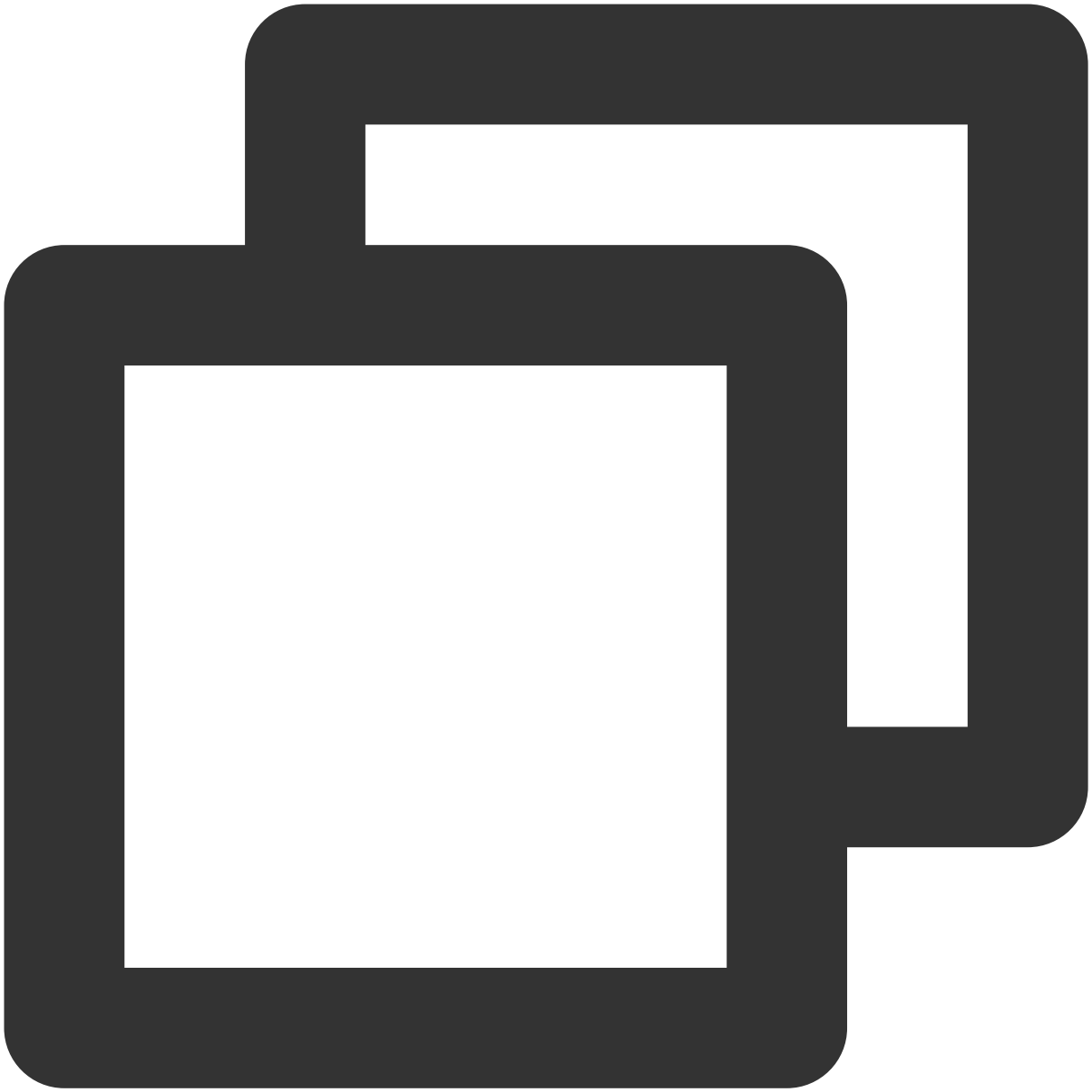
示例代码六: 上传暂停、继续与取消

对于上传任务，可以通过以下方式暂停：



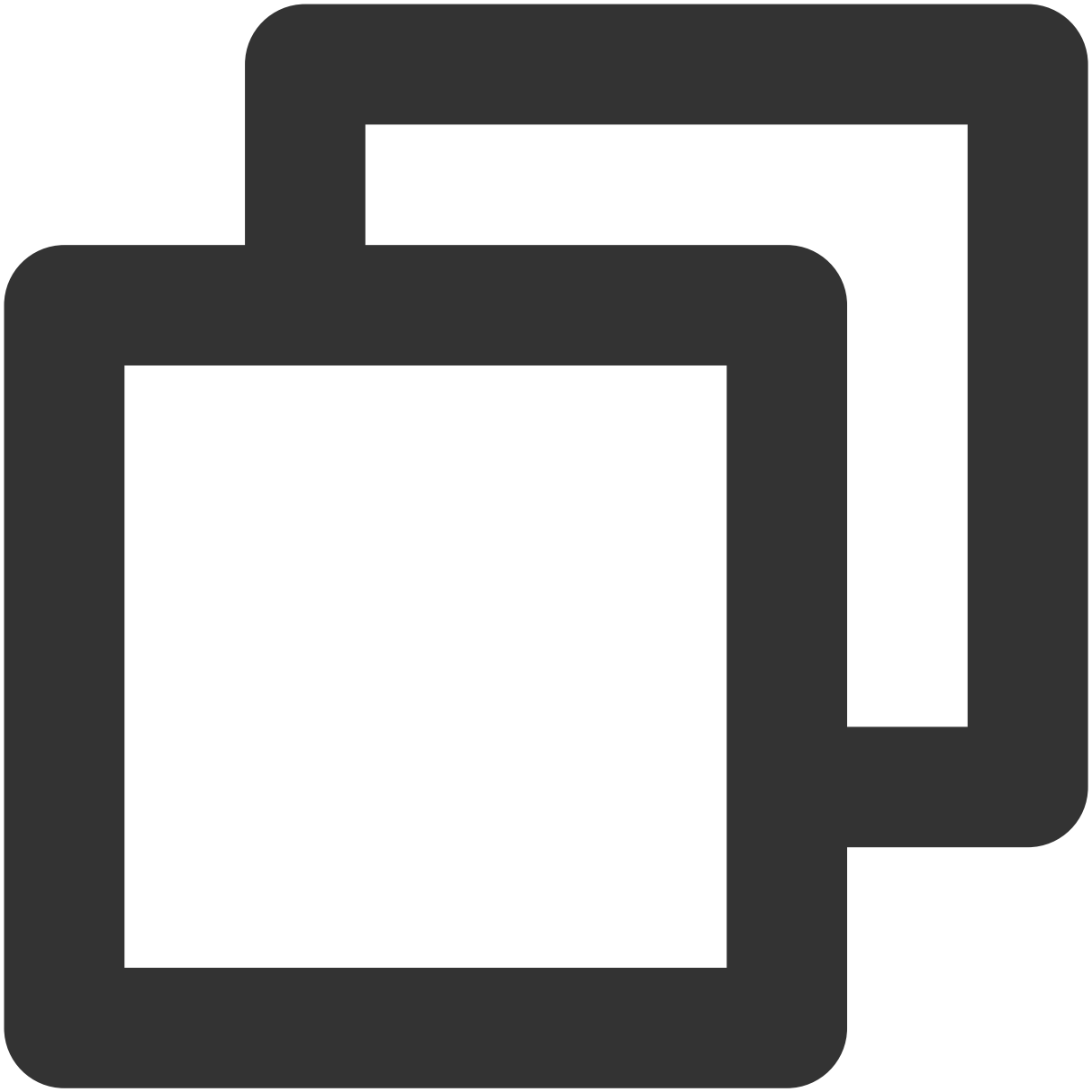
```
// 如果上传过程中，已经发起了最后的 Complete Multipart Upload 请求，那么暂停会失败
boolean pauseSuccess = cosxmlUploadTask.pauseSafely();
```

暂停之后，可以通过以下方式续传：



```
// 如果暂停成功，可以恢复上传
if (pauseSuccess) {
    cosxmlUploadTask.resume();
}
```

也通过以下方式取消上传：

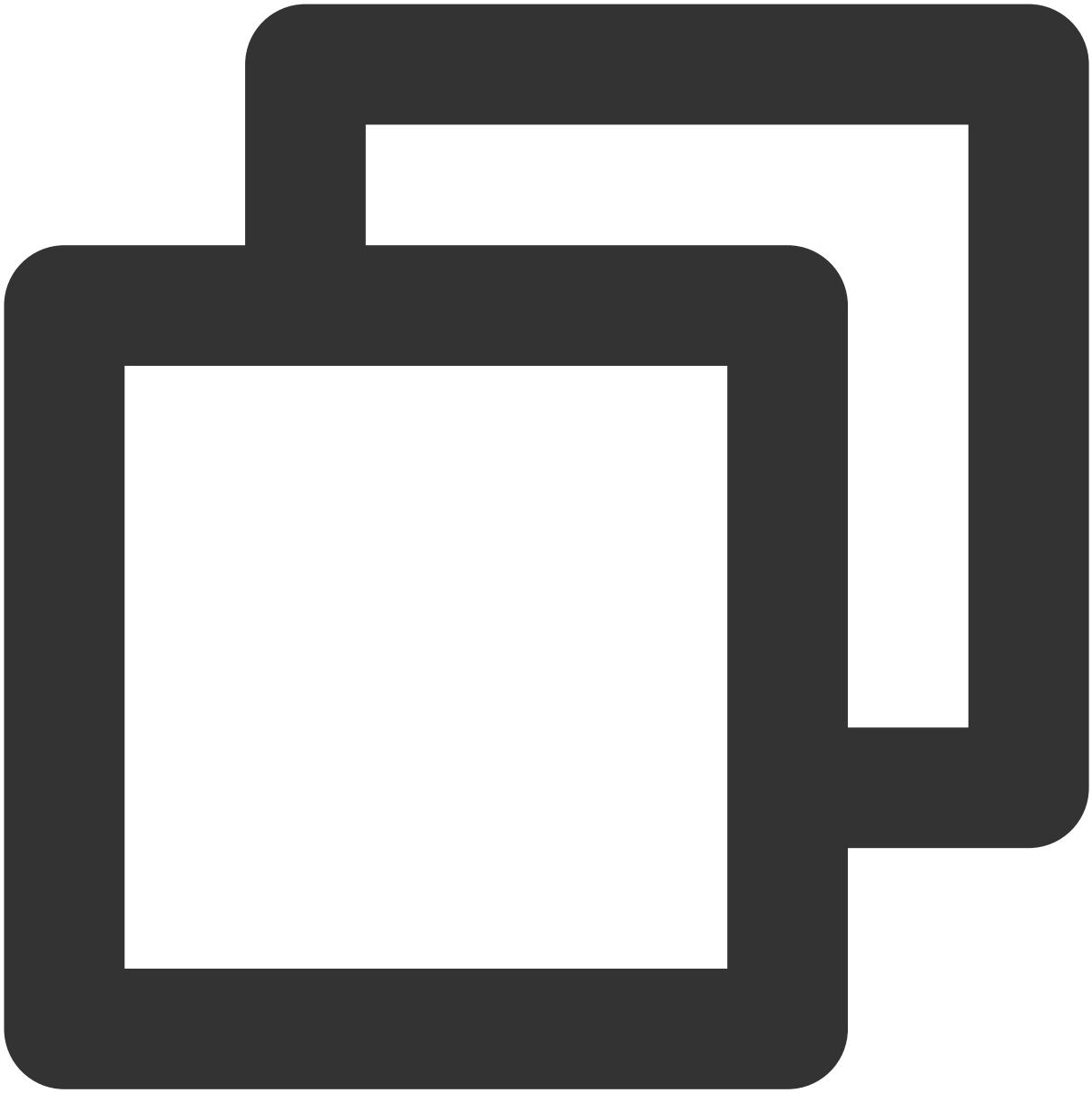


```
cosxmlUploadTask.cancel();
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码七: 批量上传



```
//本地文件的绝对路径
File[] files = new File(context.getCacheDir(), "exampleDirectory").listFiles();

// 开始批量上传
for (File file : files) {
    //若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传；否则，赋值 null
    String uploadId = null;

    // 上传文件
    COSXMLUploadTask cosxmlUploadTask = transferManager.upload(bucket, cosPath,
            file.getAbsolutePath(), uploadId);
```

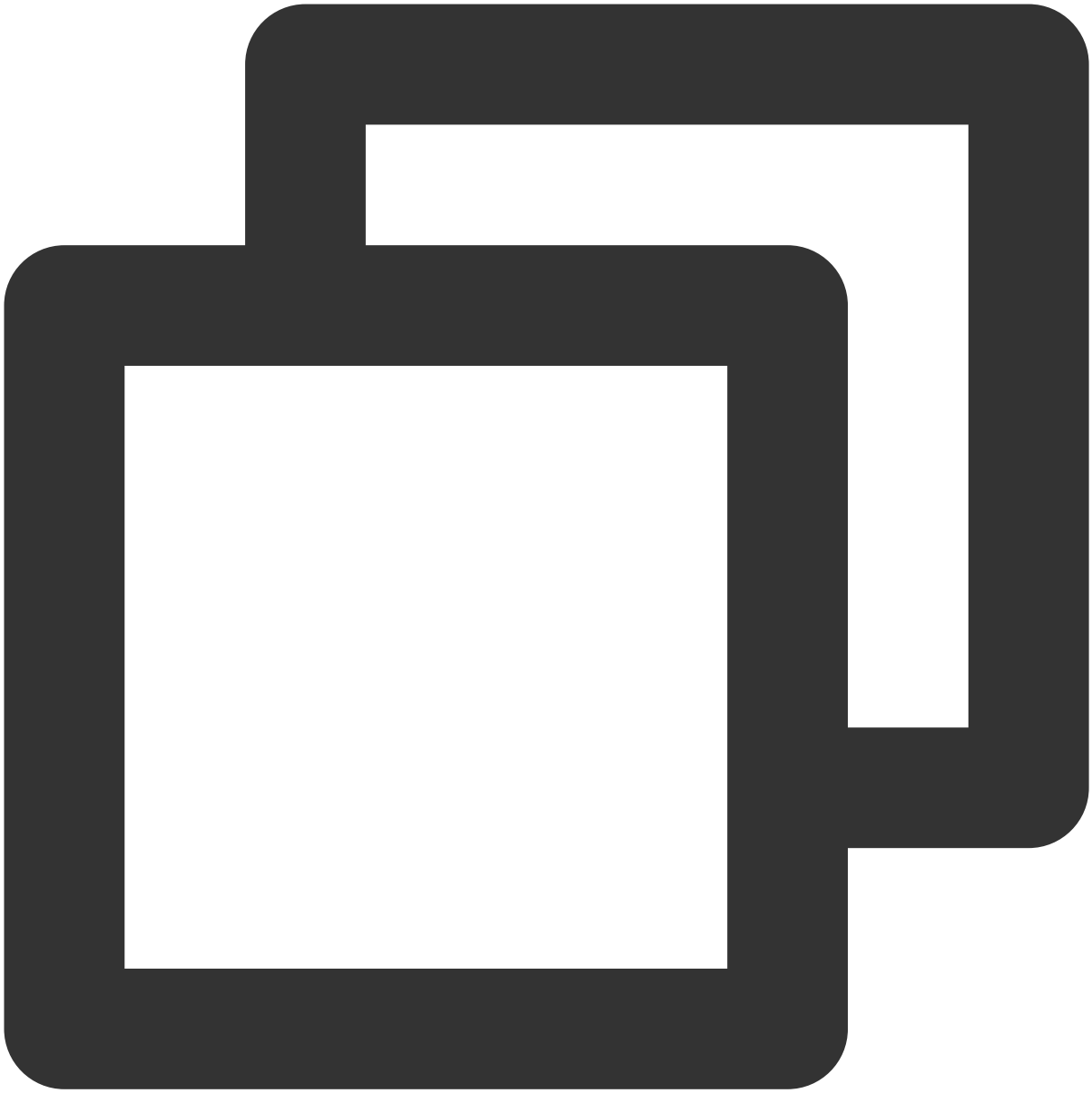
```
    //设置返回结果回调
    cosxmlUploadTask.setCosXmlResultListener(new CosXmlResultListener() {
        @Override
        public void onSuccess(CosXmlRequest request, CosXmlResult result) {
            COSXMLUploadTask.COSXMLUploadTaskResult uploadResult =
                    (COSXMLUploadTask.COSXMLUploadTaskResult) result;
        }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
}
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码八：创建目录



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
// 文件夹在存储桶中的位置标识符，即称对象键，必须以 '/' 结尾
String cosPath = "exampleobject/";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, new byte[
cosXmlService.putObjectAsync(putObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutObjectResult putObjectResult =
                (PutObjectResult) result;
    }
```

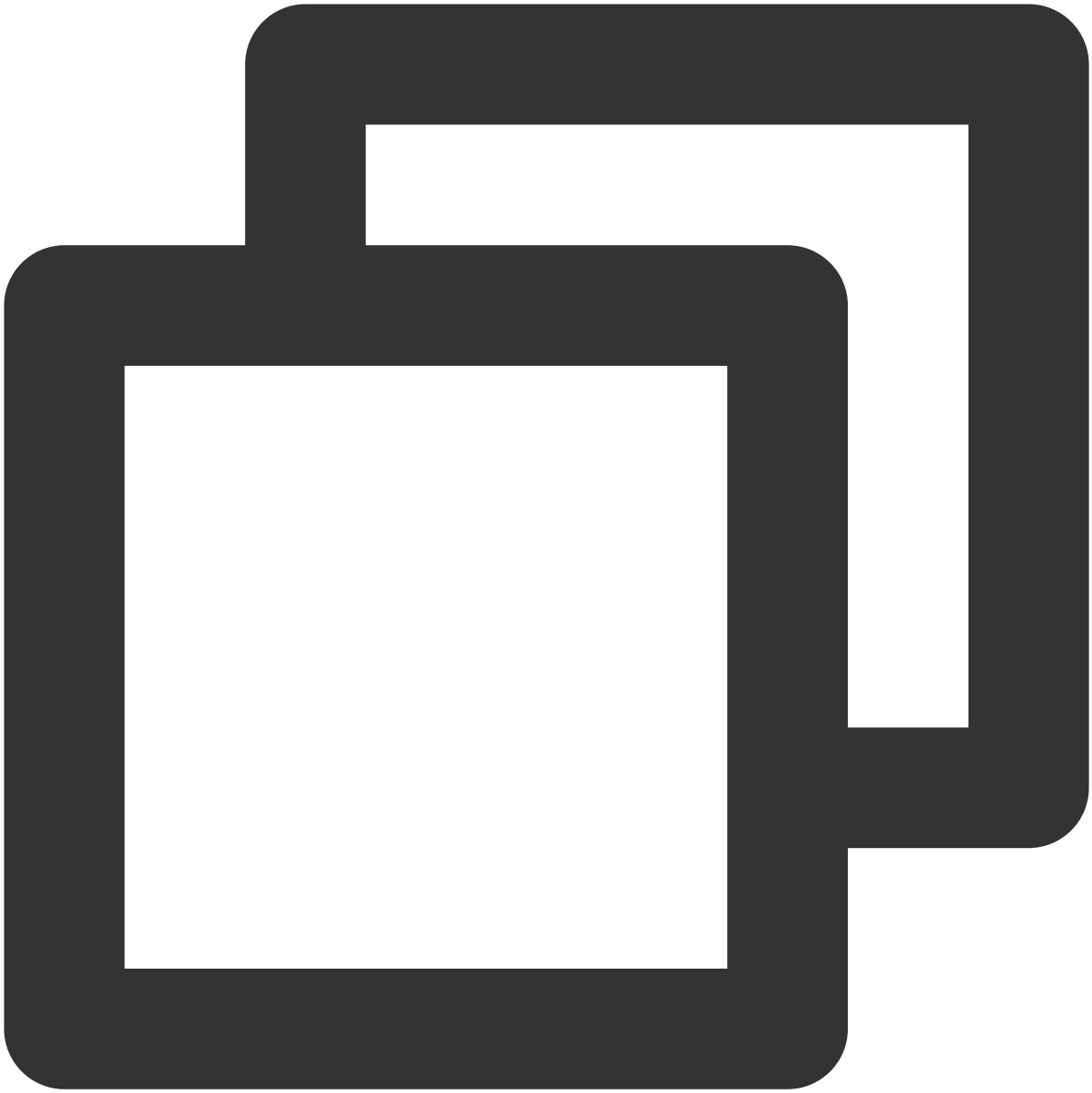
```
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 生成预签名 URL 文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

示例代码九：设置低优先级任务



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
// 文件夹在存储桶中的位置标识符，即称对象键，必须以 '/' 结尾
String cosPath = "exampleobject/";
String srcPath = new File(context.getCacheDir(), "exampleobject")
        .toString(); //本地文件的绝对路径
PutObjectRequest putObjectRequest= new PutObjectRequest(bucket, cosPath, srcPath);
putObjectRequest.setPriorityLow(); // 设置为低优先级上传任务
final COSXMLUploadTask uploadTask = transferManager.upload(putObjectRequest, null);
uploadTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
```

```
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {}


    @Override
    public void onFail(CosXmlRequest request, CosXmlClientException clientException
});
```

说明

更多完整示例，请前往 GitHub 查看。

简单操作

简单上传对象

功能说明

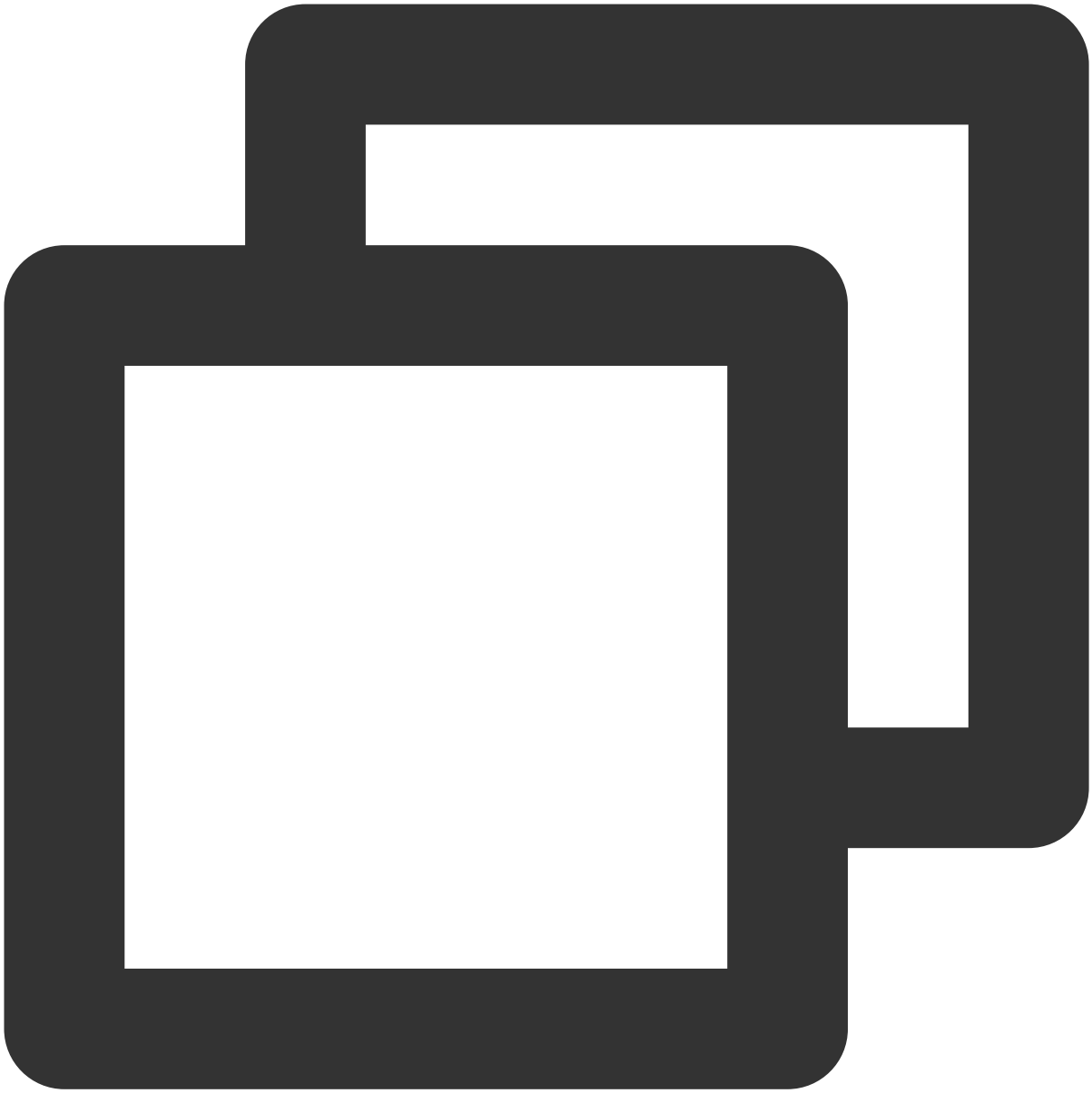
PUT Object 接口可以上传一个对象至指定存储桶中，该操作需要请求者对存储桶有 WRITE 权限。最大支持上传不超过5GB的对象，5GB以上对象请使用 分块上传 或 高级接口 上传。

注意

Key（文件名）不能以 `/` 结尾，否则会被识别为文件夹。

每个主账号（即同一个 APPID），存储桶的 ACL 规则数量最多为1000条，对象 ACL 规则数量不限制。如果您不需要进行对象 ACL 控制，请在上传时不要设置，默认继承存储桶权限。

示例代码



```
// 存储桶名称，由 bucketname-appid 组成，appid必须填入，可以在 COS 控制台查看存储桶名称。 htt
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象位于存储桶中的位置标识符，即对象键。
String srcPath = new File(context.getCacheDir(), "exampleobject")
        .toString();//"本地文件的绝对路径";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath,
        srcPath);

putObjectRequest.setProgressListener(new CosXmlProgressListener() {
    @Override
    public void onProgress(long progress, long max) {
```

```
            // todo Do something to update progress...
        }
    });
    cosXmlService.putObjectAsync(putObjectRequest, new CosXmlResultListener() {
        @Override
        public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
            PutObjectResult putObjectResult = (PutObjectResult) result;
        }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest cosXmlRequest,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

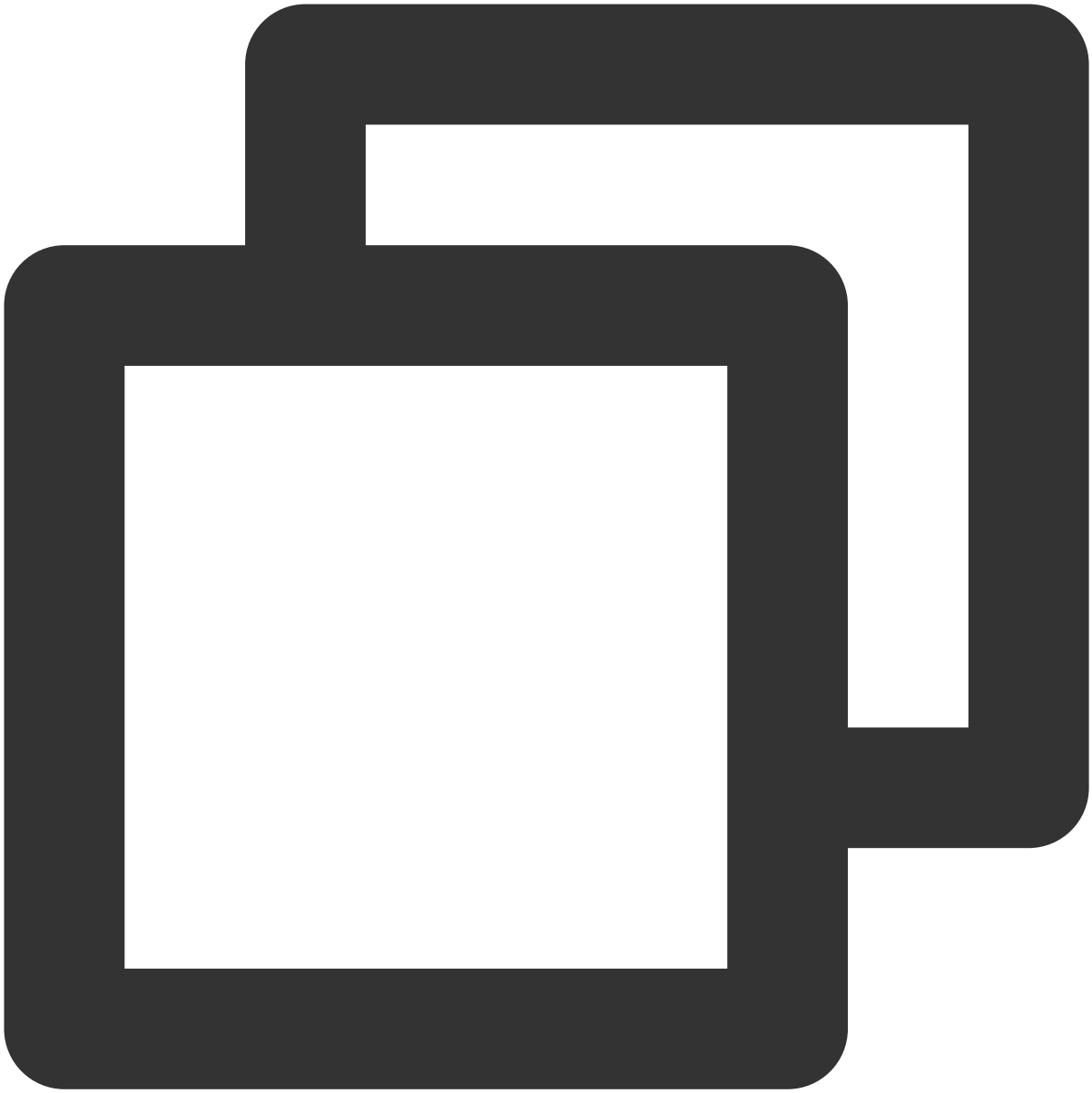
上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 生成预签名 URL文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

表单上传对象

功能说明

使用表单请求上传对象。

示例代码



```
String bucket = "examplebucket-1250000000"; //存储桶名称，格式：BucketName-APPID
String cosPath = "exampleobject"; //对象位于存储桶中的位置标识符，即对象键。
String srcPath = new File(context.getCacheDir(), "exampleobject")
        .toString();//"本地文件的绝对路径";

PostObjectRequest postObjectRequest = new PostObjectRequest(bucket, cosPath,
        srcPath);

postObjectRequest.setProgressListener(new CosXmlProgressListener() {
    @Override
    public void onProgress(long progress, long max) {
```

```
            // todo Do something to update progress...
        }
    });
    cosXmlService.postObjectAsync(postObjectRequest, new CosXmlResultListener() {
        @Override
        public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
            PutObjectResult putObjectResult = (PutObjectResult) result;
        }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest cosXmlRequest,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

分块操作

这里说明下分块上传的流程。

分块上传的流程

1. 初始化分块上传（Initiate Multipart Upload），得到 UploadId
2. 使用 UploadId 上传分块（Upload Part），或者复制分块（Upload Part Copy）
3. 完成分块上传（Complete Multipart Upload）

分块继续上传的流程

1. 如果没有记录 UploadId，查询分块上传任务（List Multipart Uploads），得到对应文件的 UploadId
2. 使用 UploadId 列出已上传的分块（List Parts）
3. 使用 UploadId 上传剩余的分块（Upload Part），或者复制剩余的分块（Upload Part Copy）
4. 完成分块上传（Complete Multipart Upload）

终止分块上传的流程

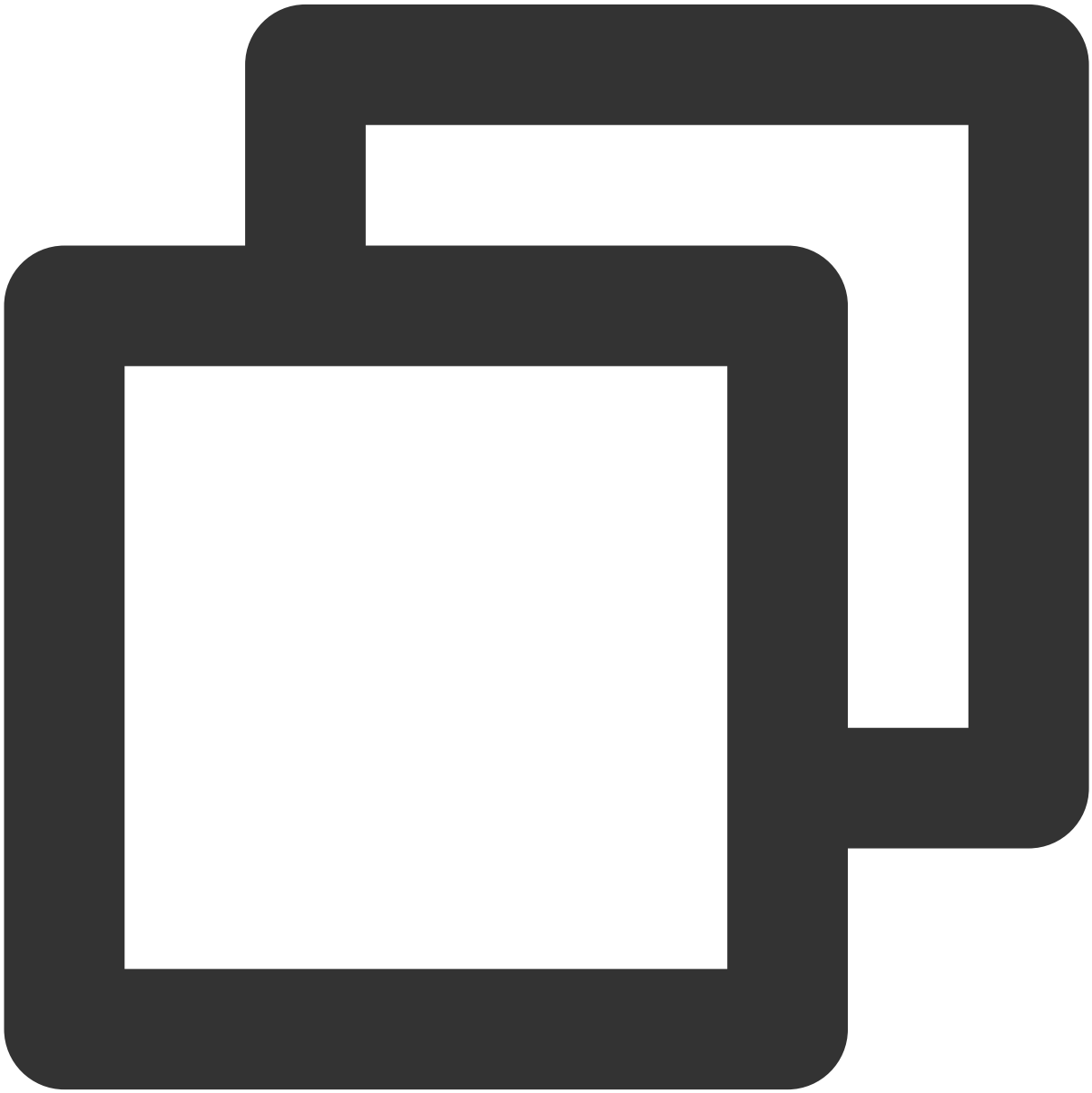
1. 如果没有记录 UploadId，查询分块上传任务（List Multipart Uploads），得到对应文件的 UploadId
2. 终止分块上传并删除已上传分块（Abort Multipart Upload）

查询分块上传

功能说明

查询指定存储桶中正在进行的分块上传（List Multipart Uploads）。

示例代码



```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
ListMultiUploadsRequest listMultiUploadsRequest =
        new ListMultiUploadsRequest(bucket);
cosXmlService.listMultiUploadsAsync(listMultiUploadsRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
        ListMultiUploadsResult listMultiUploadsResult =
                (ListMultiUploadsResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

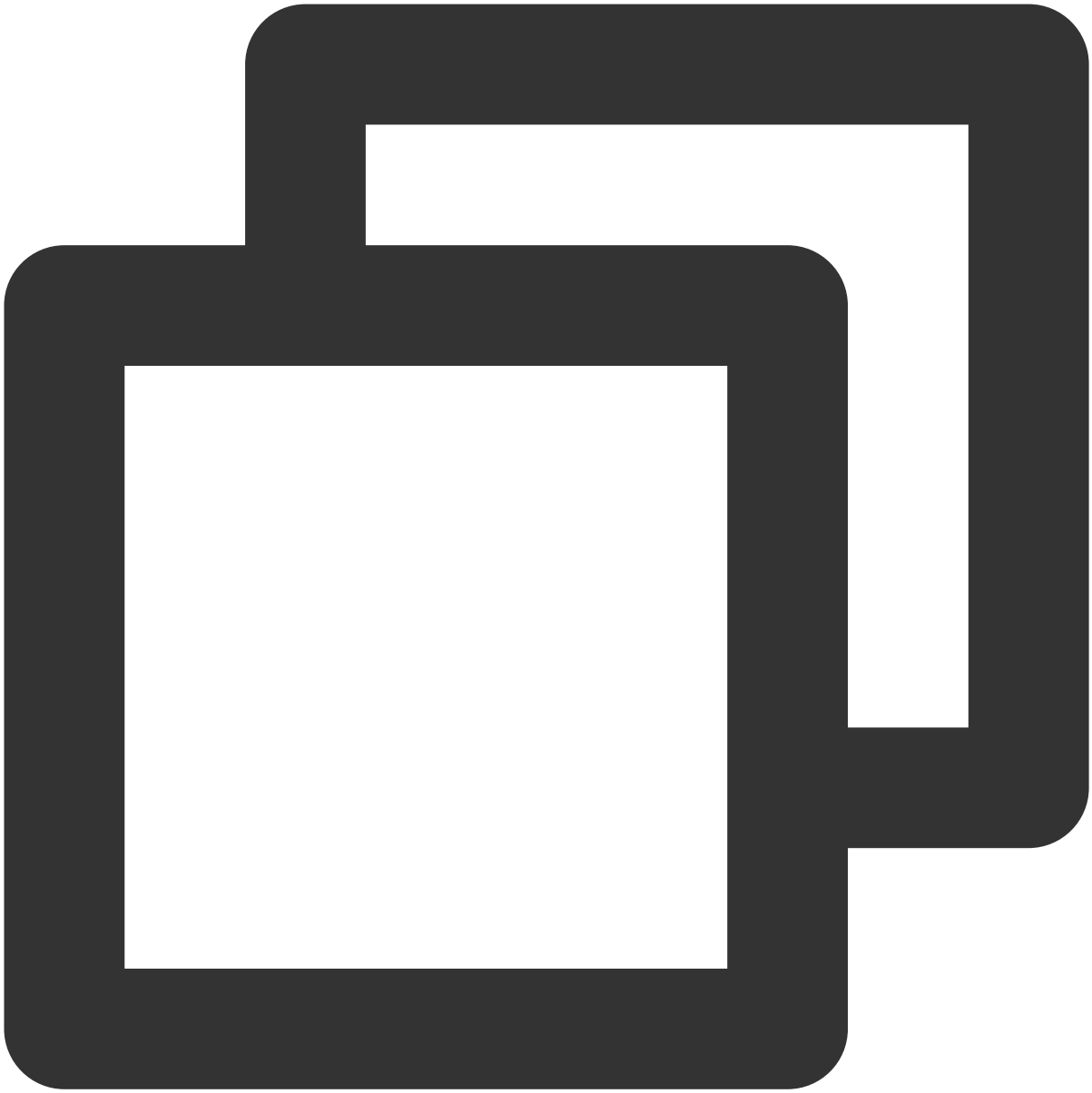
更多完整示例，请前往 GitHub 查看。

初始化分块上传

功能说明

初始化 Multipart Upload 上传操作，获取对应的 uploadId（Initiate Multipart Upload）。

示例代码



```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键。

InitMultipartUploadRequest initMultipartUploadRequest =
        new InitMultipartUploadRequest(bucket, cosPath);
cosXmlService.initMultipartUploadAsync(initMultipartUploadRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
        // 分片上传的 uploadId
```

```
            uploadId = ((InitMultipartUploadResult) result)
                    .initMultipartUpload.uploadId;
        }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest cosXmlRequest,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

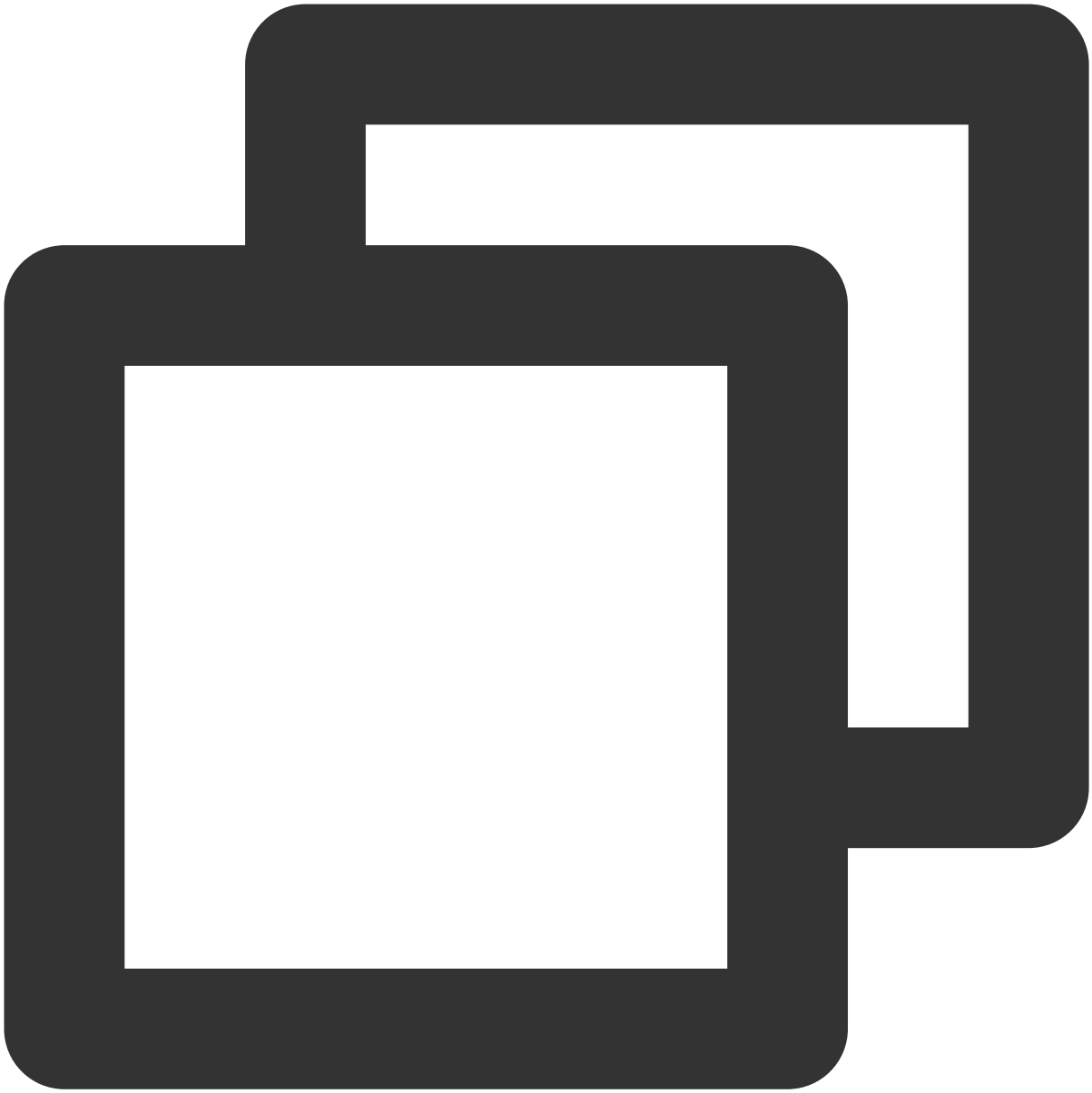
说明

更多完整示例，请前往 GitHub 查看。

上传分块

分块上传对象（Upload Part）。

示例代码



```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键
UploadPartRequest uploadPartRequest = new UploadPartRequest(bucket, cosPath,
        partNumber, srcFile.getPath(), offset, PART_SIZE, uploadId);

uploadPartRequest.setProgressListener(new CosXmlProgressListener() {
    @Override
    public void onProgress(long progress, long max) {
        // todo Do something to update progress...
    }
```

```
});

cosXmlService.uploadPartAsync(uploadPartRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
        String eTag = ((UploadPartResult) result).eTag;
        eTags.put(partNumber, eTag);
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

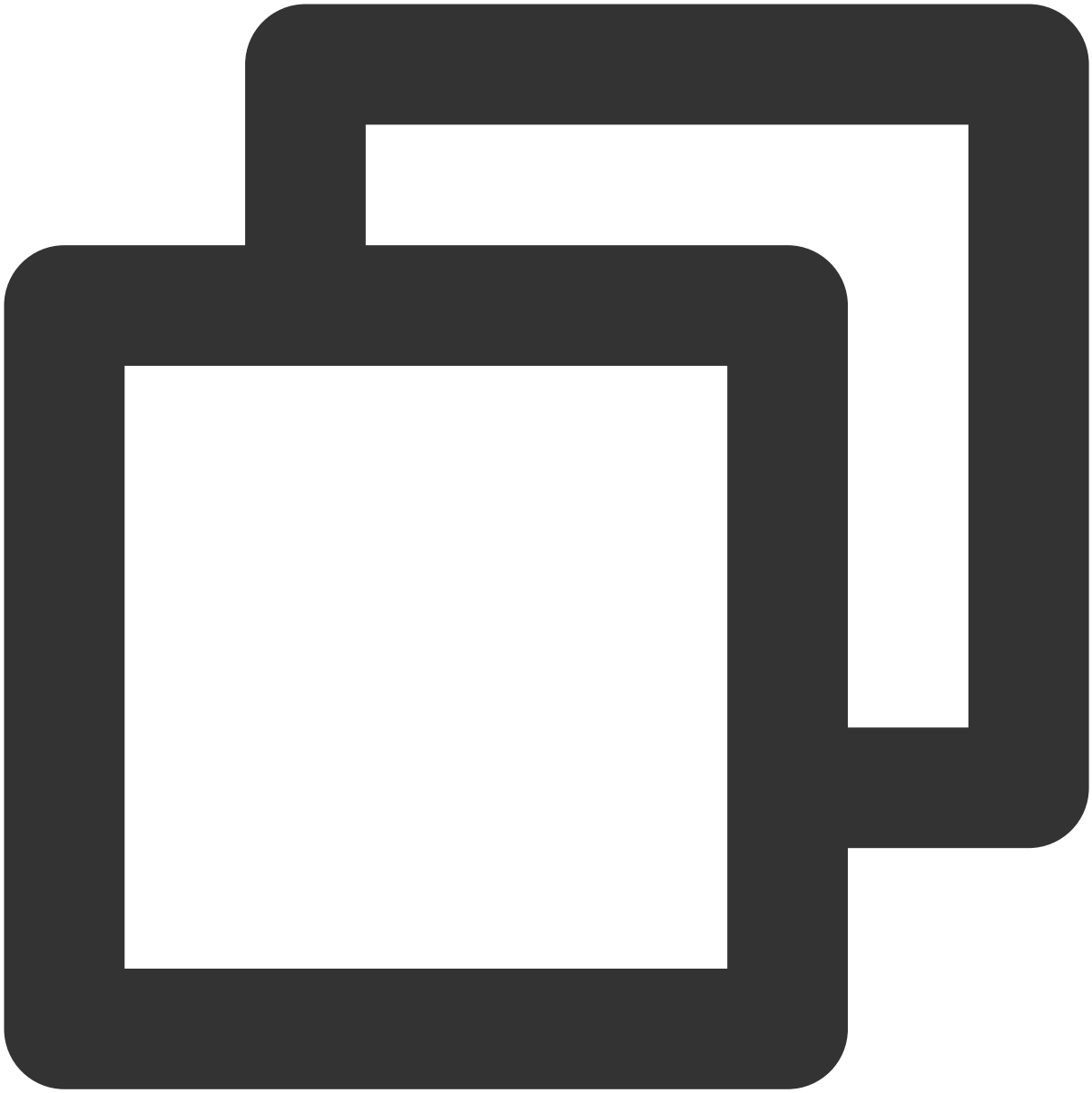
更多完整示例，请前往 GitHub 查看。

查询已上传的分块

功能说明

查询特定分块上传操作中的已上传的块（List Parts）。

示例代码



```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键。

ListPartsRequest listPartsRequest = new ListPartsRequest(bucket, cosPath,
        uploadId);
cosXmlService.listPartsAsync(listPartsRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
        ListParts listParts = ((ListPartsResult) result).listParts;
    }
```

```
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

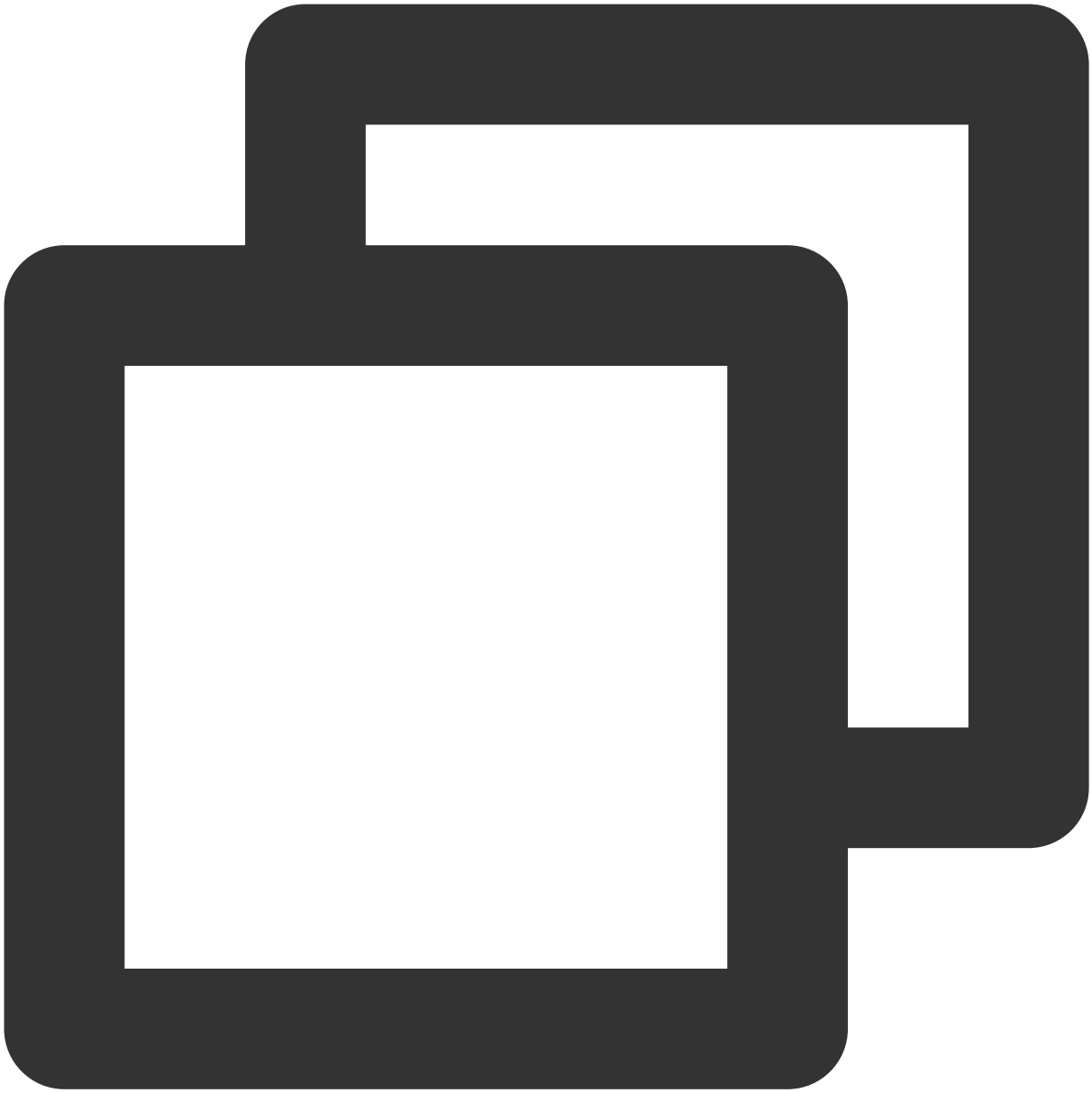
更多完整示例，请前往 GitHub 查看。

完成分块上传

功能说明

完成整个文件的分块上传（Complete Multipart Upload）。

示例代码



```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键。

CompleteMultiUploadRequest completeMultiUploadRequest =
        new CompleteMultiUploadRequest(bucket,
        cosPath, uploadId, eTags);
cosXmlService.completeMultiUploadAsync(completeMultiUploadRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
```

```
            CompleteMultiUploadResult completeMultiUploadResult =
                    (CompleteMultiUploadResult) result;
        }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest cosXmlRequest,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

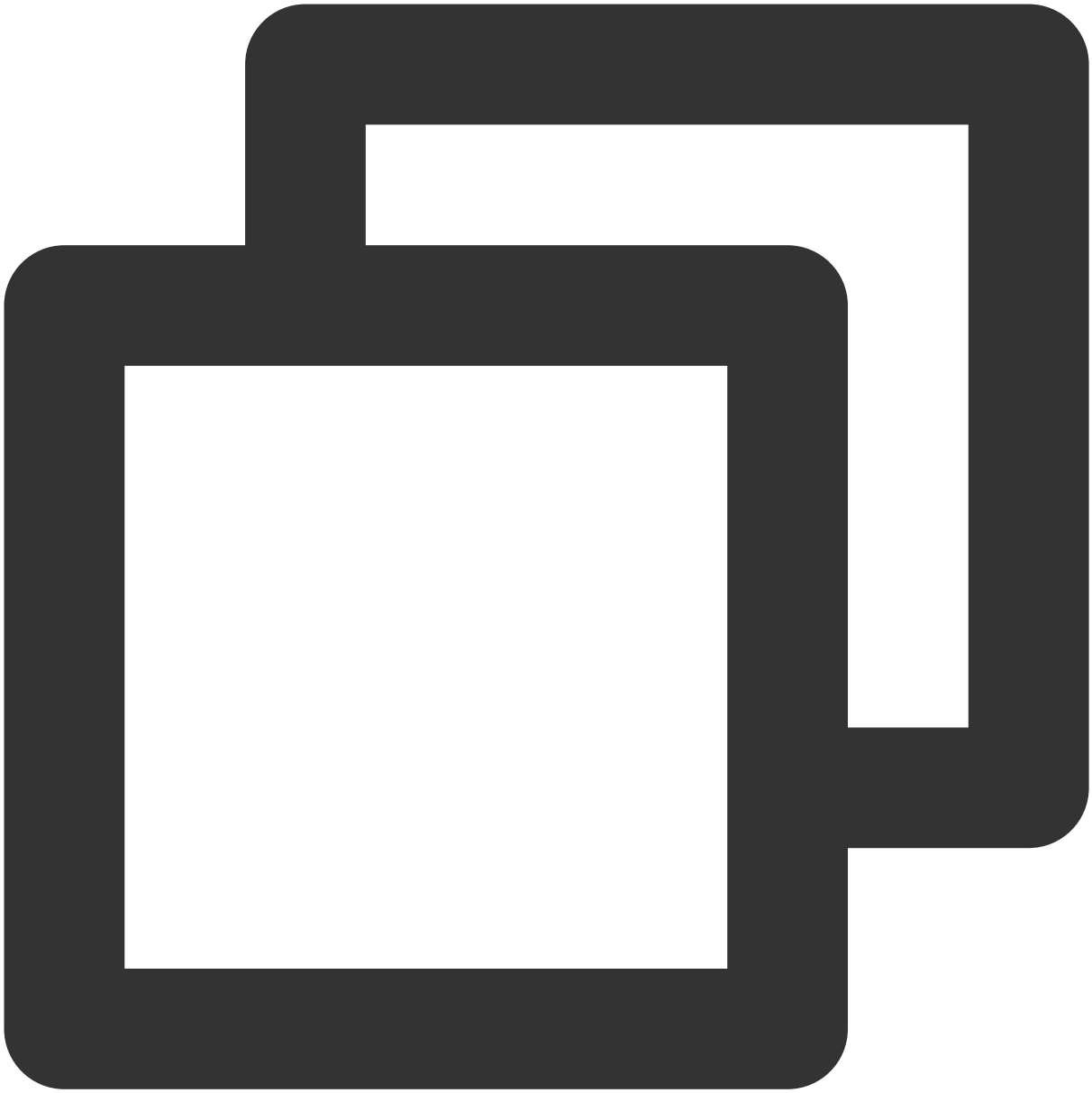
更多完整示例，请前往 GitHub 查看。

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块（Abort Multipart Upload）。

示例代码



```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键。

AbortMultiUploadRequest abortMultiUploadRequest =
        new AbortMultiUploadRequest(bucket,
        cosPath, uploadId);
cosXmlService.abortMultiUploadAsync(abortMultiUploadRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
```

```
            AbortMultiUploadResult abortMultiUploadResult =
                    (AbortMultiUploadResult) result;
        }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest cosXmlRequest,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

下载对象

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地

SDK API 参考

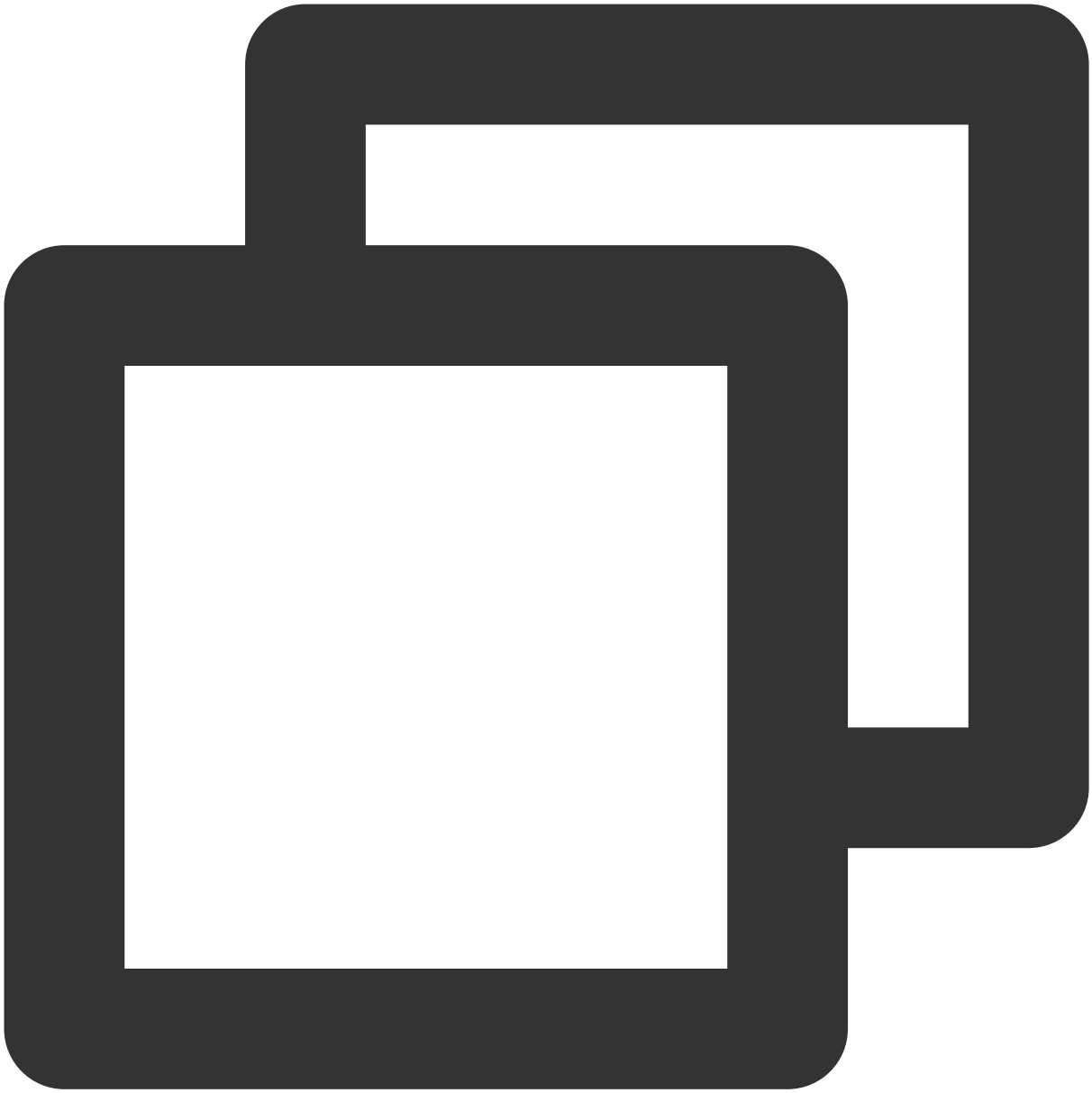
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

高级接口（推荐）

下载对象

高级接口支持暂停、恢复以及取消下载请求，同时支持断点下载功能。

示例代码一: 下载对象



```
// 高级下载接口支持断点续传，所以会在下载前先发起 HEAD 请求获取文件信息。
// 如果您使用的是临时密钥或者使用子账号访问，请确保权限列表中包含 HeadObject 的权限。

// 初始化 TransferConfig，这里使用默认配置，如果需要定制，请参考 SDK 接口文档
TransferConfig transferConfig = new TransferConfig.Builder().build();
//初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);

// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
```

```
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
//本地目录路径
String savePathDir = context.getExternalCacheDir().toString();
//本地保存的文件名，若不填（null），则与 COS 上的文件名一样
String savedFileName = "exampleobject";

Context applicationContext = context.getApplicationContext(); // application
// context
COSXMLDownloadTask cosxmlDownloadTask =
        transferManager.download(applicationContext,
                bucket, cosPath, savePathDir, savedFileName);

//设置下载进度回调
cosxmlDownloadTask.setCosXmlProgressListener(new CosXmlProgressListener() {
    @Override
    public void onProgress(long complete, long target) {
        // todo Do something to update progress...
    }
});
//设置返回结果回调
cosxmlDownloadTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        COSXMLDownloadTask.COSXMLDownloadTaskResult downloadTaskResult =
                (COSXMLDownloadTask.COSXMLDownloadTaskResult) result;
    }

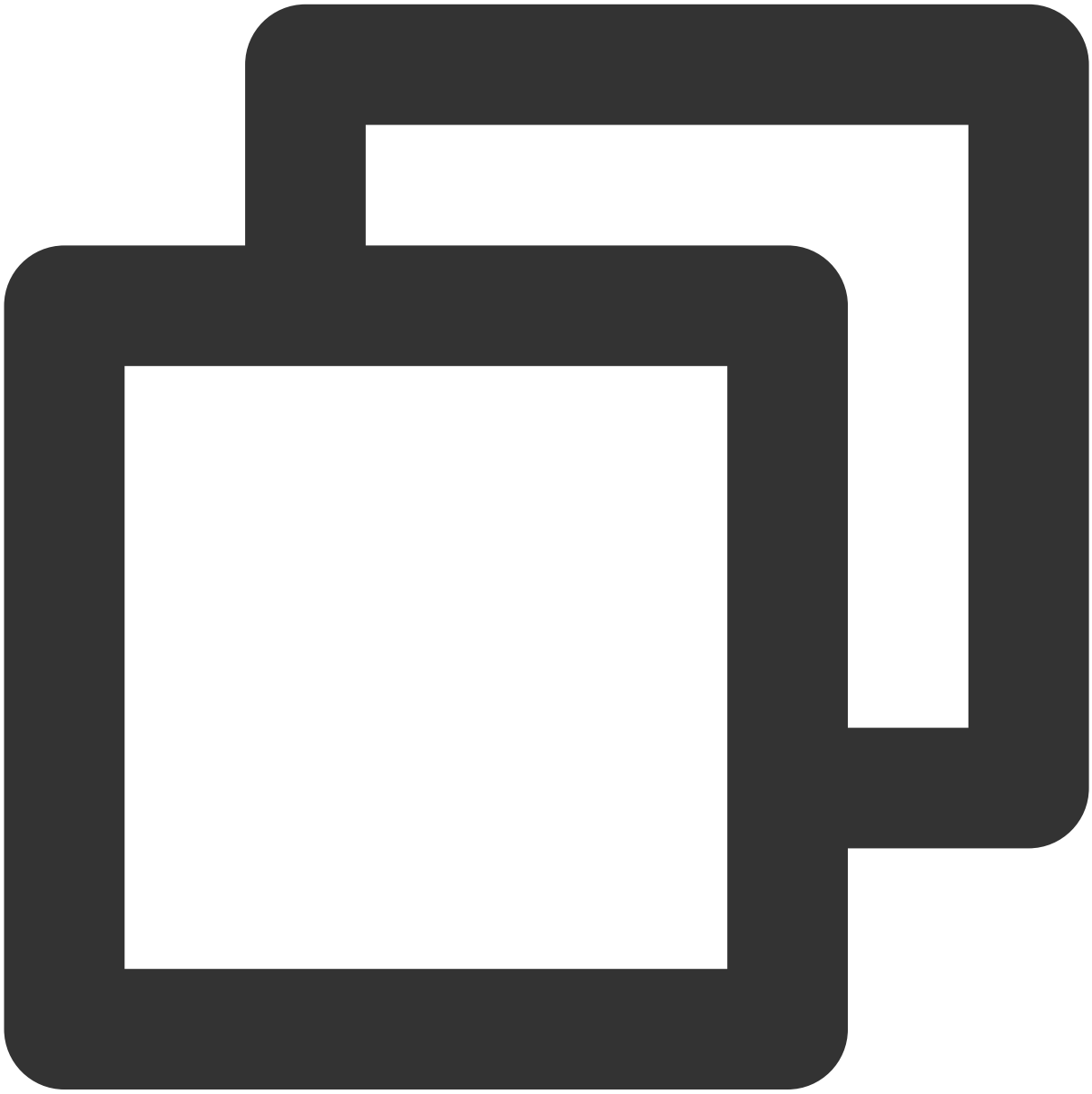
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
//设置任务状态回调，可以查看任务过程
cosxmlDownloadTask.setTransferStateListener(new TransferStateListener() {
    @Override
    public void onStateChanged(TransferState state) {
        // todo notify transfer state
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

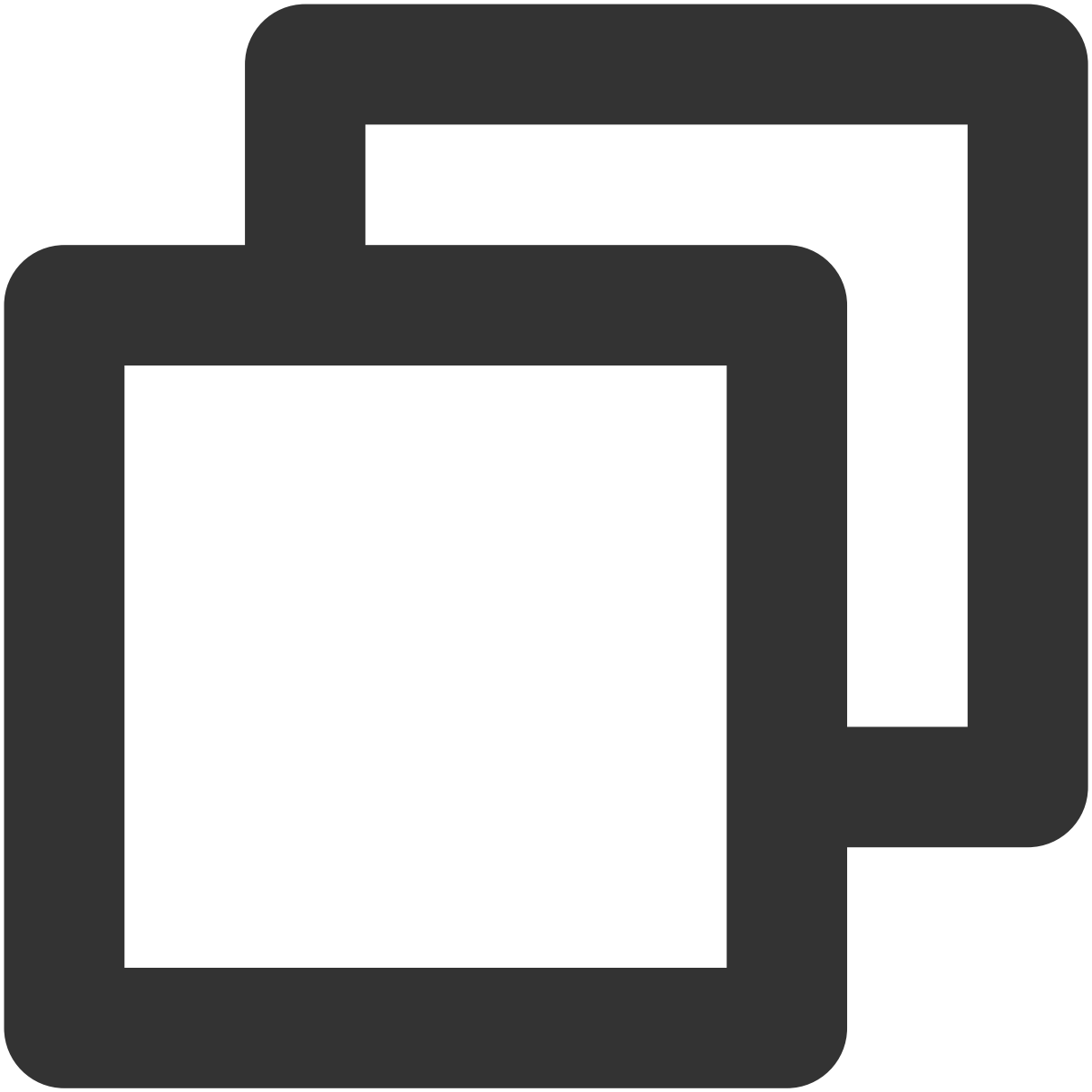
示例代码二: 下载暂停、继续与取消

对于下载任务，可以通过以下方式暂停：



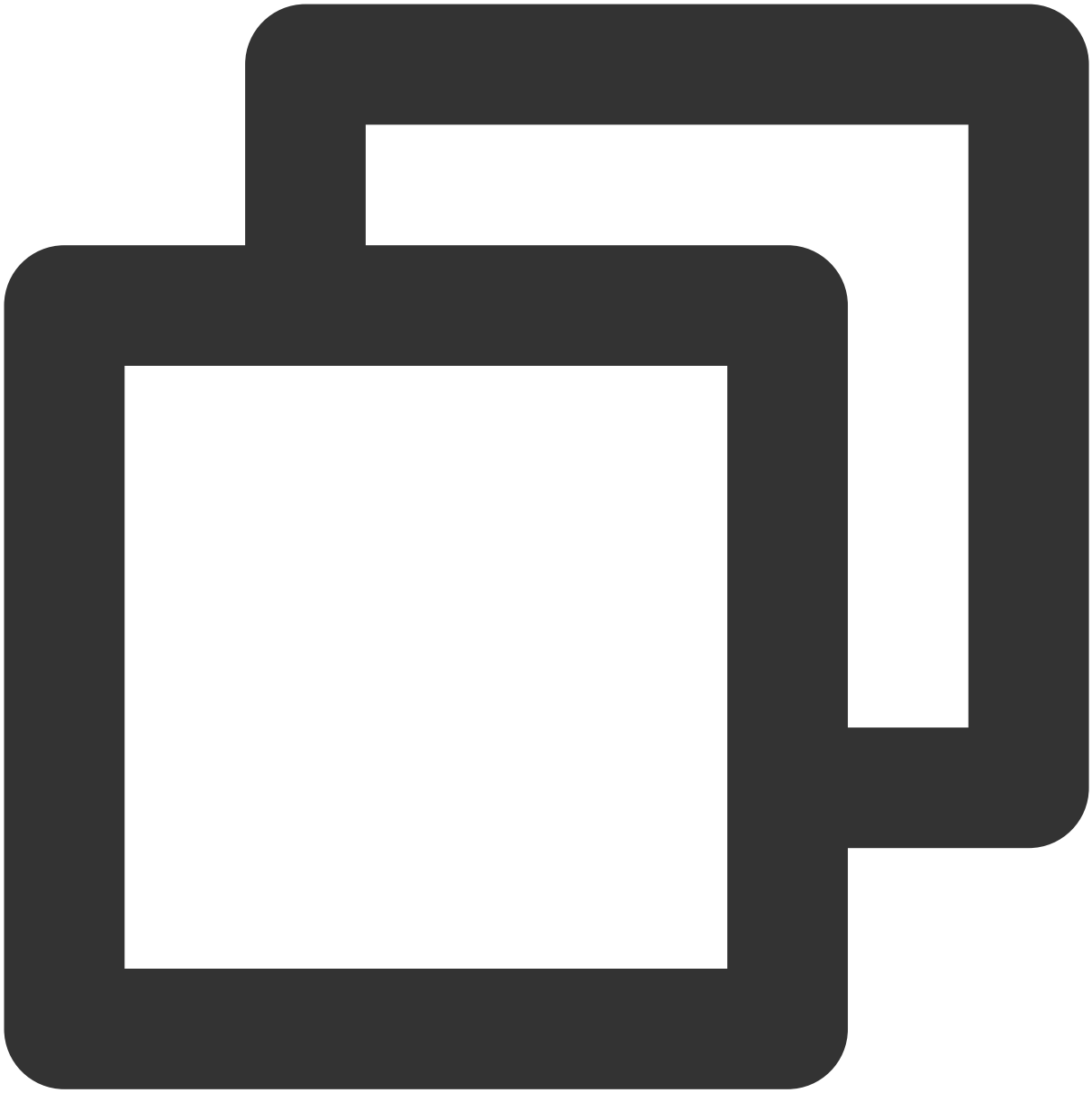
```
cosxmlDownloadTask.pause();
```

暂停之后，可以通过以下方式续传：



```
cosxmlDownloadTask.resume();
```

也通过以下方式取消下载：

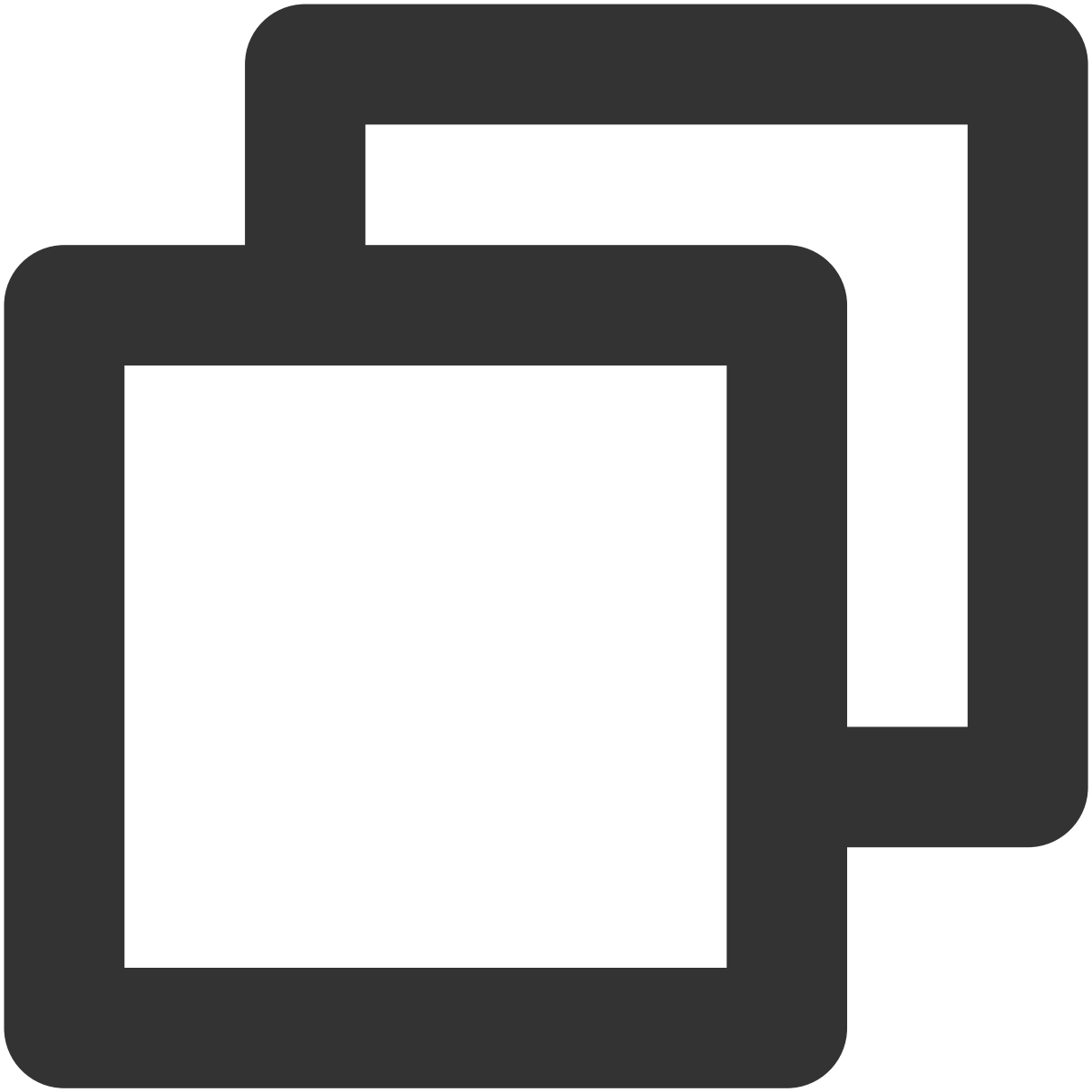


```
cosxmlDownloadTask.cancel();
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码三: 设置下载支持断点续传



```
// 初始化 TransferConfig，这里使用默认配置，如果需要定制，请参考 SDK 接口文档
// TransferManager 支持断点下载，您只需要保证 bucket、cosPath、savePathDir、savedFileName
// 参数一致，SDK 便会从上次已经下载的位置继续下载。
TransferConfig transferConfig = new TransferConfig.Builder().build();
//初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
//本地目录路径
```

```
String savePathDir = context.getExternalCacheDir().toString();
//本地保存的文件名，若不填（null），则与 COS 上的文件名一样
String savedFileName = "exampleobject";

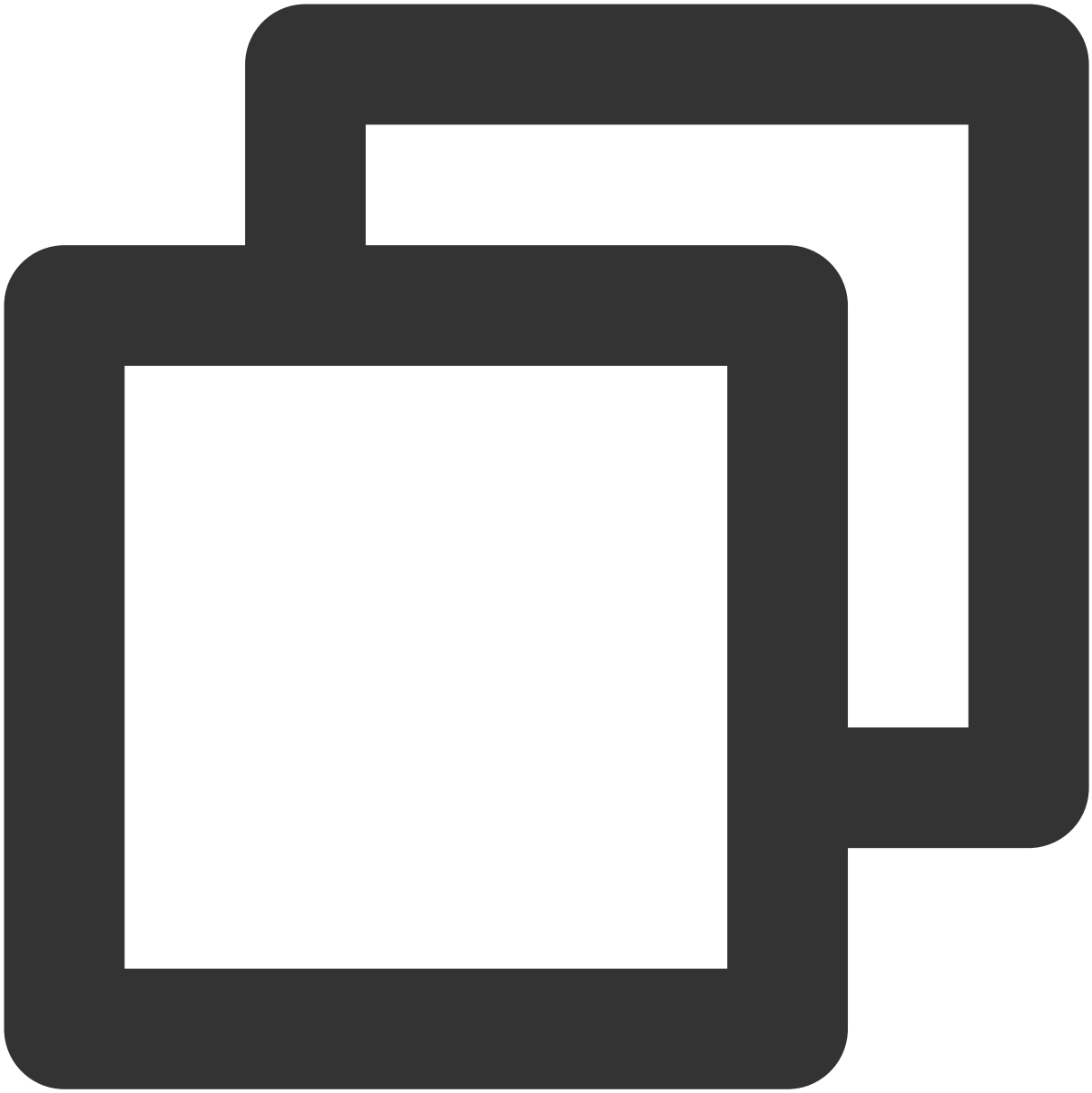
GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, cosPath, savePathD

Context applicationContext = context.getApplicationContext(); // application
// context
COSXMLDownloadTask cosxmlDownloadTask =
        transferManager.download(applicationContext, getObjectRequest);
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码四: 批量下载



```
// 对象在存储桶中的位置标识符，即称对象键
String[] cosPaths = new String[] {
        "exampleobject1",
        "exampleobject2",
        "exampleobject3",
};

for (String cosPath : cosPaths) {

    COSXMLDownloadTask cosxmlDownloadTask =
            transferManager.download(applicationContext,
```

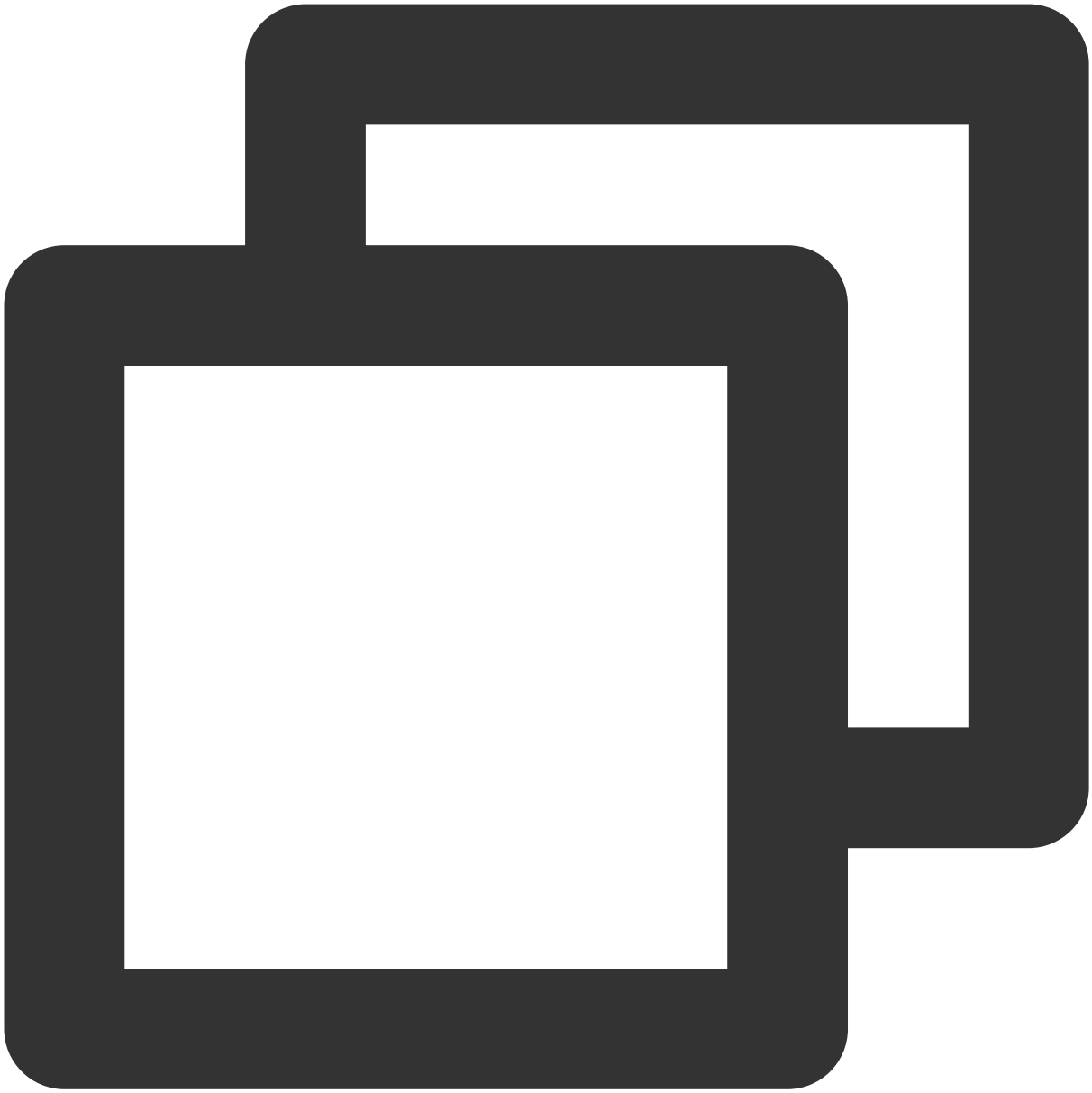
```
                    bucket, cosPath, savePathDir, savedFileName);
        // 设置返回结果回调
        cosxmlDownloadTask.setCosXmlResultListener(new CosXmlResultListener() {
            @Override
            public void onSuccess(CosXmlRequest request, CosXmlResult result) {
                COSXMLDownloadTask.COSXMLDownloadTaskResult downloadResult =
                        (COSXMLDownloadTask.COSXMLDownloadTaskResult) result;
            }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest request,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
                if (clientException != null) {
                    clientException.printStackTrace();
                } else {
                    serviceException.printStackTrace();
                }
            }
        });
    }
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码五: 下载目录



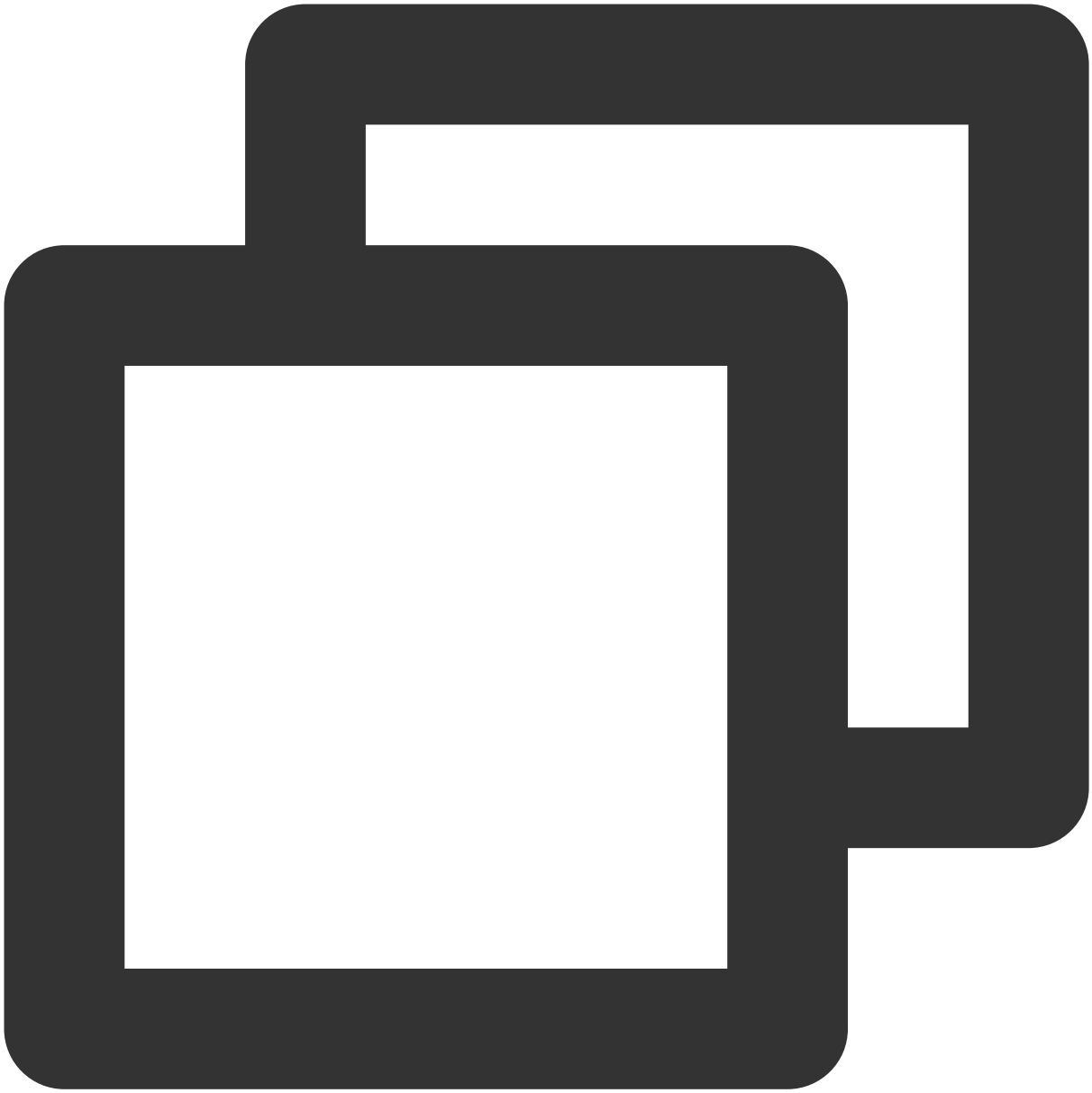
```
boolean isTruncated = true;
String marker = null;
try {
    while (isTruncated) {
        GetBucketRequest getBucketRequest = new GetBucketRequest(region, bucket, di
        // 设置分页信息
        getBucketRequest.setMarker(marker);
        // 设置不查询子目录
        getBucketRequest.setDelimiter("/");
        GetBucketResult getBucketResult = cosXmlService.getBucket(getBucketRequest)
        // 批量下载
```

```
            for (ListBucket.Contents content : getBucketResult.listBucket.contentsList)
                GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, conten
                transferManager.download(context,getObjectRequest);
            }
            isTruncated = getBucketResult.listBucket.isTruncated;
            marker = getBucketResult.listBucket.nextMarker;
        }
    } catch (CosXmlServiceException serviceException) {
        serviceException.printStackTrace();
    } catch (CosXmlClientException clientException) {
        clientException.printStackTrace();
    }
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码六: 匿名下载（不用传入密钥，对公开文件进行下载）



```
// 初始化 TransferConfig，这里使用默认配置，如果需要定制，请参考 SDK 接口文档
TransferConfig transferConfig = new TransferConfig.Builder().build();
// 初始化 TransferManager
CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig.Builder()
        .setRegion("ap-guangzhou")
        .builder();
// 匿名下载生成 CosXmlService 不需要传入密钥生成器
CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig);
TransferManager transferManager = new TransferManager(cosXmlService, transferConfig
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
```

```
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
// 本地目录路径
String savePathDir = context.getExternalCacheDir().toString();
// 本地保存的文件名，若不填（null），则与 COS 上的文件名一样
String savedFileName = "exampleobject";

GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, cosPath, savePathD

Context applicationContext = context.getApplicationContext(); // application
// context
COSXMLDownloadTask cosxmlDownloadTask =
        transferManager.download(applicationContext, getObjectRequest);
```

说明

更多完整示例，请前往 GitHub 查看。

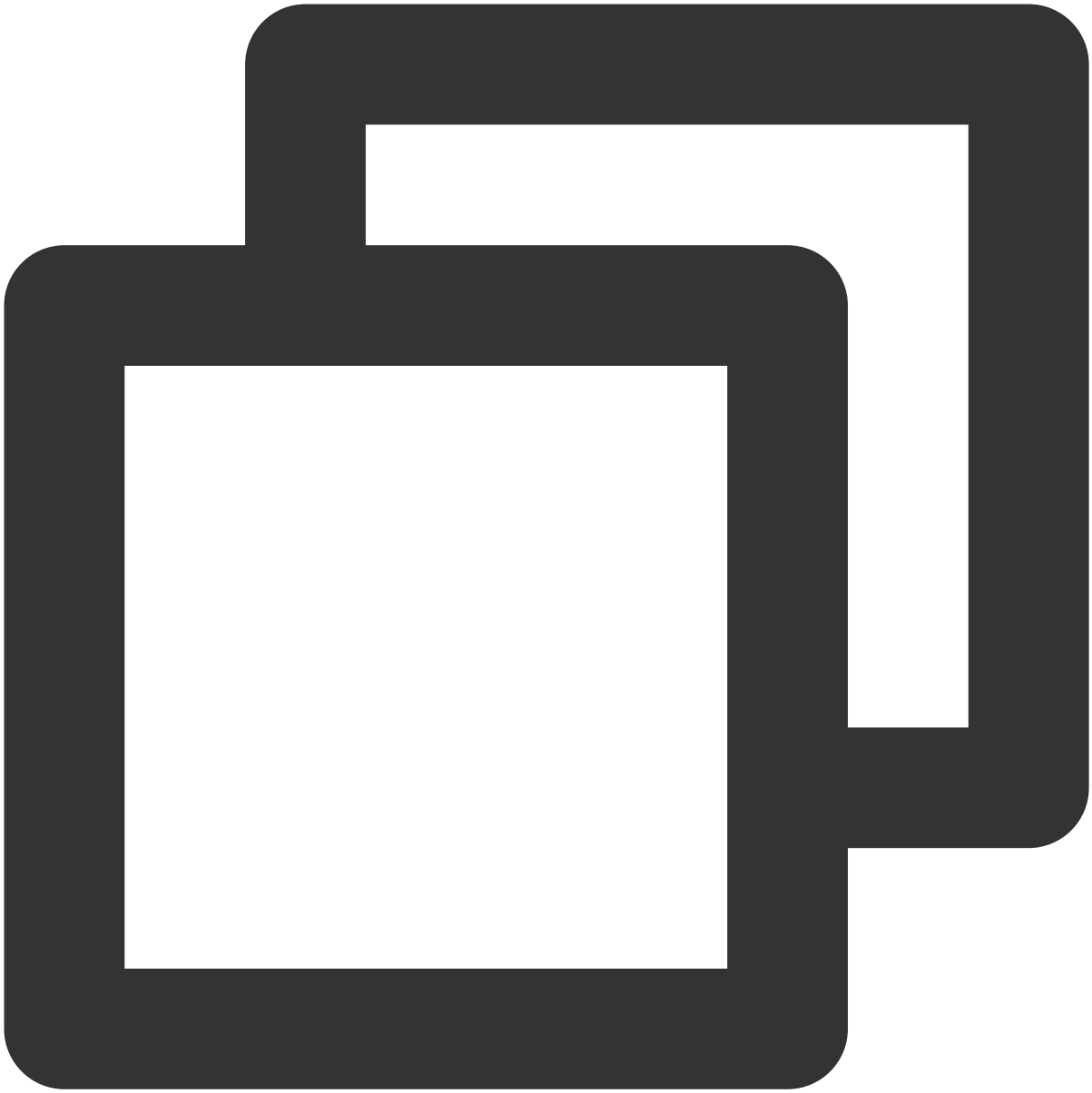
简单操作

下载对象

功能说明

下载一个 Object（文件/对象）至本地（GET Object）。

示例代码



```
String bucket = "examplebucket-1250000000"; //存储桶名称，格式：BucketName-APPID
String cosPath = "exampleobject"; //对象位于存储桶中的位置标识符，即对象键
String savePath = context.getExternalCacheDir().toString(); //本地路径

GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, cosPath,
        savePath);
getObjectRequest.setProgressListener(new CosXmlProgressListener() {
    @Override
    public void onProgress(long progress, long max) {
        // todo Do something to update progress...
    }
```

```
});

cosXmlService.getObjectAsync(getObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest,
                          CosXmlResult cosXmlResult) {
        GetObjectResult getObjectResult = (GetObjectResult) cosXmlResult;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

复制与移动对象

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于复制、移动对象的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object - Copy	设置对象复制（修改对象属性）	复制文件到目标路径
DELETE Object	删除单个对象	在存储桶中删除指定对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传/复制	查询正在进行中的分块上传/复制信息
Initiate Multipart Upload	初始化分块上传/复制	初始化分块上传/复制操作
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传/复制块	查询特定分块操作中的已上传/复制的块
Complete Multipart Upload	完成分块上传/复制	完成整个文件的分块上传/复制
Abort Multipart Upload	终止分块上传/复制	终止一个分块操作并删除已上传/复制的块

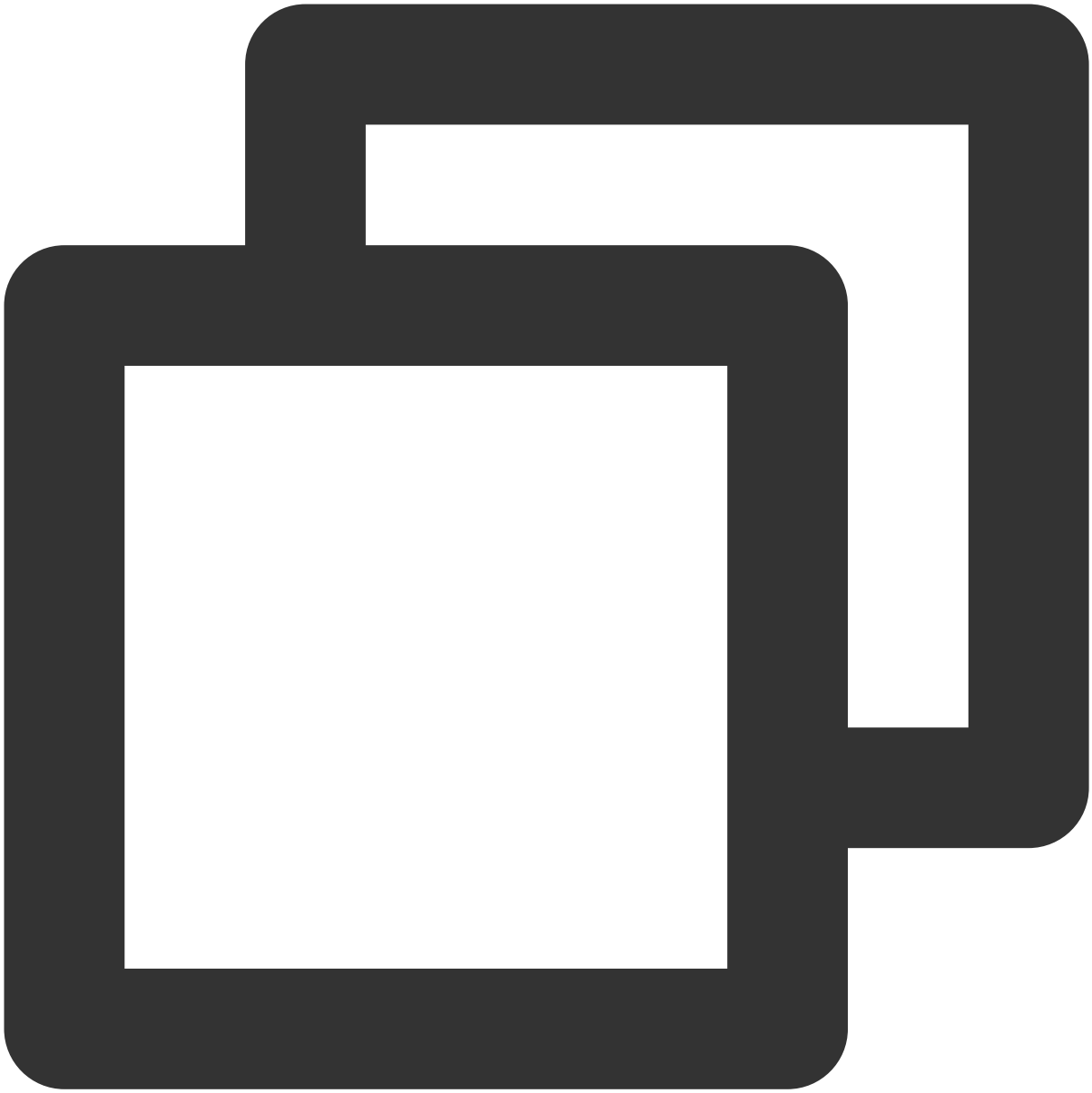
SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

复制对象

高级接口封装了简单复制、分块复制接口的异步请求，并支持暂停、恢复以及取消复制请求。

示例代码



```
// 初始化 TransferConfig，这里使用默认配置，如果需要定制，请参考 SDK 接口文档
TransferConfig transferConfig = new TransferConfig.Builder().build();
//初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);

String sourceAppid = "1250000000"; //账号 APPID
String sourceBucket = "sourcebucket-1250000000"; //源对象所在的存储桶
String sourceRegion = "COS_REGION"; //源对象的存储桶所在的地域
String sourceCosPath = "sourceObject"; //源对象的对象键
//构造源对象属性
```

```
CopyObjectRequest.CopySourceStruct copySourceStruct =
        new CopyObjectRequest.CopySourceStruct(
                sourceAppid, sourceBucket, sourceRegion, sourceCosPath);
//目标桶
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
//目标对象
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键

//复制对象
COSXMLCopyTask cosxmlCopyTask = transferManager.copy(bucket, cosPath,
        copySourceStruct);

//设置返回结果回调
cosxmlCopyTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        COSXMLCopyTask.COSXMLCopyTaskResult copyResult =
                (COSXMLCopyTask.COSXMLCopyTaskResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
//设置任务状态回调，可以查看任务过程
cosxmlCopyTask.setTransferStateListener(new TransferStateListener() {
    @Override
    public void onStateChanged(TransferState state) {
        // todo notify transfer state
    }
});
```

移动对象

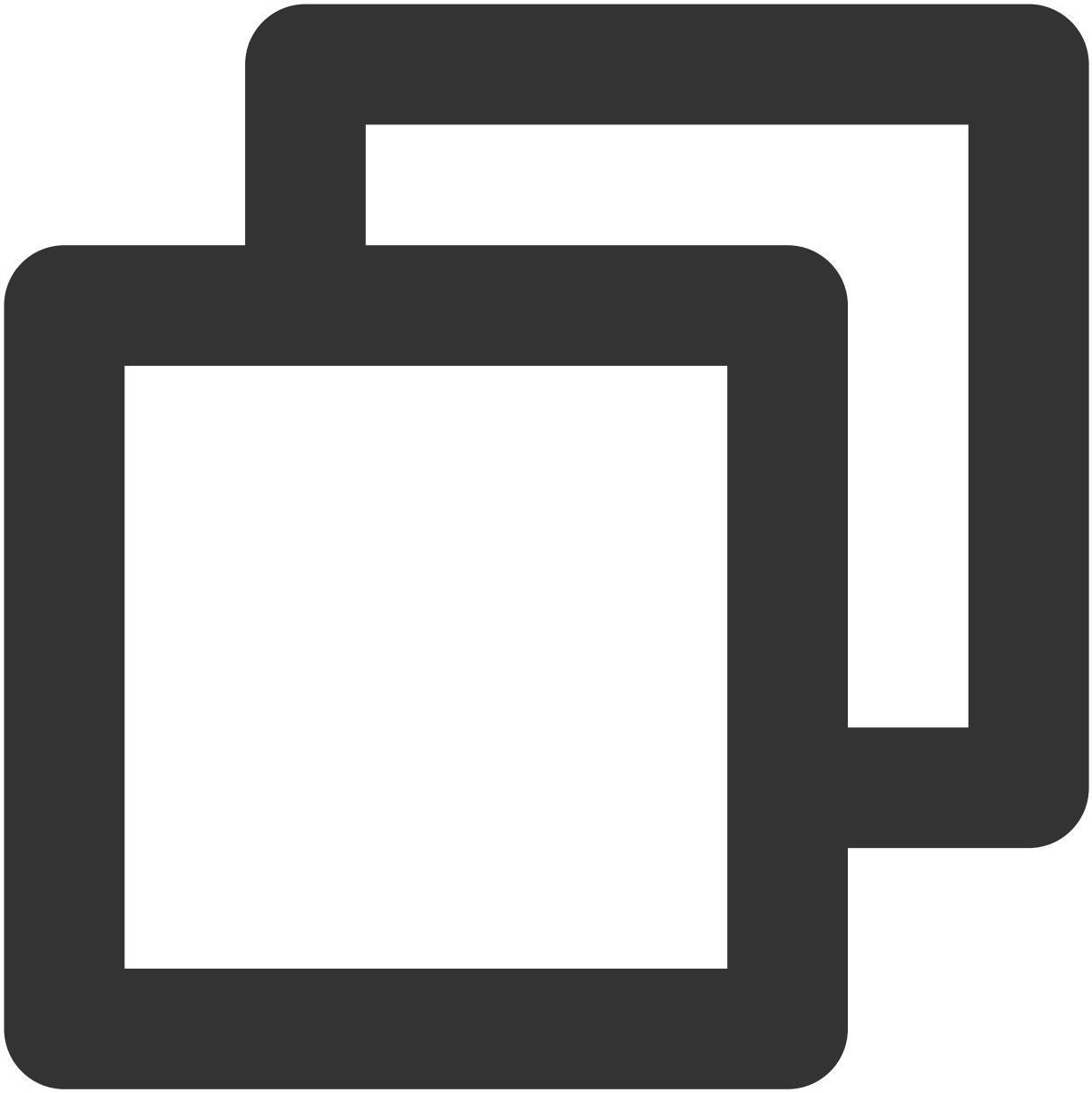
移动对象主要包括两个操作：复制源对象到目标位置，删除源对象。

由于 COS 通过存储桶名称（Bucket）和对象键（ObjectKey）来标识对象。移动对象也就意味着修改这个对象的标识，COS Android SDK 目前没有提供修改对象唯一标识名的单独接口，但是可以通过组合**复制对象**加上**删除对象**的基本操作，来达到修改对象标识的目的，从而实现移动对象。

例如将 mybucket-1250000000 这个存储桶中的 picture.jpg 这个对象移动到同个存储桶的 doc 路径下。首先可以复制 picture.jpg 对象到存储桶的 doc 路径下，即对象键设定为 doc/picture.jpg，复制完成后删除 picture.jpg 这个对象，来实现“移动”的效果。

同样的，如果想将 mybucket-1250000000 这个存储桶里的 picture.jpg 这个对象移动到 myanothorbucket-1250000000 这个存储桶里，可以先将对象复制到 myanothorbucket-1250000000 存储桶，然后删除原来的对象。

示例代码



```
final String sourceAppid = "1250000000"; //账号 appid
final String sourceBucket = "sourcebucket-1250000000"; //"源对象所在的存储桶
final String sourceRegion = "COS_REGION"; //源对象的存储桶所在的地域
final String sourceKey = "sourceObject"; //源对象键
//构造源对象属性
CopyObjectRequest.CopySourceStruct copySource = new CopyObjectRequest.CopySourceStr
        sourceRegion, sourceKey);

String bucket = "examplebucket-1250000000"; //目标存储桶，格式：BucketName-APPID
String key = "exampleobject"; //目标对象的对象键
```

```
// copy(String bucket, String cosPath, CopyObjectRequest.CopySourceStruct copySourc
COSXMLCopyTask copyTask = transferManager.copy(bucket, key, copySource);
copyTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        try {
            // 复制成功后删除文件
            DeleteObjectRequest deleteObjectRequest = new DeleteObjectRequest(sourc
            DeleteObjectResult deleteResult = cosXmlService.deleteObject(deleteObje
        } catch (CosXmlClientException e) {
            e.printStackTrace();
        } catch (CosXmlServiceException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onFail(CosXmlRequest request, CosXmlClientException exception, CosX

    }
});
```

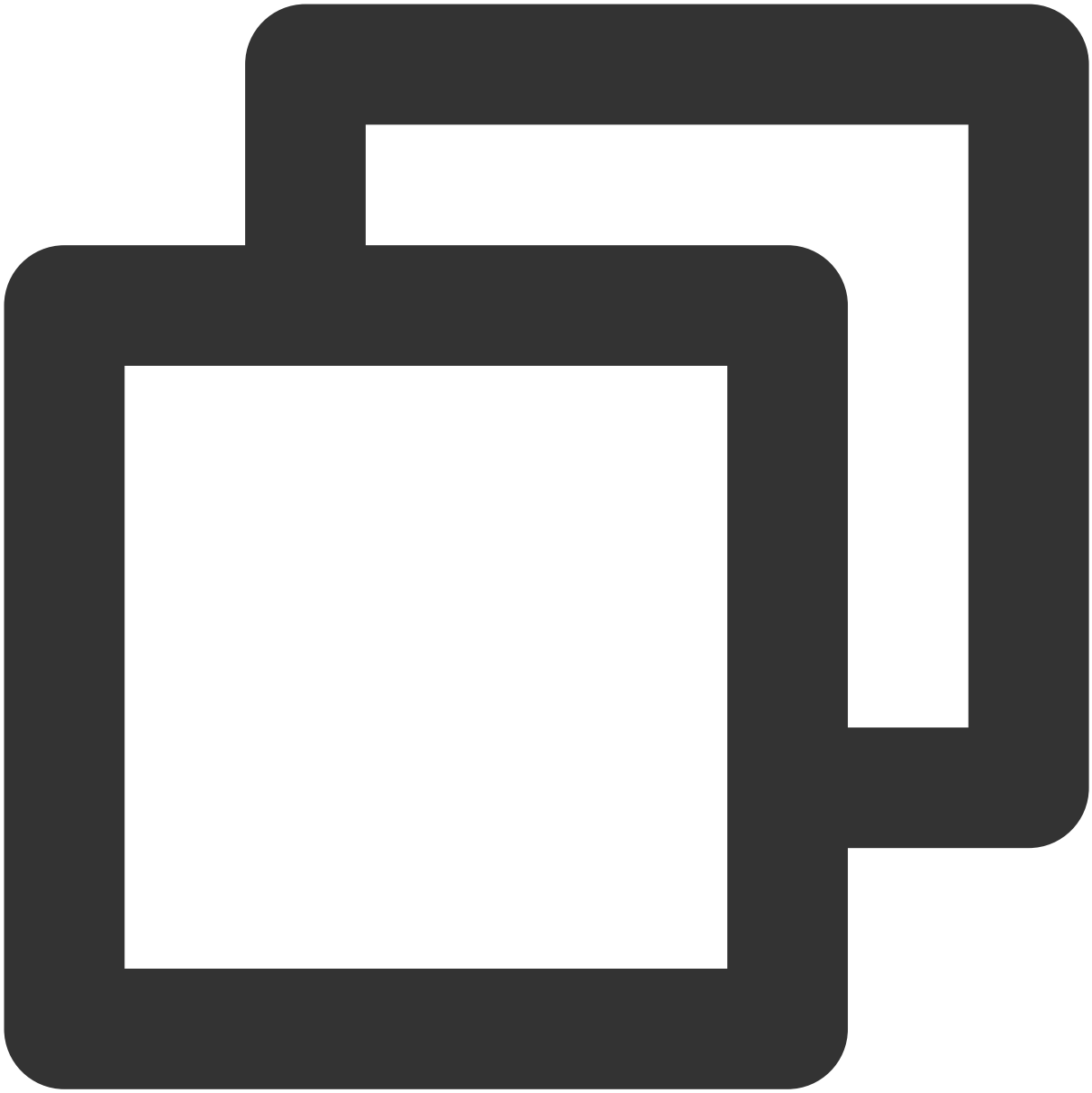
说明

更多完整示例，请前往 GitHub 查看。

复制对象（修改属性）

复制文件到目标路径（PUT Object-Copy）。

示例代码一: 复制对象时保留对象属性



```
String sourceAppid = "1250000000"; //账号 APPID
String sourceBucket = "sourcebucket-1250000000"; //源对象所在的存储桶
String sourceRegion = "COS_REGION"; //源对象的存储桶所在的地域
String sourceCosPath = "sourceObject"; //源对象键
// 构造源对象属性
CopyObjectRequest.CopySourceStruct copySourceStruct =
        new CopyObjectRequest.CopySourceStruct(
        sourceAppid, sourceBucket, sourceRegion, sourceCosPath);

// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
```

```
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(bucket, cosPath,
        copySourceStruct);

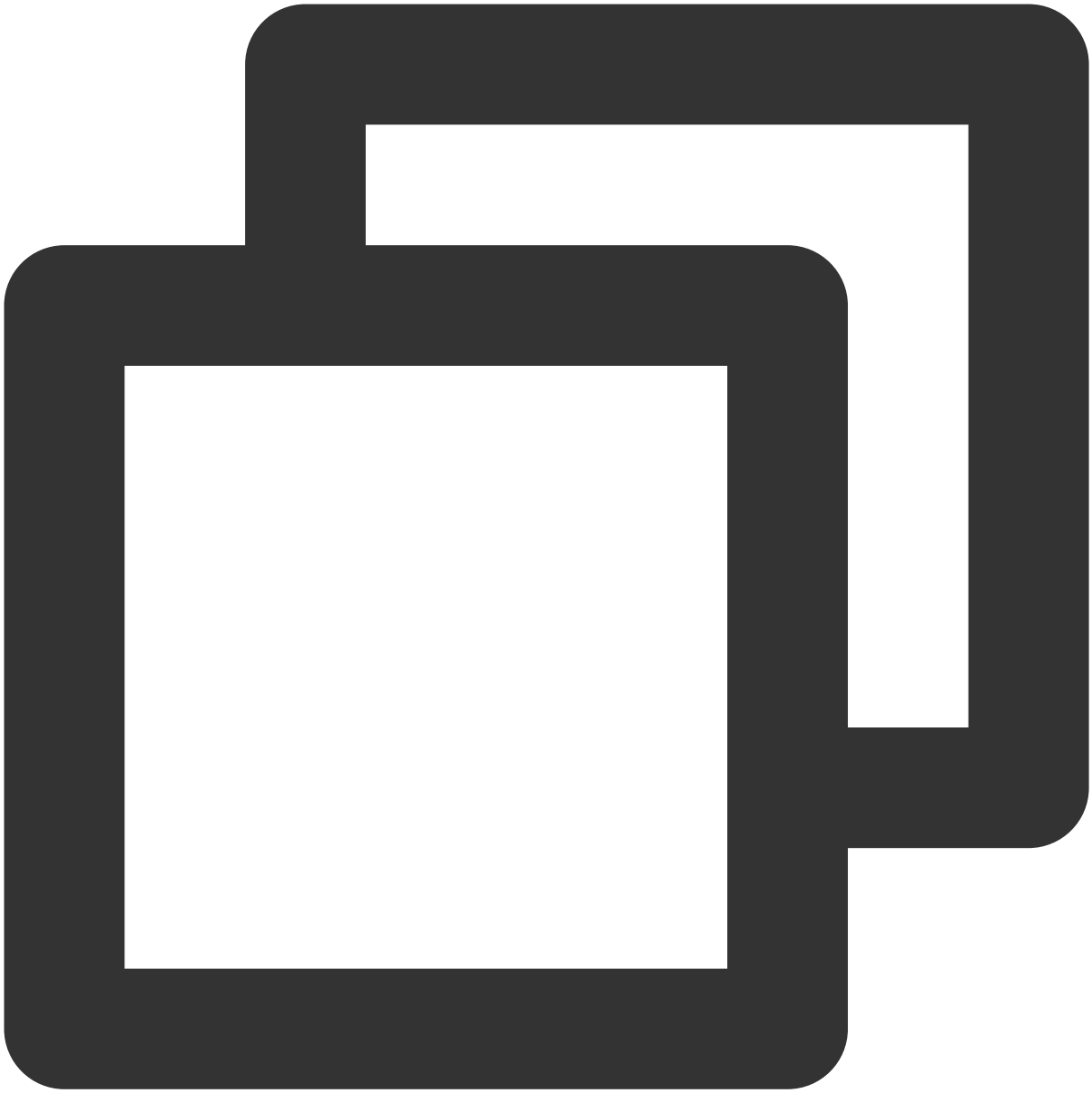
cosXmlService.copyObjectAsync(copyObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        CopyObjectResult copyObjectResult = (CopyObjectResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码二: 复制对象时替换对象属性



```
String sourceAppid = "1250000000"; //账号 APPID
String sourceBucket = "sourcebucket-1250000000"; //源对象所在的存储桶
String sourceRegion = "COS_REGION"; //源对象的存储桶所在的地域
String sourceCosPath = "sourceObject"; //源对象键
// 构造源对象属性
CopyObjectRequest.CopySourceStruct copySourceStruct =
        new CopyObjectRequest.CopySourceStruct(
        sourceAppid, sourceBucket, sourceRegion, sourceCosPath);

// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
```

```
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(bucket, cosPath,
        copySourceStruct);
copyObjectRequest.setCopyMetaDataDirective(MetaDataDirective.REPLACED);
copyObjectRequest.setXCOSMeta("x-cos-metadata-oldKey", "newValue");

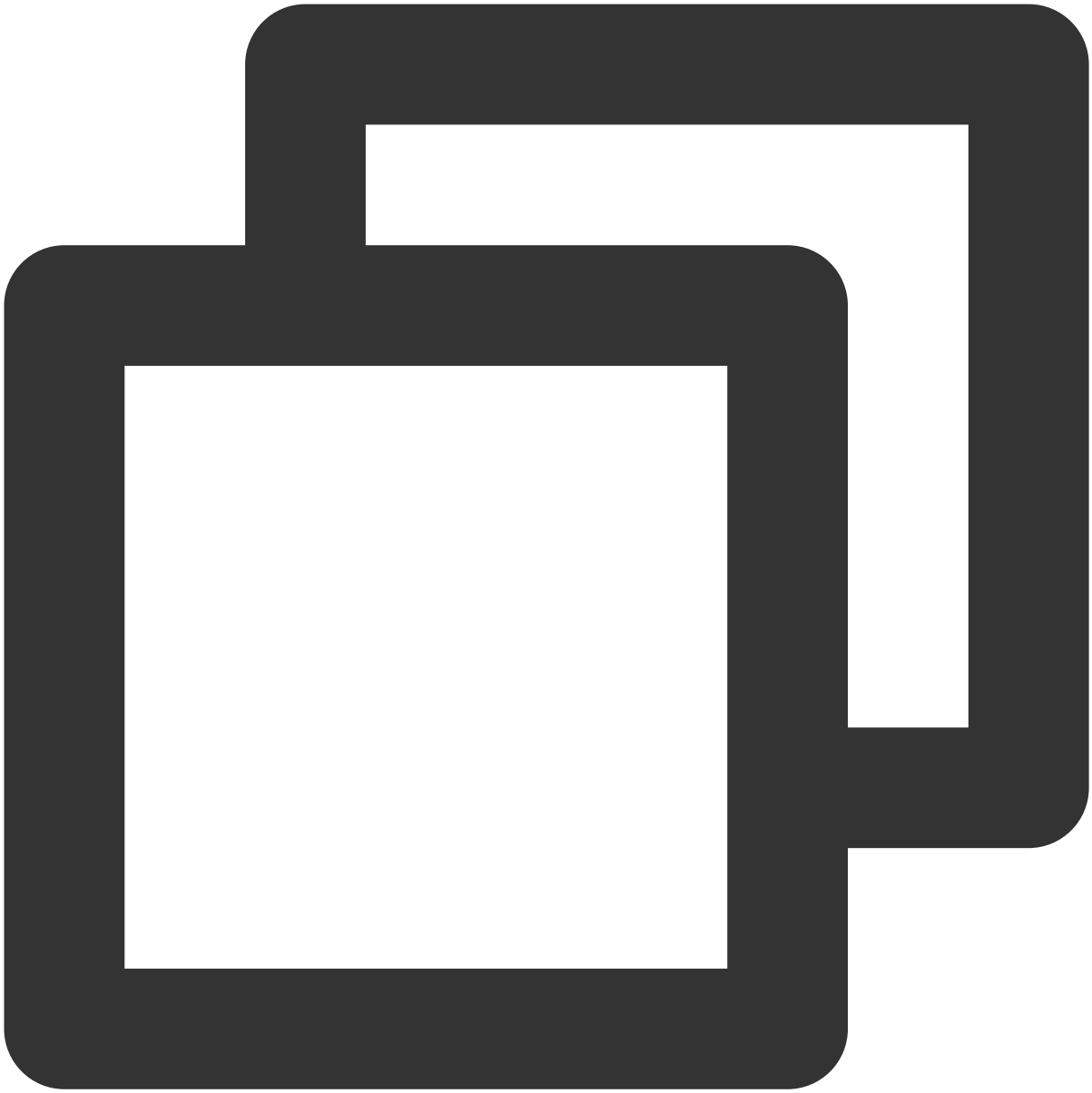
cosXmlService.copyObjectAsync(copyObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        CopyObjectResult copyObjectResult = (CopyObjectResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码三: 修改对象元数据



```
String appId = "1250000000"; //账号 APPID
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String region = "COS_REGION"; //源对象的存储桶所在的地域
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键
// 构造源对象属性
CopyObjectRequest.CopySourceStruct copySourceStruct =
        new CopyObjectRequest.CopySourceStruct(
        appId, bucket, region, cosPath);

CopyObjectRequest copyObjectRequest = new CopyObjectRequest(bucket, cosPath,
```

```
        copySourceStruct);
copyObjectRequest.setCopyMetaDataDirective(MetaDataDirective.REPLACED);
// 修改元数据为新值
copyObjectRequest.setXCOSMeta("x-cos-metadata-oldKey", "newValue");

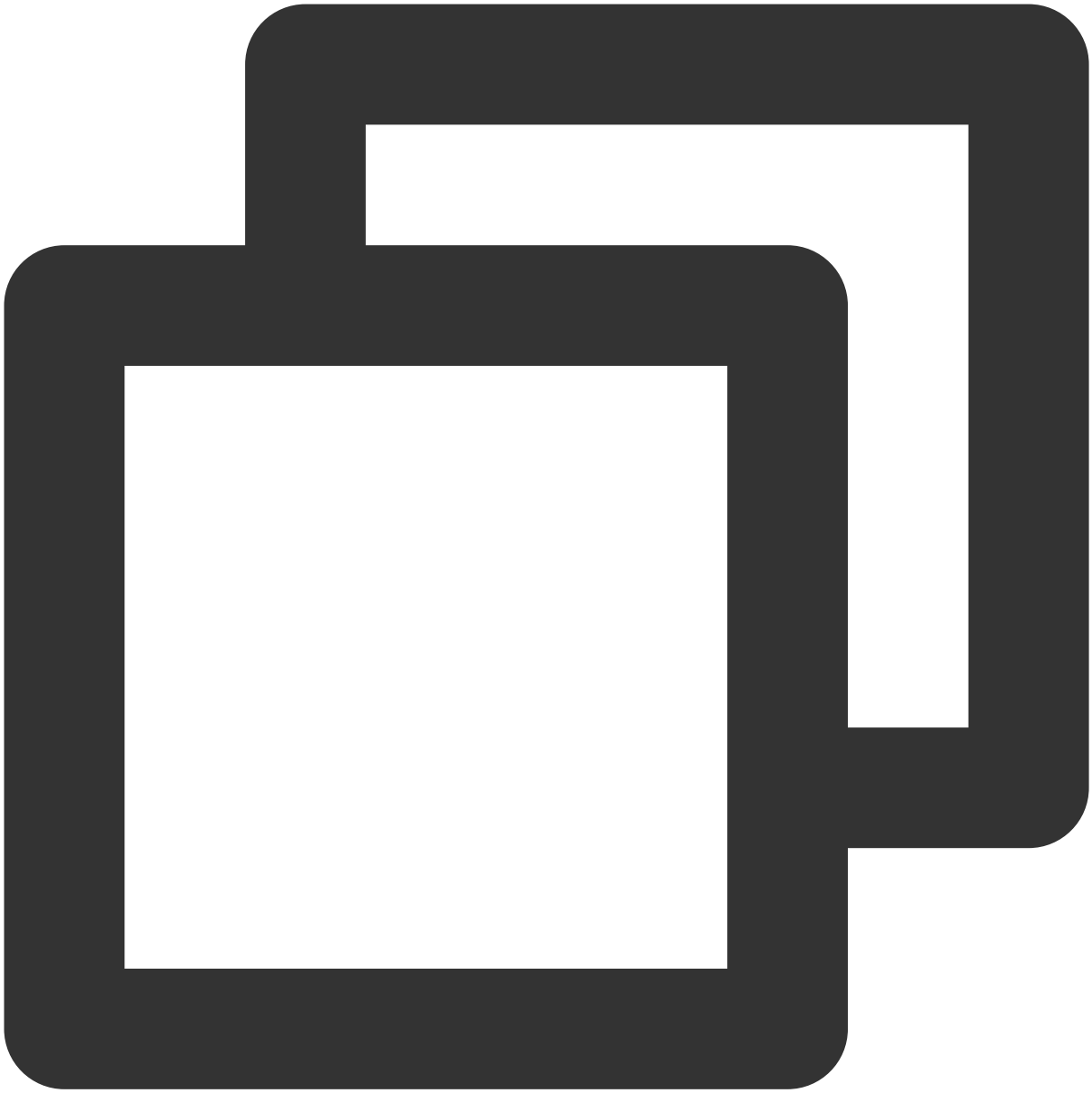
cosXmlService.copyObjectAsync(copyObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        CopyObjectResult copyObjectResult = (CopyObjectResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码四: 修改对象存储类型



```
String appId = "1250000000"; //账号 APPID
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String region = "COS_REGION"; //源对象的存储桶所在的地域
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键
// 构造源对象属性
CopyObjectRequest.CopySourceStruct copySourceStruct =
        new CopyObjectRequest.CopySourceStruct(
        appId, bucket, region, cosPath);

CopyObjectRequest copyObjectRequest = new CopyObjectRequest(bucket, cosPath,
```

```
        copySourceStruct);
// 修改为低频存储
copyObjectRequest.setCosStorageClass(COSStorageClass.STANDARD_IA);

cosXmlService.copyObjectAsync(copyObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        CopyObjectResult copyObjectResult = (CopyObjectResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

列出对象

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于列出对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象
GET Bucket Object Versions	查询对象及其历史版本列表	查询存储桶下的部分或者全部对象及其历史版本信息

SDK API 参考

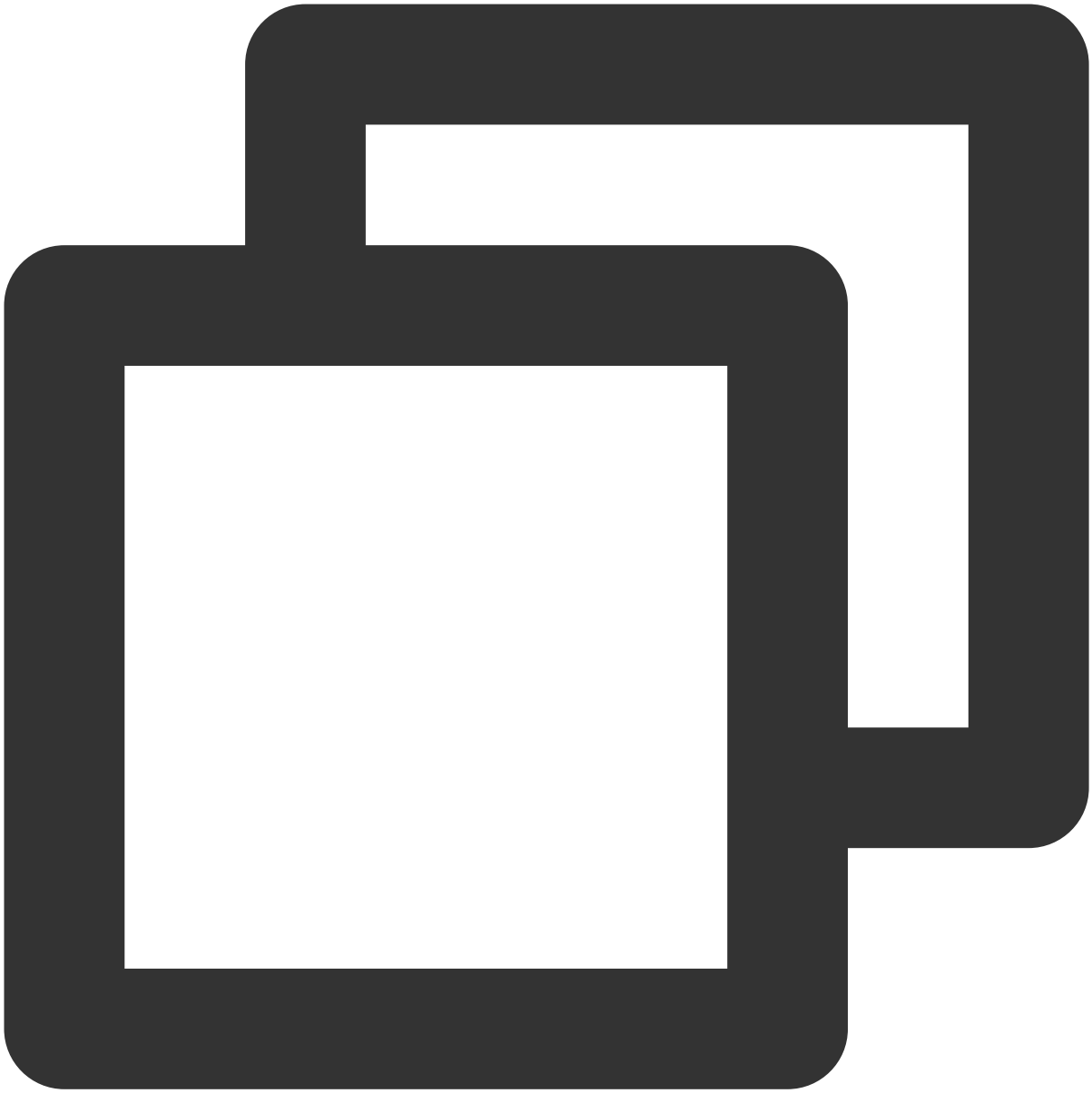
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

示例代码一: 获取第一页数据



```
String bucketName = "examplebucket-1250000000"; //格式：BucketName-APPID;
final GetBucketRequest getBucketRequest = new GetBucketRequest(bucketName);

// 前缀匹配，用来规定返回的对象前缀地址
getBucketRequest.setPrefix("dir/");

// 单次返回最大的条目数量，默认1000
getBucketRequest.setMaxKeys(100);

cosXmlService.getBucketAsync(getBucketRequest, new CosXmlResultListener() {
    @Override
```

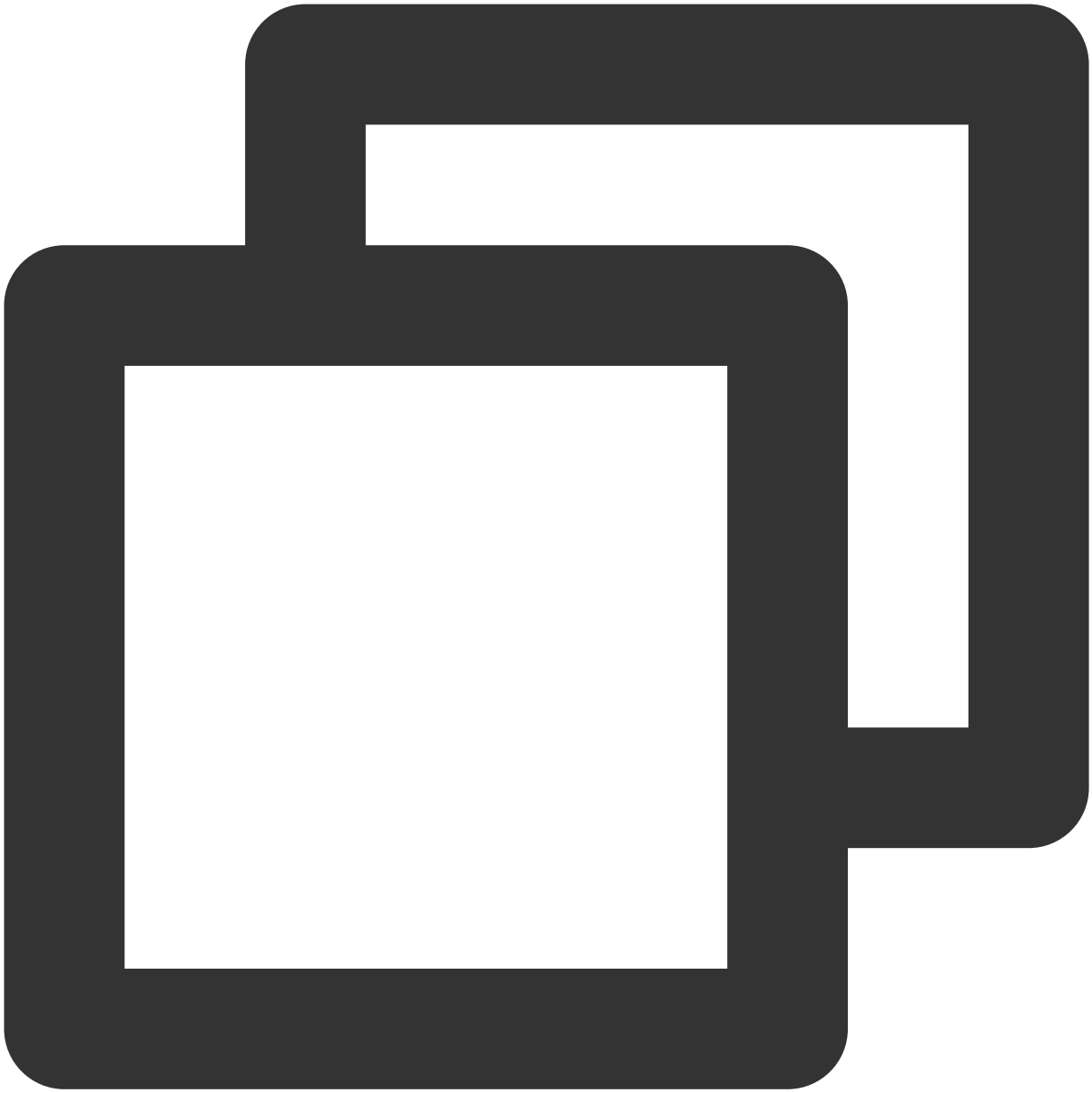
```
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketResult getBucketResult = (GetBucketResult) result;
        if (getBucketResult.listBucket.isTruncated) {
            // 表示数据被截断，需要拉取下一页数据
            prevPageResult = getBucketResult;
        }
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码二：请求下一页数据



```
String bucketName = "examplebucket-1250000000"; //格式：BucketName-APPID;


GetBucketRequest getBucketRequest = new GetBucketRequest(bucketName);

// 前缀匹配，用来规定返回的对象前缀地址
getBucketRequest.setPrefix("dir/");

// prevPageResult 是上一页的返回结果，这里的 nextMarker 表示下一页的起始位置
String nextMarker = prevPageResult.listBucket.nextMarker;
getBucketRequest.setMarker(nextMarker);
```

```
// 单次返回最大的条目数量，默认1000
getBucketRequest.setMaxKeys(100);

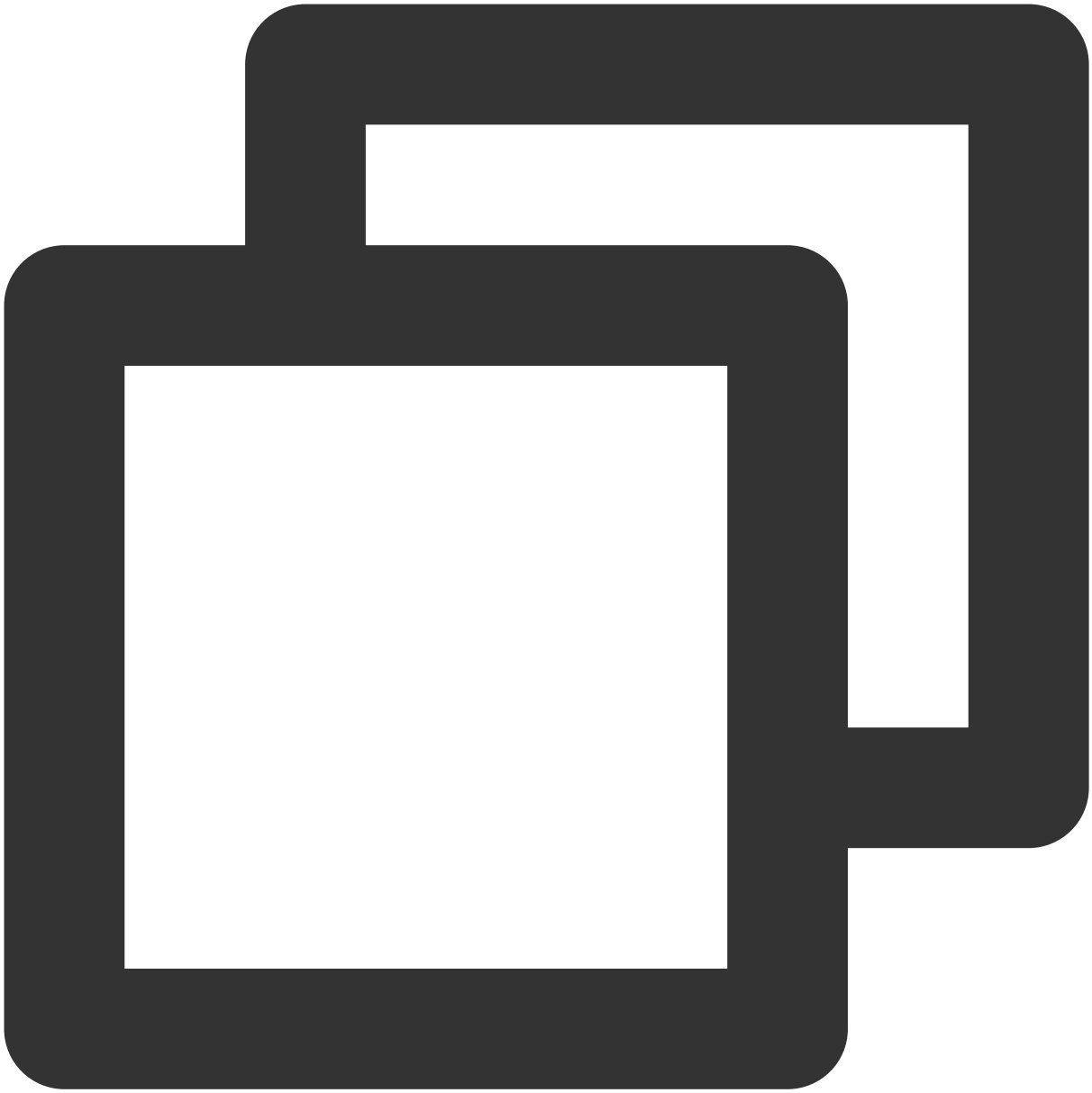
cosXmlService.getBucketAsync(getBucketRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketResult getBucketResult = (GetBucketResult) result;
        if (getBucketResult.listBucket.isTruncated) {
            // 表示数据被截断，需要拉取下一页数据
        }
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码三：获取对象列表与子目录



```
String bucketName = "examplebucket-1250000000"; //格式：BucketName-APPID;
GetBucketRequest getBucketRequest = new GetBucketRequest(bucketName);

// 前缀匹配，用来规定返回的对象前缀地址
getBucketRequest.setPrefix("dir/");

// 单次返回最大的条目数量，默认1000
getBucketRequest.setMaxKeys(100);

// 定界符为一个符号，如果有 Prefix，
// 则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，
```

```
// 然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始
getBucketRequest.setDelimiter("/");

cosXmlService.getBucketAsync(getBucketRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketResult getBucketResult = (GetBucketResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

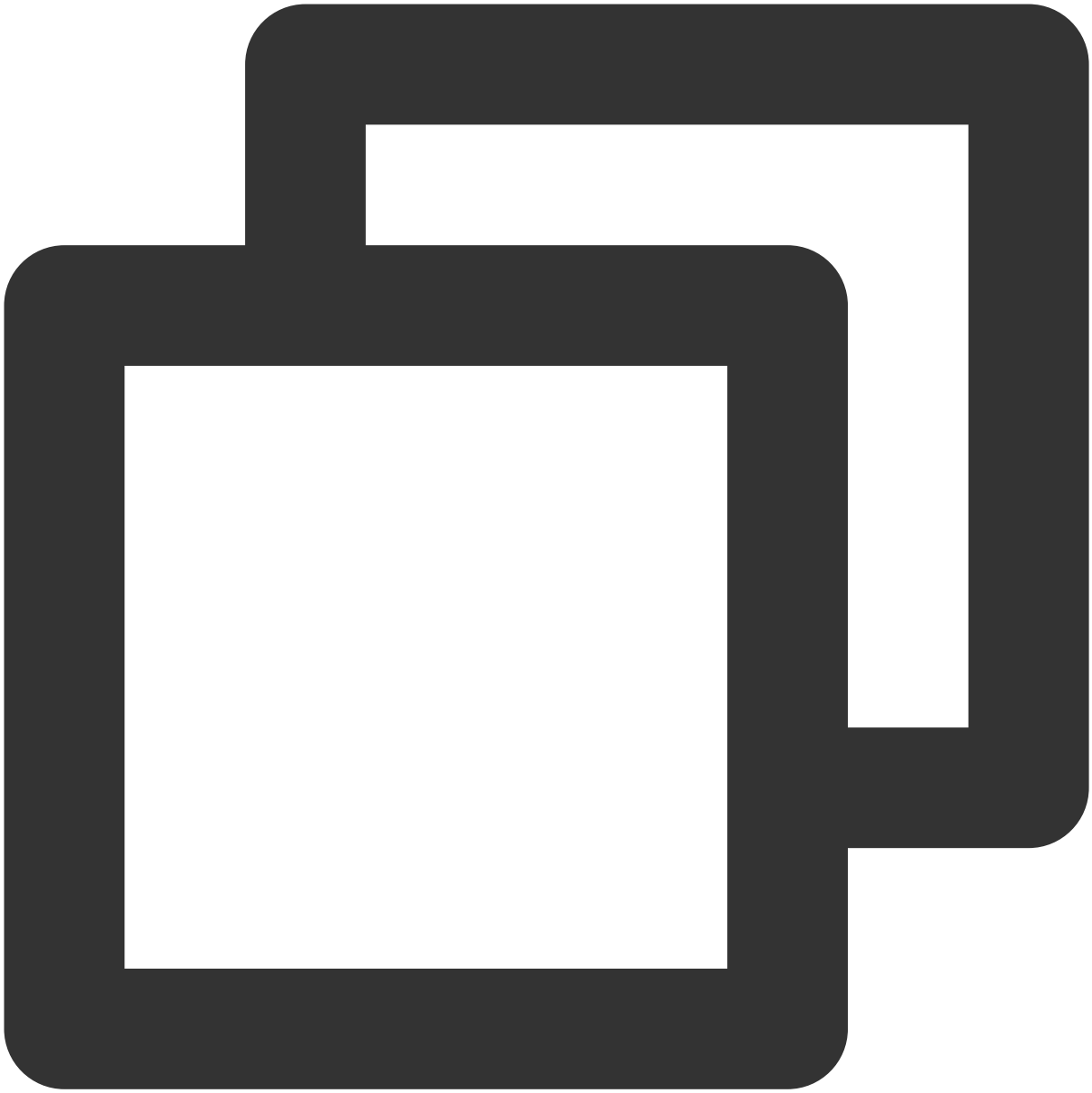
更多完整示例，请前往 GitHub 查看。

查询对象历史版本列表

功能说明

查询开启版本控制的存储桶下的部分或者全部对象。

示例代码一：获取对象历史版本列表第一页数据



```
String bucketName = "examplebucket-1250000000"; //格式：BucketName-APPID;
final GetBucketObjectVersionsRequest getBucketRequest =
        new GetBucketObjectVersionsRequest(bucketName);

// 前缀匹配，用来规定返回的对象前缀地址
getBucketRequest.setPrefix("dir/");

// 单次返回最大的条目数量，默认1000
getBucketRequest.setMaxKeys(100);

cosXmlService.getBucketObjectVersionsAsync(getBucketRequest,
```

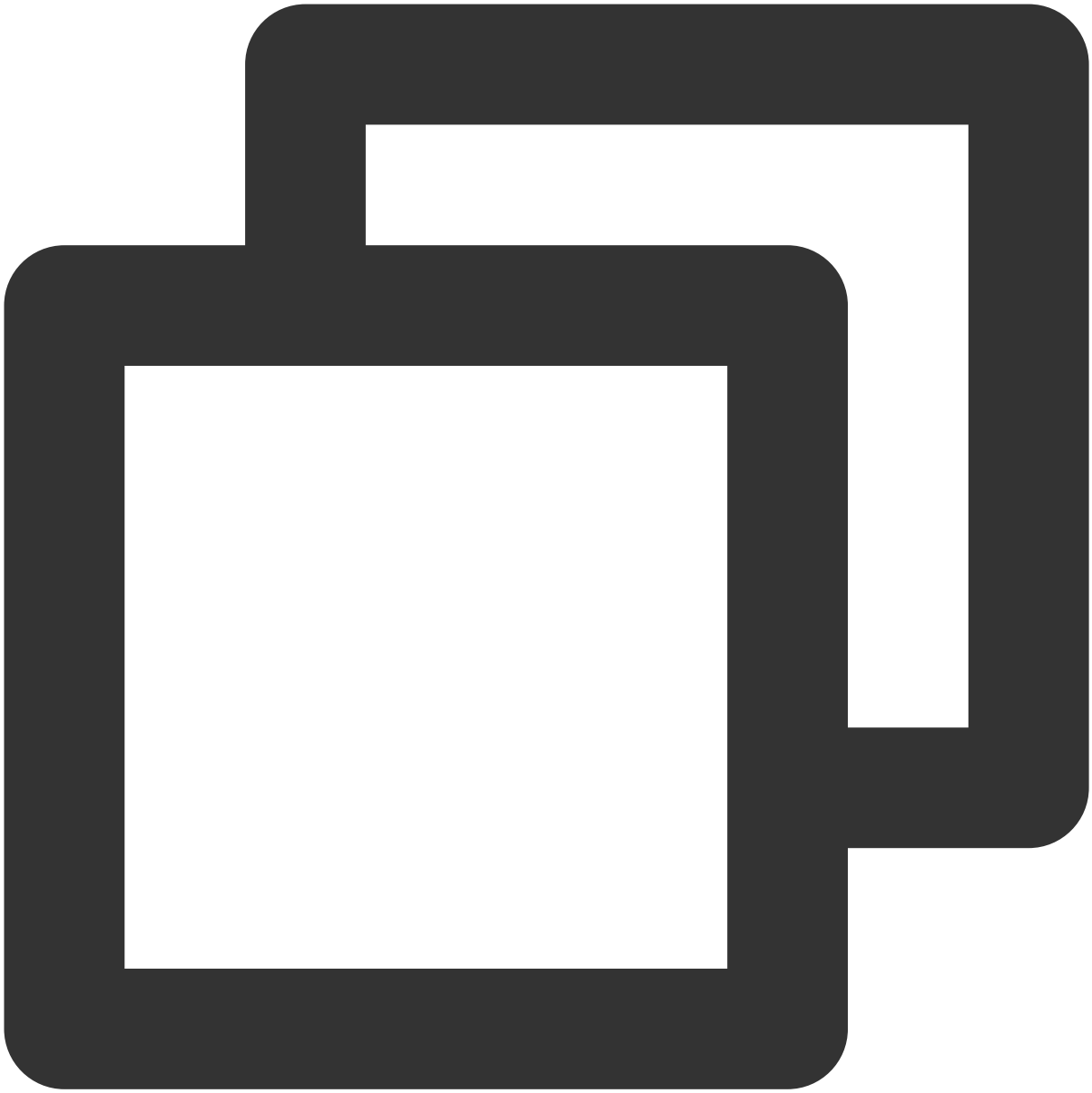
```
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketObjectVersionsResult getBucketResult =
                (GetBucketObjectVersionsResult) result;
        if (getBucketResult.listVersionResult.isTruncated) {
            // 表示数据被截断，需要拉取下一页数据
            prevPageResult = getBucketResult;
        }
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码二：获取对象历史版本列表下一页数据



```
String bucketName = "examplebucket-1250000000"; //格式：BucketName-APPID;
final GetBucketObjectVersionsRequest getBucketRequest =
        new GetBucketObjectVersionsRequest(bucketName);

// 前缀匹配，用来规定返回的对象前缀地址
getBucketRequest.setPrefix("dir/");

// 单次返回最大的条目数量，默认1000
getBucketRequest.setMaxKeys(100);

// prevPageResult 是上一页的返回结果，这里的 nextMarker 与 nextVersionIdMarker
```

```
// 表示下一页的起始位置
getBucketRequest.setKeyMarker(prevPageResult.listVersionResult
        .nextKeyMarker);
getBucketRequest.setVersionIdMarker(prevPageResult.listVersionResult
        .nextVersionIdMarker);

cosXmlService.getBucketObjectVersionsAsync(getBucketRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketObjectVersionsResult getBucketResult =
                (GetBucketObjectVersionsResult) result;
        if (getBucketResult.listVersionResult.isTruncated) {
            // 表示数据被截断，需要拉取下一页数据
            prevPageResult = getBucketResult;
        }
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

删除对象

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于对象的删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

SDK API 参考

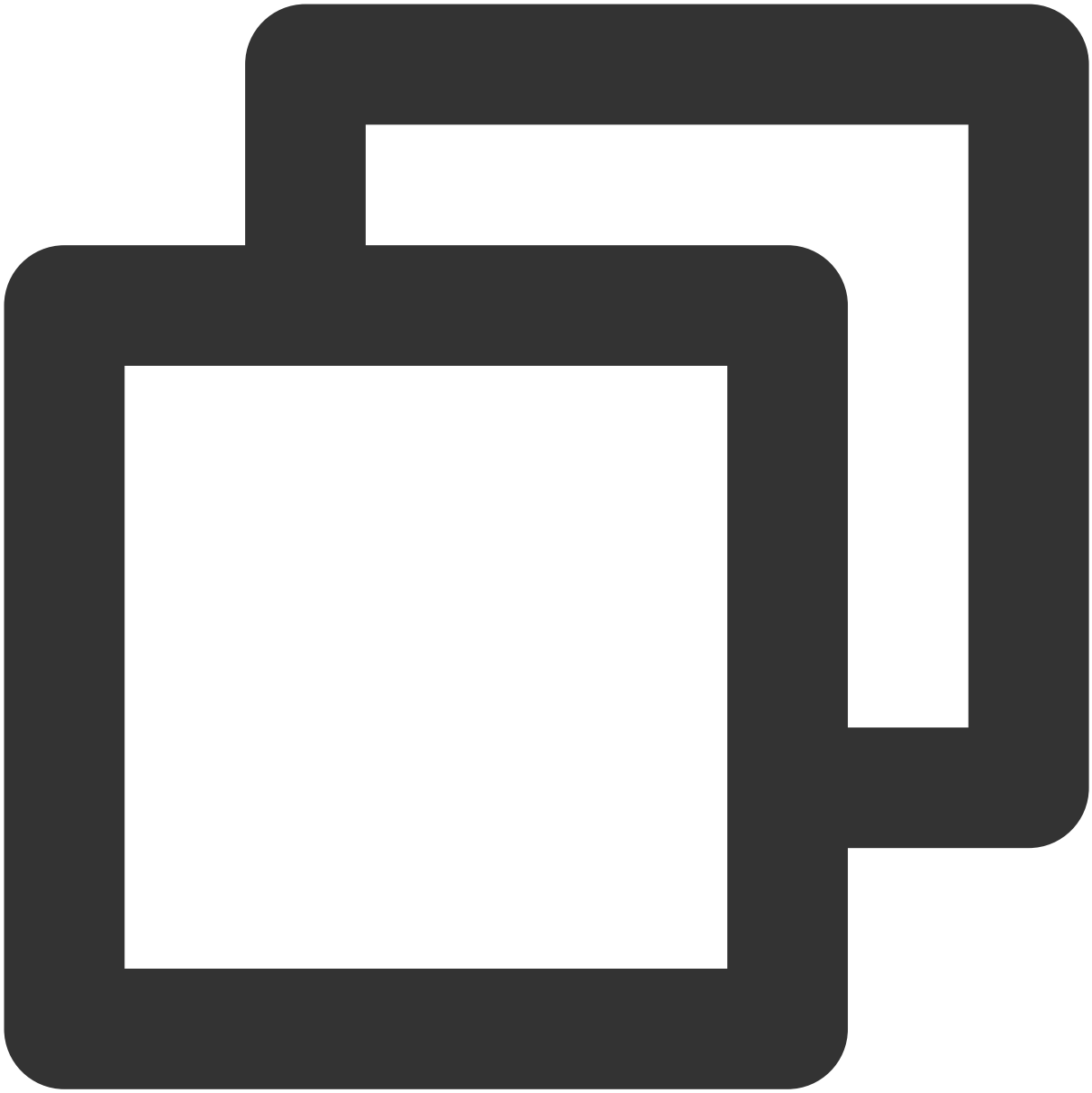
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

删除单个对象

功能说明

删除指定的对象（DELETE Object）。

示例代码



```
String bucket = "examplebucket-1250000000"; //存储桶名称，格式：BucketName-APPID
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键

DeleteObjectRequest deleteObjectRequest = new DeleteObjectRequest(bucket,
        cosPath);
cosXmlService.deleteObjectAsync(deleteObjectRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
        DeleteObjectResult deleteObjectResult = (DeleteObjectResult) result;
    }
```

```
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

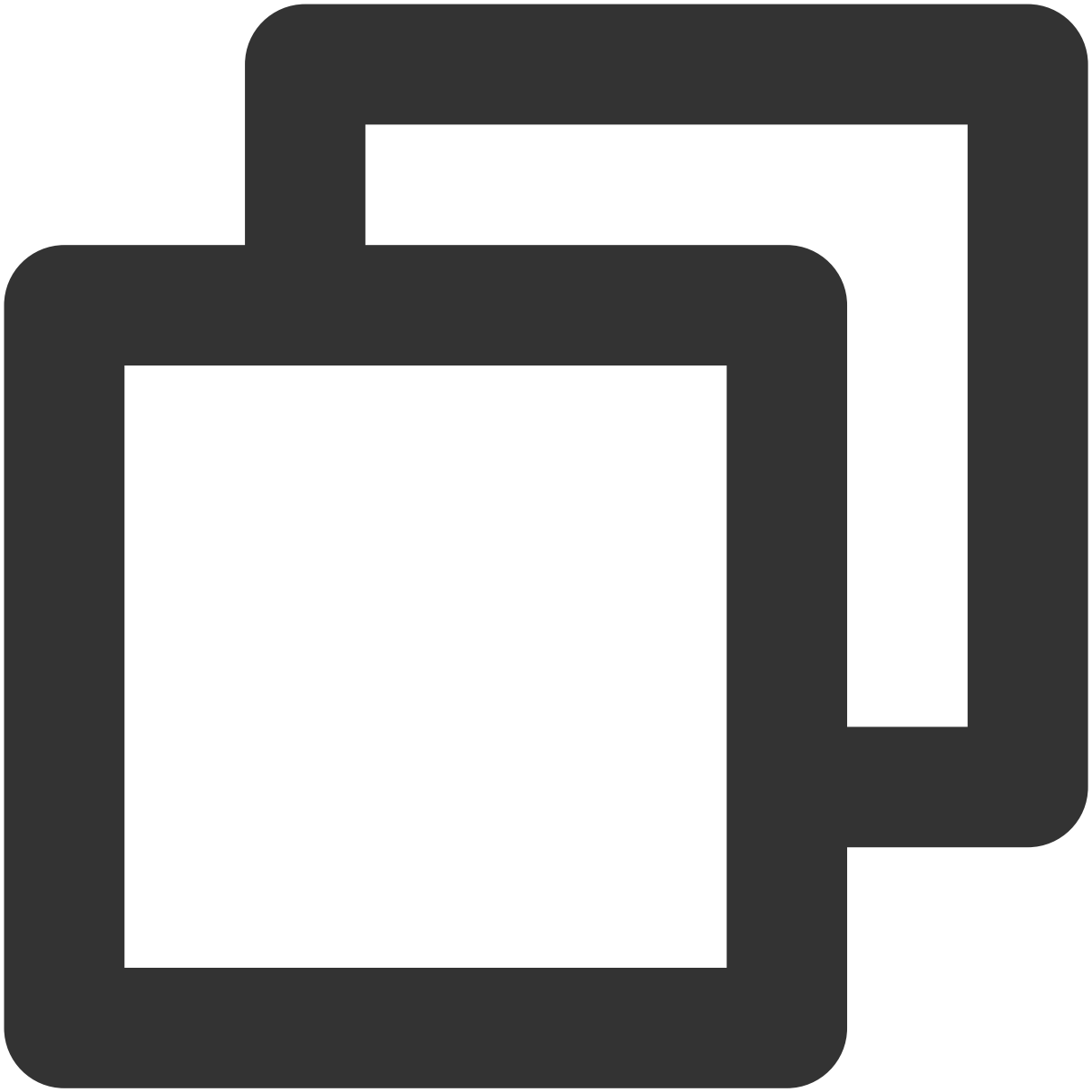
更多完整示例，请前往 GitHub 查看。

删除多个对象

功能说明

批量删除多个对象（Delete Multi Objects）。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
List<String> objectList = new ArrayList<String>();
objectList.add("exampleobject1"); //对象在存储桶中的位置标识符，即对象键
objectList.add("exampleobject2"); //对象在存储桶中的位置标识符，即对象键

DeleteMultiObjectRequest deleteMultiObjectRequest =
        new DeleteMultiObjectRequest(bucket, objectList);
// Quiet 模式只返回报错的 Object 信息。否则返回每个 Object 的删除结果。
deleteMultiObjectRequest.setQuiet(true);
cosXmlService.deleteMultiObjectAsync(deleteMultiObjectRequest,
```

```
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
        DeleteMultiObjectResult deleteMultiObjectResult =
                (DeleteMultiObjectResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

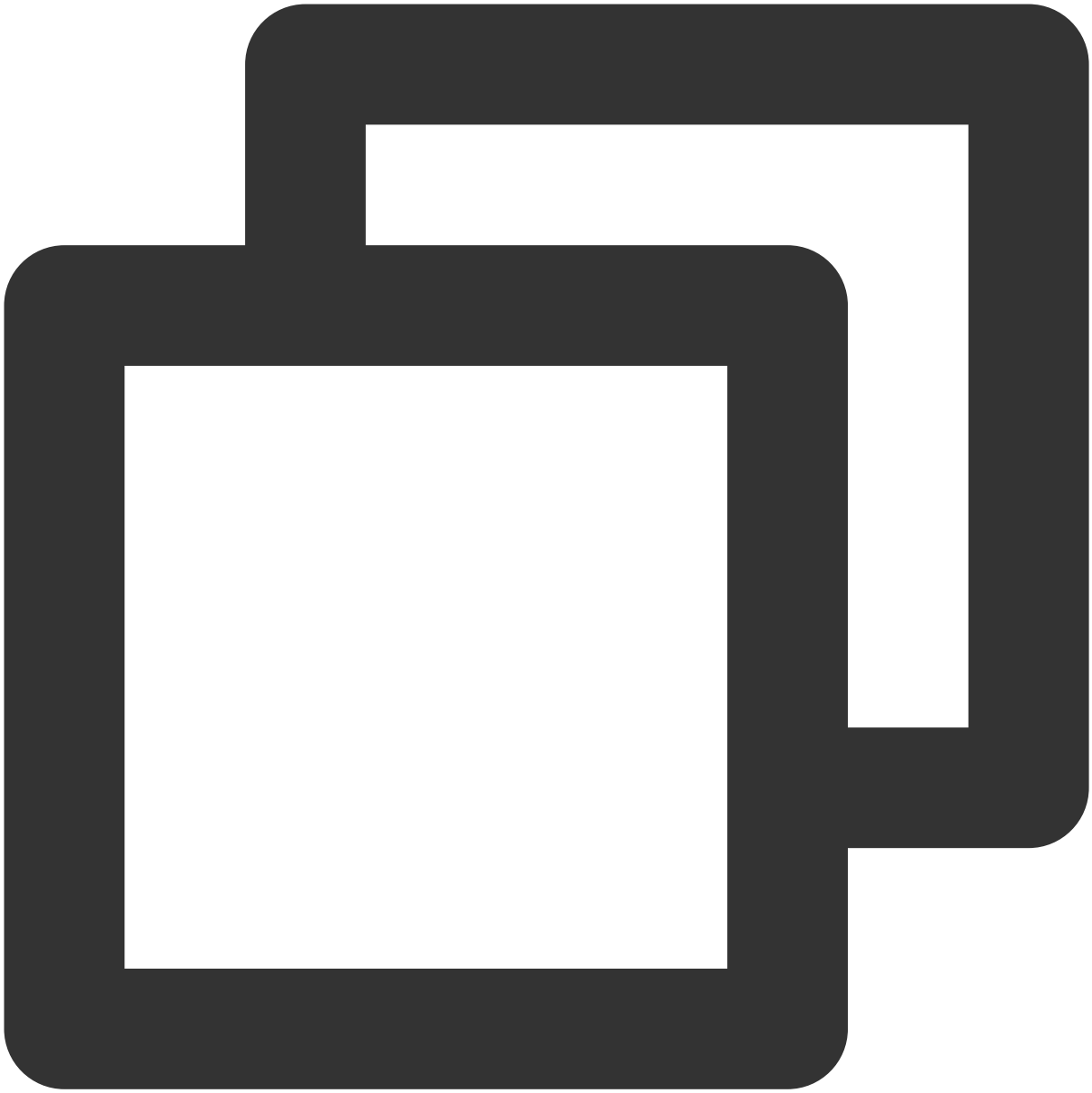
删除目录

功能说明

COS 上的文件夹概念是以 `/` 分隔对象名，形成类似文件系统的路径，从而模拟出来的。所以删除文件夹的操作，在 COS 上相当于删除一批有着同样前缀的对象。例如：文件夹 `prefix/` ，代表的是以 `prefix/` 为前缀的所有对象，所以删除 `prefix/` 意味着删除以 `prefix/` 为前缀的所有对象。

目前 COS Android SDK 没有提供一个接口去支持这样的操作，但是可以通过基本操作的组合，达到同样的效果。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String prefix = "folder1/"; //指定前缀

GetBucketRequest getBucketRequest = new GetBucketRequest(bucket);
getBucketRequest.setPrefix(prefix);

// prefix表示要删除的文件夹
getBucketRequest.setPrefix(prefix);
// 设置最大遍历出多少个对象，一次listobject最大支持1000
getBucketRequest.setMaxKeys(1000);
```

```
GetBucketResult getBucketResult = null;

do {
    try {
        getBucketResult = cosXmlService.getBucket(getBucketRequest);
        List<ListBucket.Contents> contents = getBucketResult.listBucket.contentsLis
        DeleteMultiObjectRequest deleteMultiObjectRequest = new DeleteMultiObjectRe
        for (ListBucket.Contents content : contents) {
            deleteMultiObjectRequest.setObjectList(content.key);
        }
        cosXmlService.deleteMultiObject(deleteMultiObjectRequest);
        getBucketRequest.setMarker(getBucketResult.listBucket.nextMarker);
    } catch (CosXmlClientException e) {
        e.printStackTrace();
        return;
    } catch (CosXmlServiceException e) {
        e.printStackTrace();
        return;
    }
} while (getBucketResult.listBucket.isTruncated);
```

说明

更多完整示例，请前往 GitHub 查看。

恢复归档对象

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于恢复归档对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
POST Object restore	恢复归档对象	将归档类型的对象取回访问

SDK API 参考

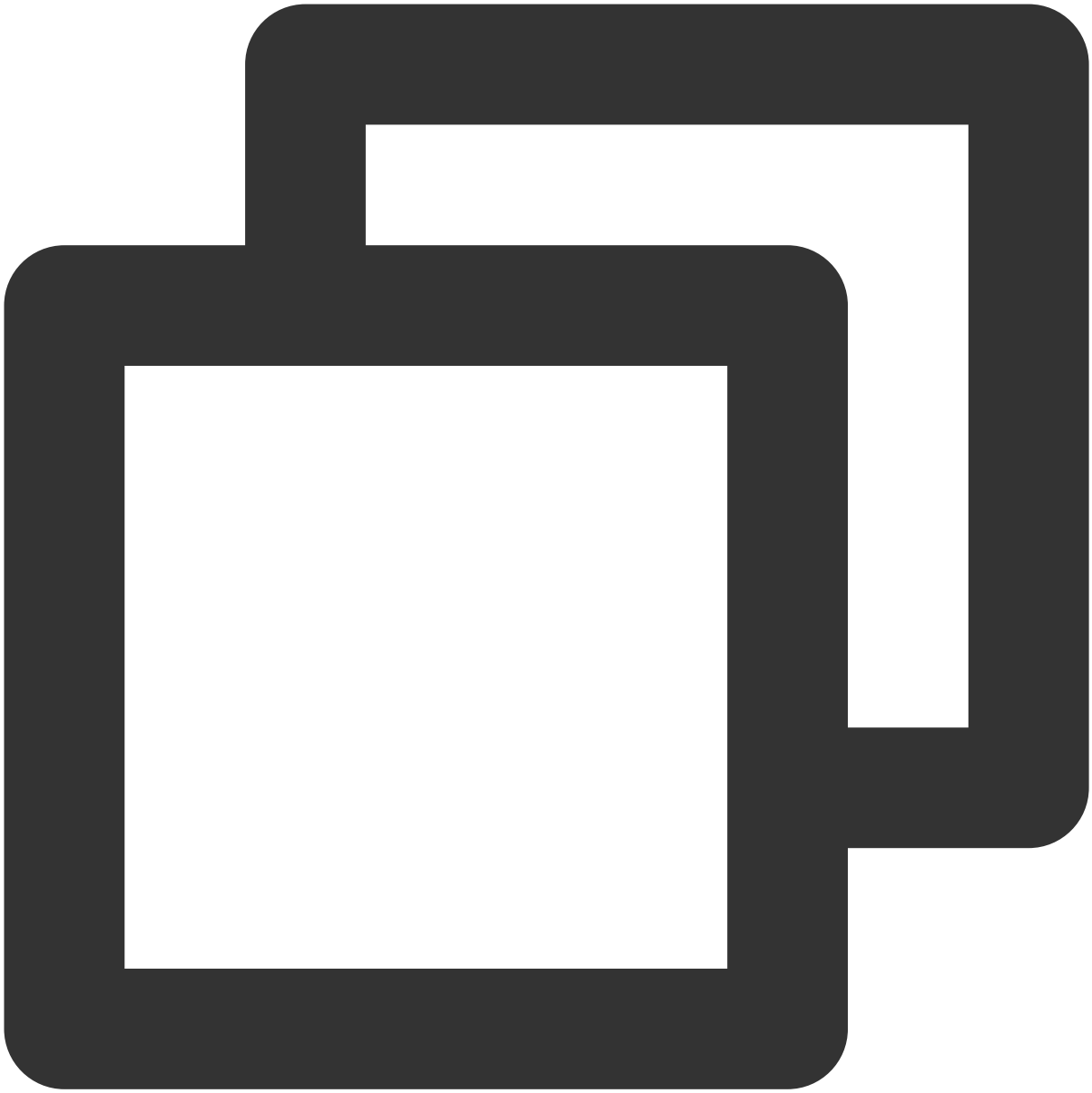
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

恢复归档对象

功能说明

将归档类型的对象取回访问（POST Object restore）。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键。
RestoreRequest restoreRequest = new RestoreRequest(bucket, cosPath);
restoreRequest.setExpireDays(5); // 保留5天
restoreRequest.setTier(RestoreConfigure.Tier.Standard); // 标准恢复模式

cosXmlService.restoreObjectAsync(restoreRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        RestoreResult restoreResult = (RestoreResult) result;
```

```
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

查询对象元数据

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于查询对象元数据操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

SDK API 参考

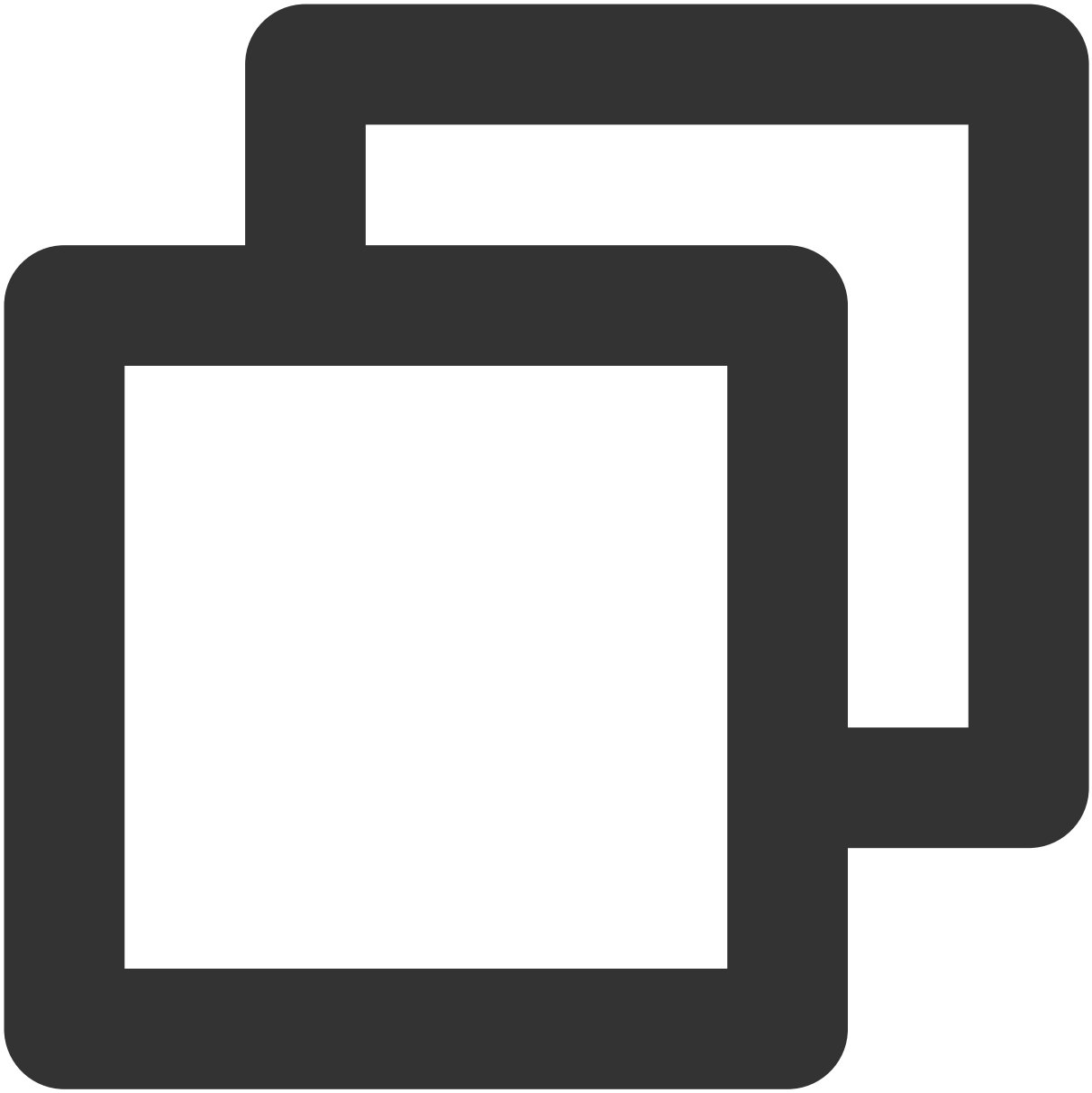
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

查询对象元数据

功能说明

查询 Object 的 Meta 信息。

示例代码



```
String bucket = "examplebucket-1250000000"; //存储桶名称，格式：BucketName-APPID
String cosPath = "exampleobject"; //对象位于存储桶中的位置标识符，即对象键
HeadObjectRequest headObjectRequest = new HeadObjectRequest(bucket, cosPath);
cosXmlService.headObjectAsync(headObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        HeadObjectResult headObjectResult = (HeadObjectResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
```

```
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

生成预签名 URL

最近更新时间：2024-01-04 15:52:02

简介

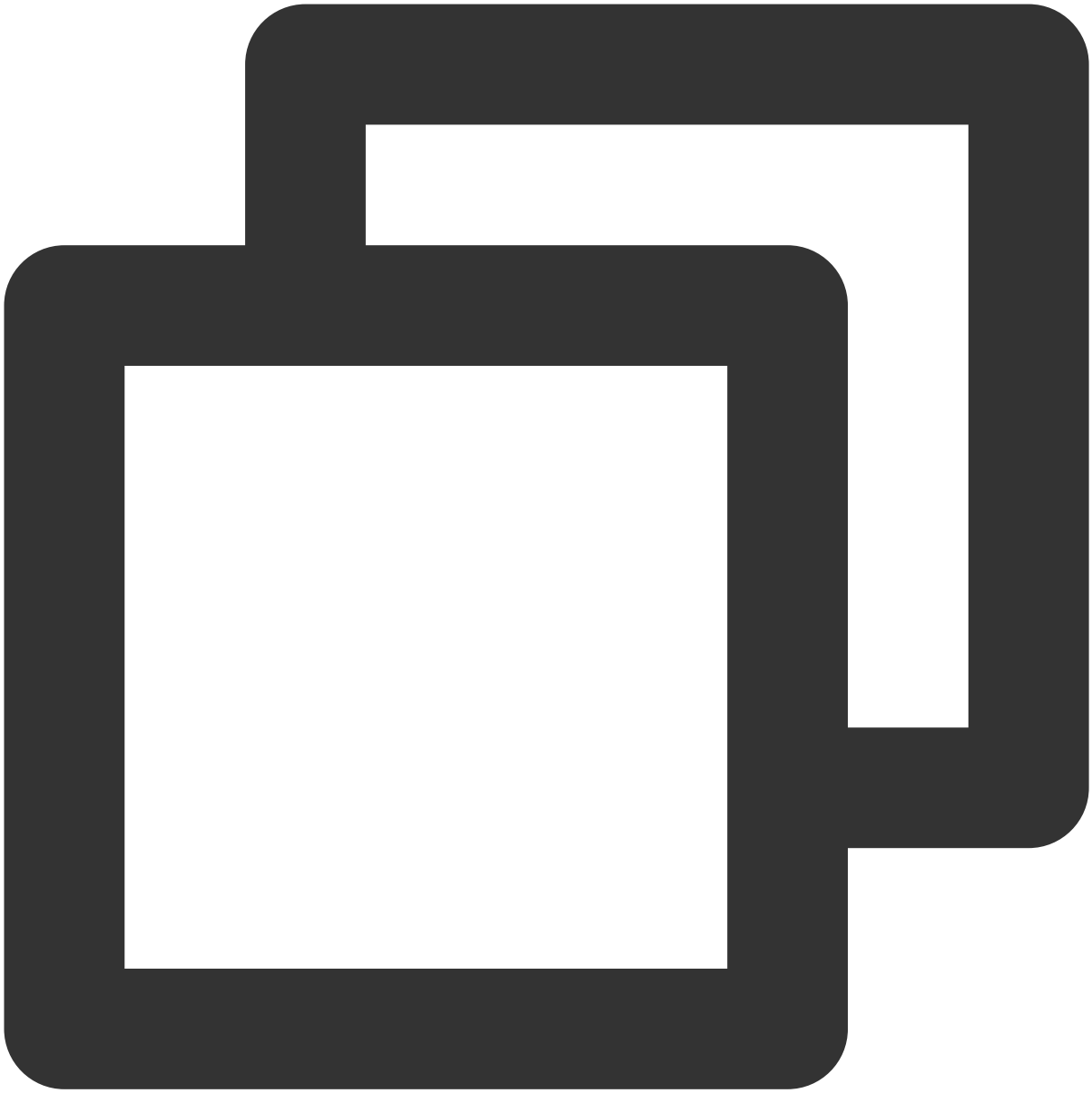
本文档提供关于生成对象预签名链接的示例代码。
关于使用预签名 URL 上传的说明请参见 预签名授权上传， 使用预签名 URL 下载的说明请参见 预签名授权下载。
说明
建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄露目标存储桶或对象之外的资源。
如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

生成对象预签名链接

示例代码一：生成预签名上传链接



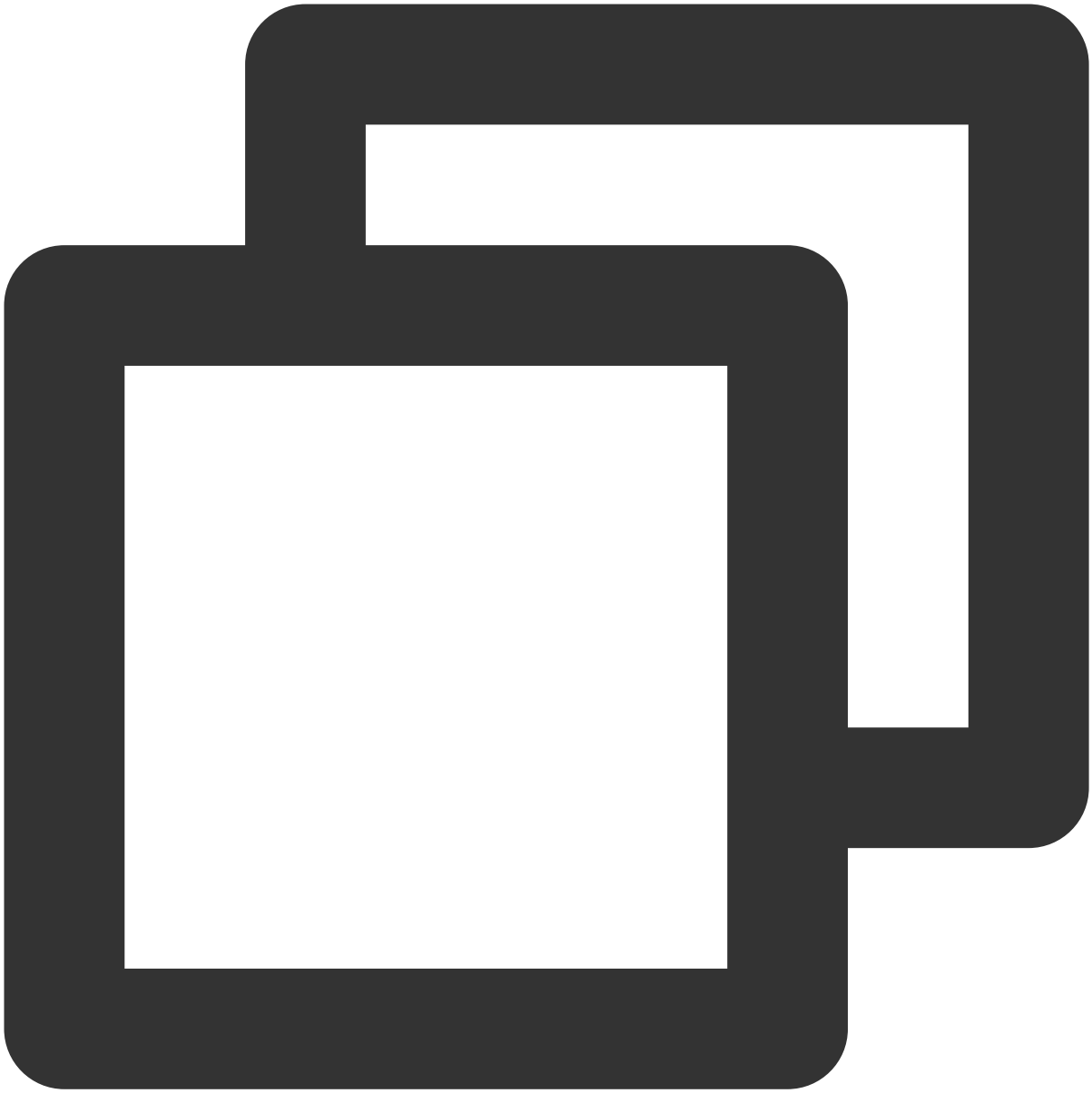
```
try {
      //存储桶名称
    String bucket = "examplebucket-1250000000";
    // 对象在存储桶中的位置标识符，对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](ht
    // 注意：用户无需对 cosPath 进行编码操作
    String cosPath = "exampleobject";
    //请求 HTTP 方法
    String method = "PUT";
    PresignedUrlRequest presignedUrlRequest = new PresignedUrlRequest(bucket
            , cosPath) {
        @Override
```

```
            public RequestBodySerializer getRequestBody()
                    throws CosXmlClientException {
                //用于计算 put 等需要带上 body 的请求的签名 URL
                return RequestBodySerializer.string("text/plain",
                        "this is test");
            }
        };
        presignedUrlRequest.setRequestMethod(method);
        // 设置签名有效期为 60s，注意这里是签名有效期，您需要自行保证密钥有效期
        presignedUrlRequest.setSignKeyTime(60);
        // 设置不签名 Host
        presignedUrlRequest.addNoSignHeader("Host");
        String urlWithSign = cosXmlService.getPresignedURL(presignedUrlRequest);
    } catch (CosXmlClientException e) {
        e.printStackTrace();
    }
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码二：生成预签名下载链接



```
try {
    //存储桶名称
    String bucket = "examplebucket-1250000000";
    // 对象在存储桶中的位置标识符，对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](ht
    // 注意：用户无需对 cosPath 进行编码操作
    String cosPath = "exampleobject";
    //请求 HTTP 方法.
    String method = "GET";
    PresignedUrlRequest presignedUrlRequest = new PresignedUrlRequest(bucket
            , cosPath);
    presignedUrlRequest.setRequestMethod(method);
```

```
    // 设置签名有效期为 60s，注意这里是签名有效期，您需要自行保证密钥有效期
    presignedUrlRequest.setSignKeyTime(60);
    // 设置不签名 Host
    presignedUrlRequest.addNoSignHeader("Host");

    String urlWithSign = cosXmlService.getPresignedURL(presignedUrlRequest);

} catch (CosXmlClientException e) {
    e.printStackTrace();
}
```

说明

更多完整示例，请前往 GitHub 查看。

预请求跨域配置

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于预请求跨域配置操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
Options Object	预请求跨域配置	用预请求来确认是否可以发送真正的跨域请求

SDK API 参考

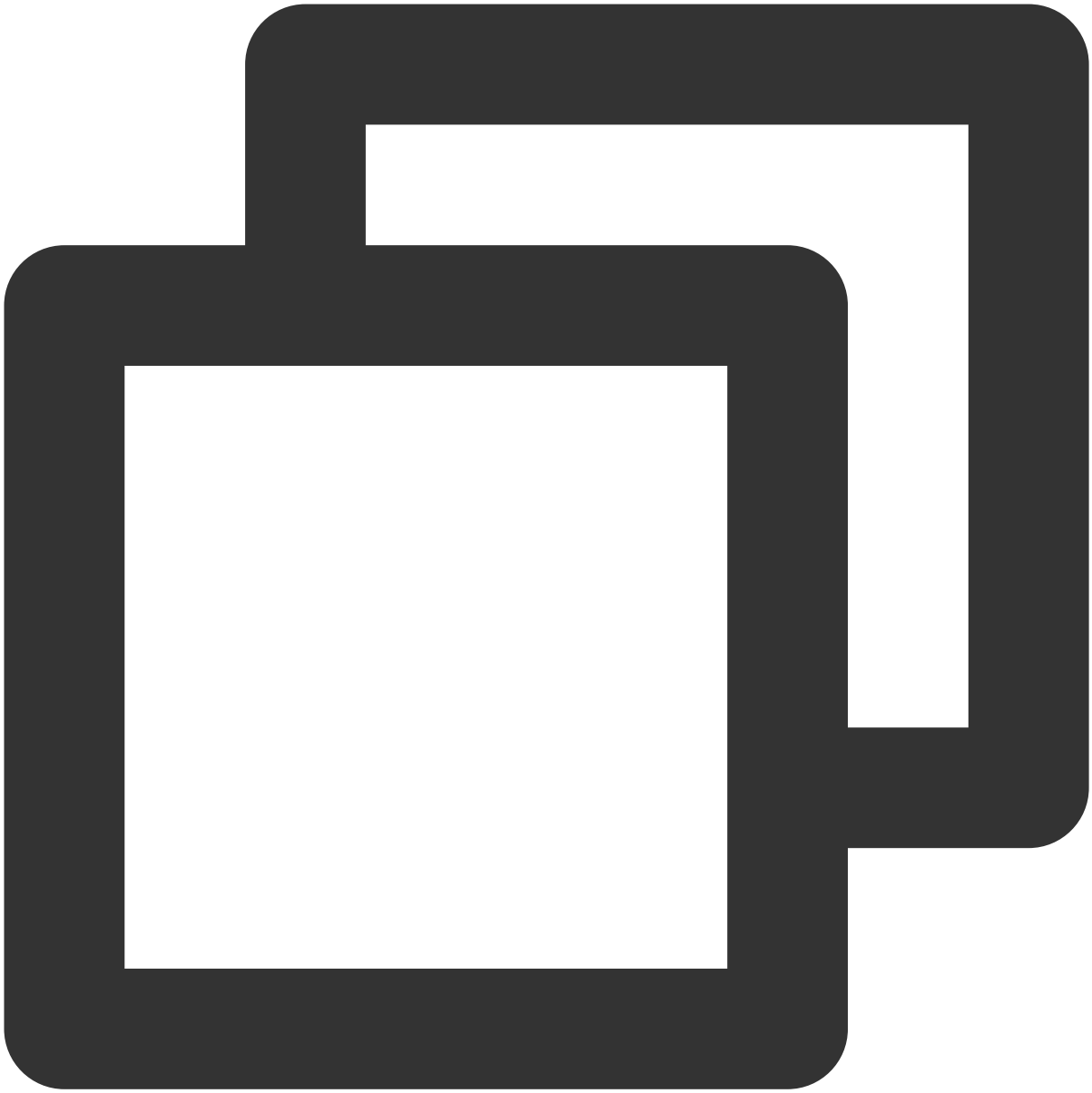
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

预请求跨域配置

功能说明

获取预请求跨域配置（Options Object）。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象位于存储桶中的位置标识符，即对象键
String origin = "https://cloud.tencent.com";
String accessMethod = "PUT";
OptionObjectRequest optionObjectRequest = new OptionObjectRequest(bucket,
        cosPath, origin,
        accessMethod);
cosXmlService.optionObjectAsync(optionObjectRequest,
        new CosXmlResultListener() {
    @Override
```

```
    public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
        OptionObjectResult optionObjectResult = (OptionObjectResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

服务端加密

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于如何使用在上传对象时开启服务端加密。服务端加密的密钥分为三种：
COS 托管加密密钥
KMS 托管加密密钥
客户提供的加密密钥

SDK API 参考

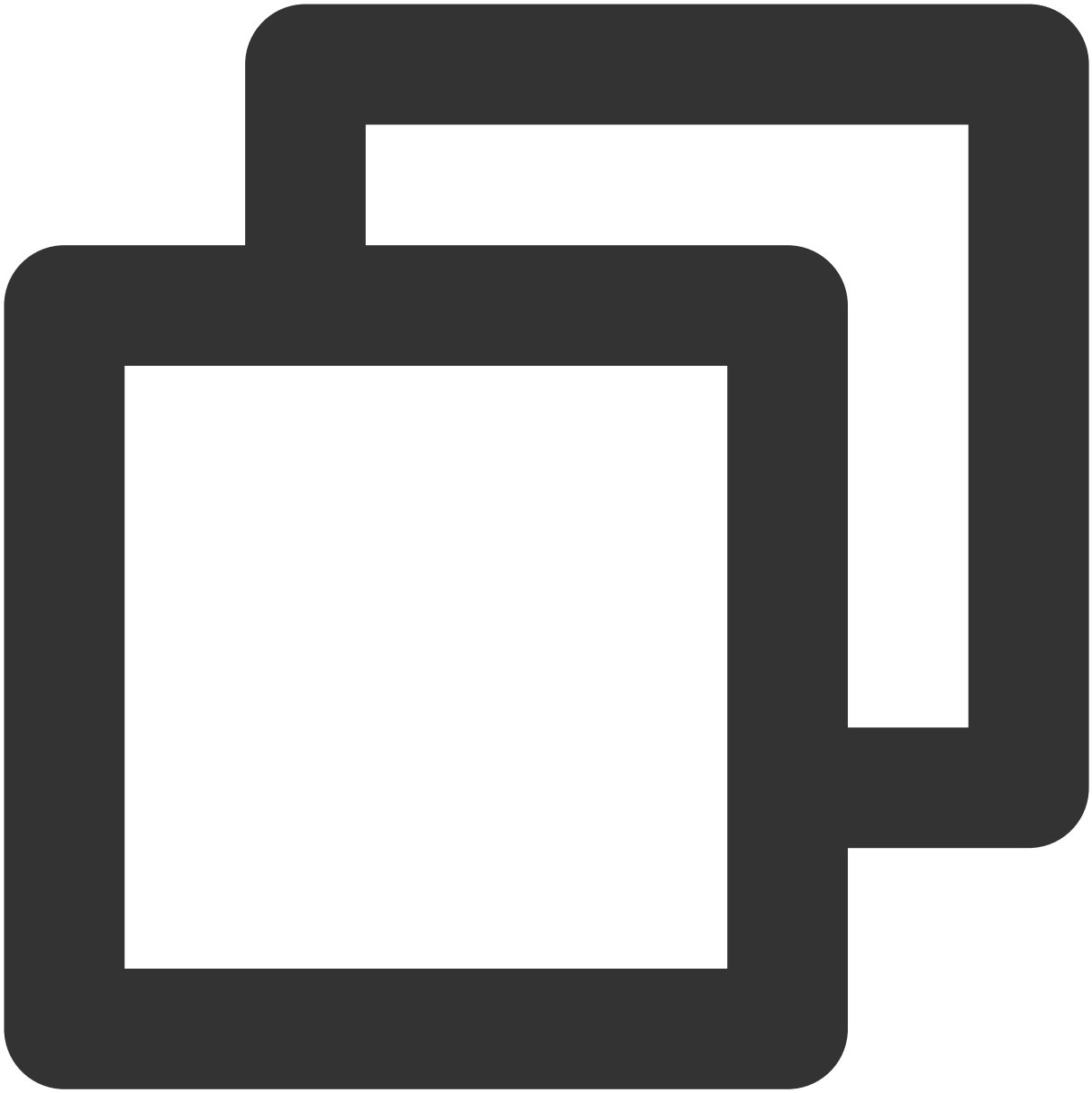
SDK 所有接口的具体参数与方法说明，请参考 SDK API。

使用 COS 托管加密密钥的服务端加密（SSE-COS）保护数据

功能说明

由腾讯云 COS 托管主密钥和管理数据。COS 会帮助您在数据写入数据中心时自动加密，并在您取用该数据时自动解密。目前支持使用 COS 主密钥对数据进行 AES-256 加密。

示例代码



```
PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, srcPath);
// 设置使用 COS 托管加密密钥的服务端加密（SSE-COS）保护数据
putObjectRequest.setCOSServerSideEncryption();

// 上传文件
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(putObjectRequest, upload
```

说明

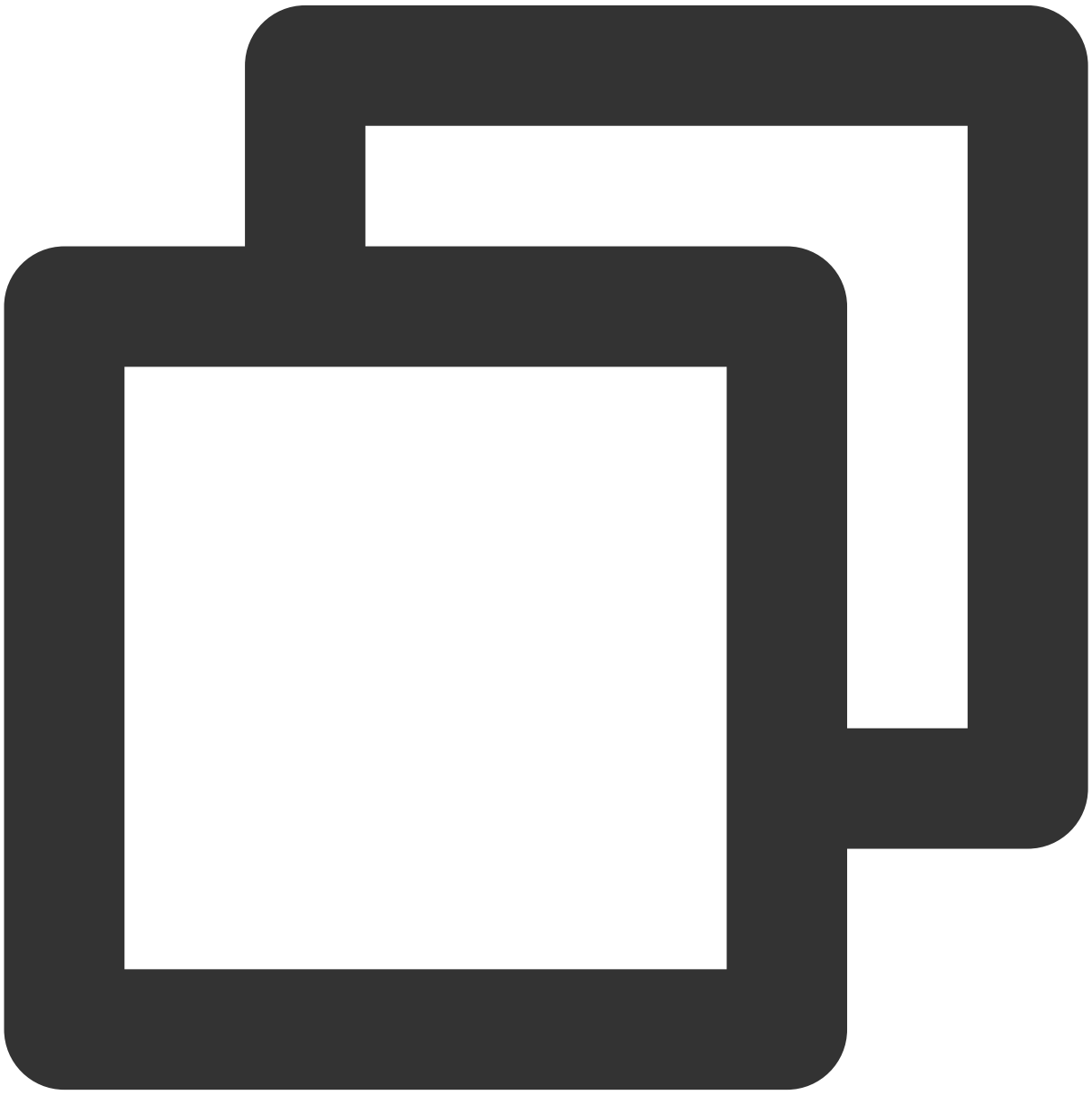
更多完整示例，请前往 GitHub 查看。

使用 KMS 托管加密密钥的服务端加密（SSE-KMS）保护数据

功能说明

SSE-KMS 加密即使用 KMS 托管密钥的服务端加密。KMS 是腾讯云推出的一款安全管理类服务，使用经过第三方认证的硬件安全模块 HSM（Hardware Security Module）来生成和保护密钥。它能够帮助用户轻松创建和管理密钥，满足用户多应用多业务的密钥管理需求以及满足监管和合规要求。关于如何开通 KMS 服务请参考：服务端加密概述。

示例代码



```
// 服务端加密密钥
String customKey = "用户主密钥 CMK";
String encryptContext = "加密上下文";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, srcPath);

// 设置使用客户提供的用户主密钥的服务端加密 （SSE-KMS）保护数据
try {
    putObjectRequest.setCOSServerSideEncryptionWithKMS(customKey, encryptContext);
} catch (CosXmlClientException e) {
    e.printStackTrace();
}
// 上传文件
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(putObjectRequest, upload
```

说明

更多完整示例，请前往 GitHub 查看。

使用客户提供的加密密钥的服务端加密 （SSE-C）保护数据

功能说明

加密密钥由用户自己提供，用户在上传对象时，COS 将使用用户提供的加密密钥对用户的数据进行 AES-256 加密。

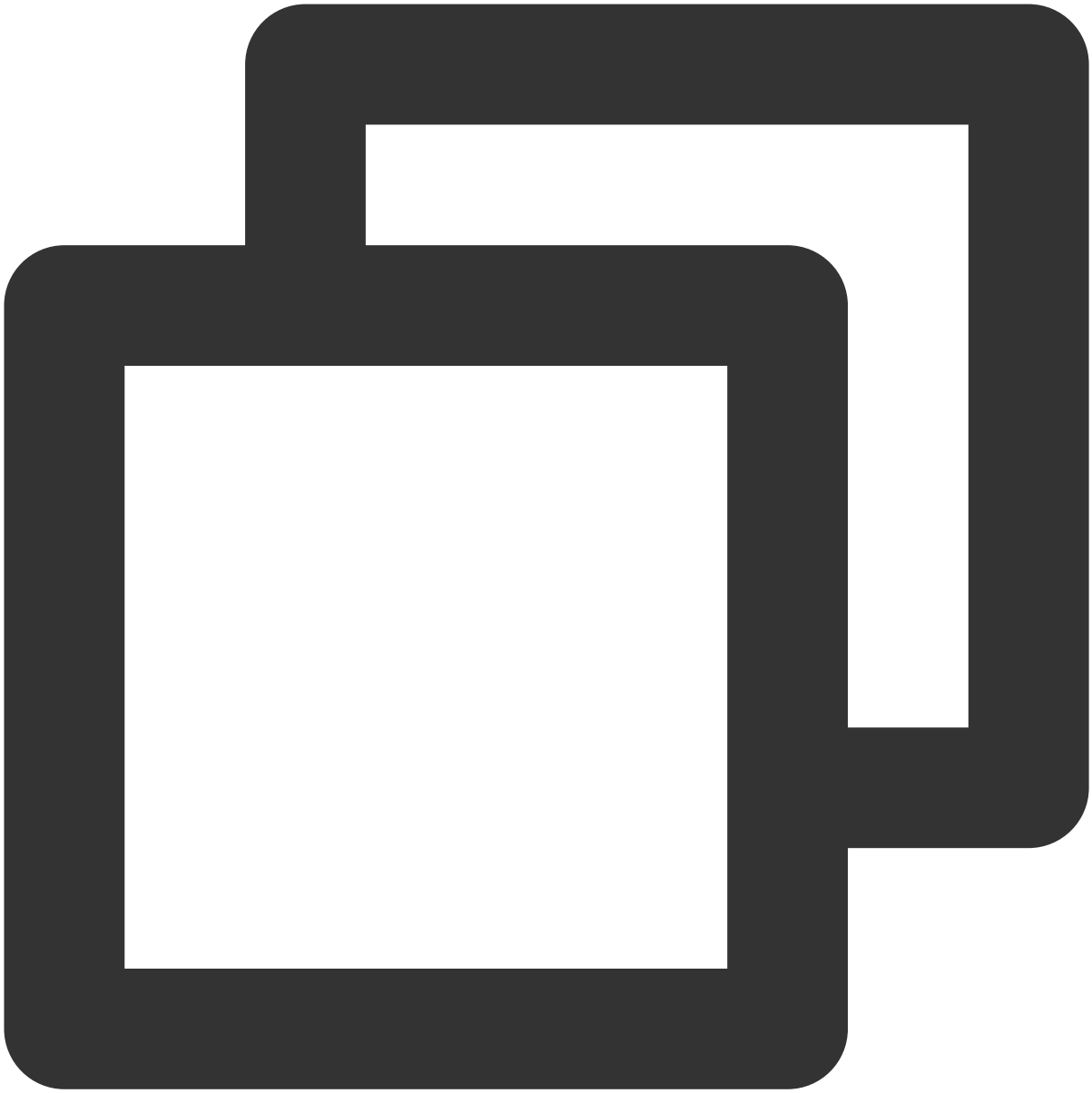
注意

该加密所运行的服务需要使用 HTTPS 请求。

用户需要提供一个32字节的字符串作为密钥，支持数字、字母、字符的组合，不支持中文。

如果上传文件时设置了密钥加密，那么在使用 GET（下载）、HEAD（查询）源对象时也需要在请求中带上相同的密钥，才能正常响应。

示例代码



```
// 服务端加密密钥
String customKey = "服务端加密密钥";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, srcPath);
// 设置使用客户提供的加密密钥的服务端加密 （SSE-C）保护数据
try {
    putObjectRequest.setCOSServerSideEncryptionWithCustomerKey(customKey);
} catch (CosXmlClientException e) {
    e.printStackTrace();
}

// 上传文件
```

```
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(putObjectRequest, upload
```

说明

更多完整示例，请前往 GitHub 查看。

单链接限速

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于调用上传下载接口时对链接进行限速。

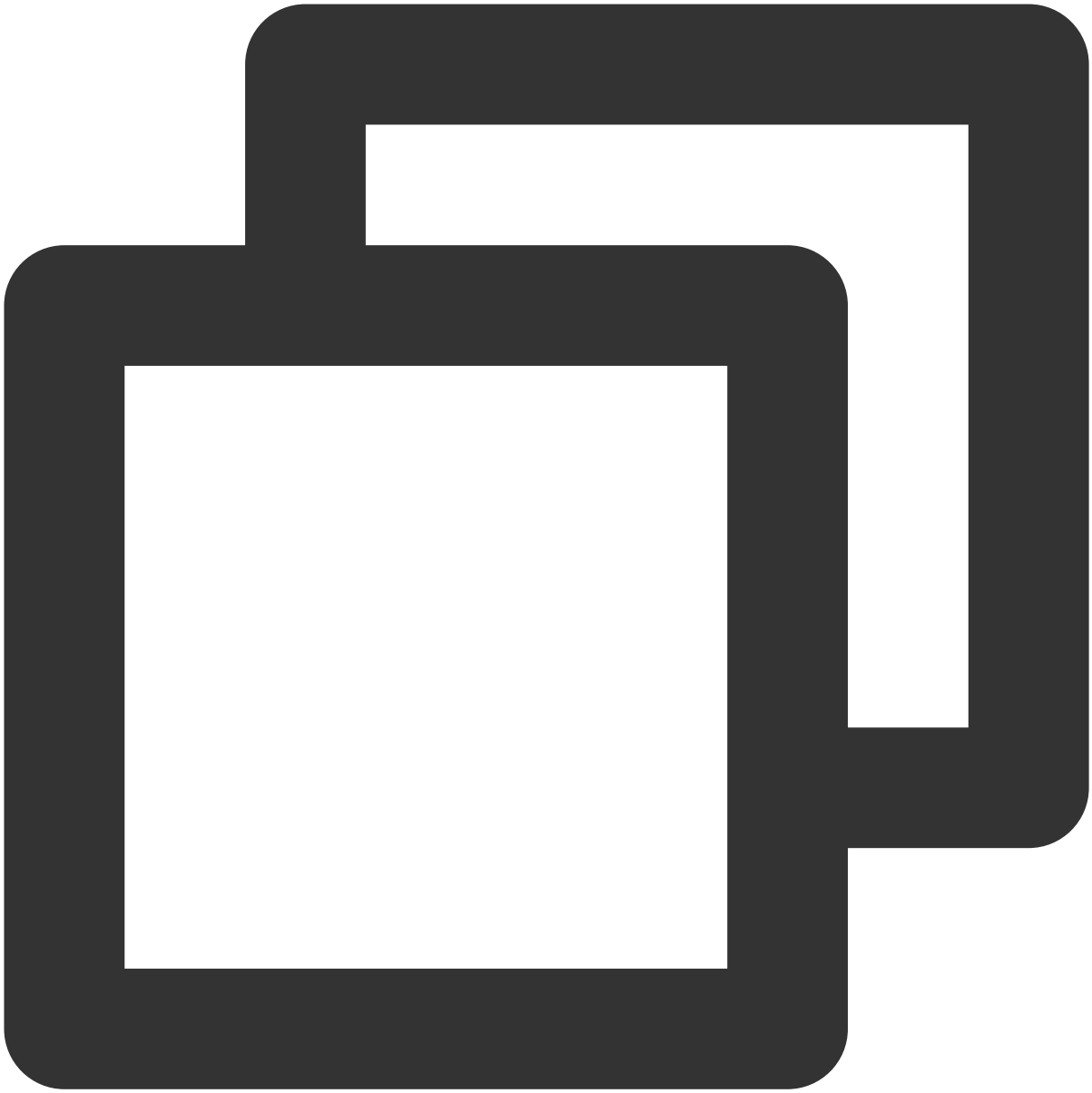
SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

使用说明

限速值设置范围为 **819200 - 838860800**，单位默认为 bit/s，即100KB/s - 100MB/s，如果超出该范围将返回400错误。

示例代码一：上传时对单链接限速



```
TransferConfig transferConfig = new TransferConfig.Builder().build();
// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);

// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String srcPath = new File(context.getCacheDir(), "exampleobject")
        .toString(); //本地文件的绝对路径
//若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传；否则，赋值 null
```

```
String uploadId = null;

PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, srcPath);
// 设置单链接限速，单位为 bit/s，示例设置为 1M/s
putObjectRequest.setTrafficLimit(1024 * 1024 * 8);

// 上传文件
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(putObjectRequest, upload

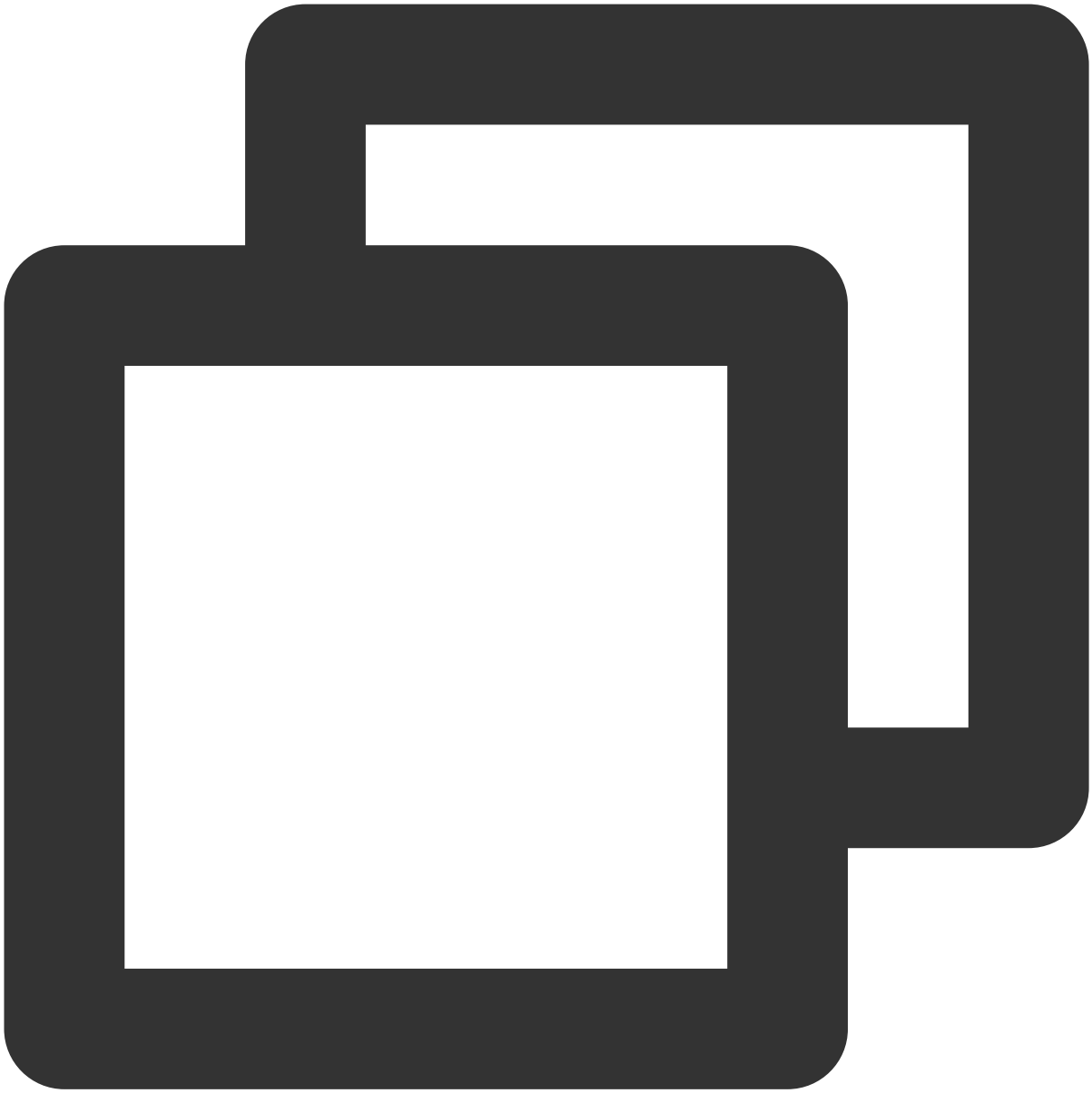
//设置上传进度回调
cosxmlUploadTask.setCosXmlProgressListener(new CosXmlProgressListener() {
    @Override
    public void onProgress(long complete, long target) {
        // todo Do something to update progress...
    }
});
//设置返回结果回调
cosxmlUploadTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        COSXMLUploadTask.COSXMLUploadTaskResult uploadResult =
                (COSXMLUploadTask.COSXMLUploadTaskResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
//设置任务状态回调，可以查看任务过程
cosxmlUploadTask.setTransferStateListener(new TransferStateListener() {
    @Override
    public void onStateChanged(TransferState state) {
        // todo notify transfer state
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码二：下载时对单链接限速



```
//.cssg-snippet-body-start:[transfer-download-object]
// 高级下载接口支持断点续传，所以会在下载前先发起 HEAD 请求获取文件信息。
// 如果您使用的是临时密钥或者使用子账号访问，请确保权限列表中包含 HeadObject 的权限。

// 初始化 TransferConfig，这里使用默认配置，如果需要定制，请参考 SDK 接口文档
TransferConfig transferConfig = new TransferConfig.Builder().build();
//初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);
```

```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
//本地目录路径
String savePathDir = context.getExternalCacheDir().toString();
//本地保存的文件名，若不填（null），则与 COS 上的文件名一样
String savedFileName = "exampleobject";

GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, cosPath, savePathD
// 设置单链接限速，单位为 bit/s，示例设置为 1M/s
getObjectRequest.setTrafficLimit(1024 * 1024 * 8);

Context applicationContext = context.getApplicationContext(); // application
// context
COSXMLDownloadTask cosxmlDownloadTask =
        transferManager.download(applicationContext, getObjectRequest);

//设置下载进度回调
cosxmlDownloadTask.setCosXmlProgressListener(new CosXmlProgressListener() {
    @Override
    public void onProgress(long complete, long target) {
        // todo Do something to update progress...
    }
});
//设置返回结果回调
cosxmlDownloadTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        COSXMLDownloadTask.COSXMLDownloadTaskResult downloadTaskResult =
                (COSXMLDownloadTask.COSXMLDownloadTaskResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
//设置任务状态回调，可以查看任务过程
```

```
cosxmlDownloadTask.setTransferStateListener(new TransferStateListener() {
    @Override
    public void onStateChanged(TransferState state) {
        // todo notify transfer state
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

检索对象内容

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于检索对象内容操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
SELECT Object Content	检索对象内容	从指定对象（CSV 格式或者 JSON 格式）中检索内容

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

检索对象内容

功能说明

COS Select 支持检索以下格式的对象数据：

CSV 格式：对象以 CSV 格式存储，并以固定的分隔符划分。

JSON 格式：对象以 JSON 格式存储，可以是 JSON 文件或者 JSON 列表。

注意

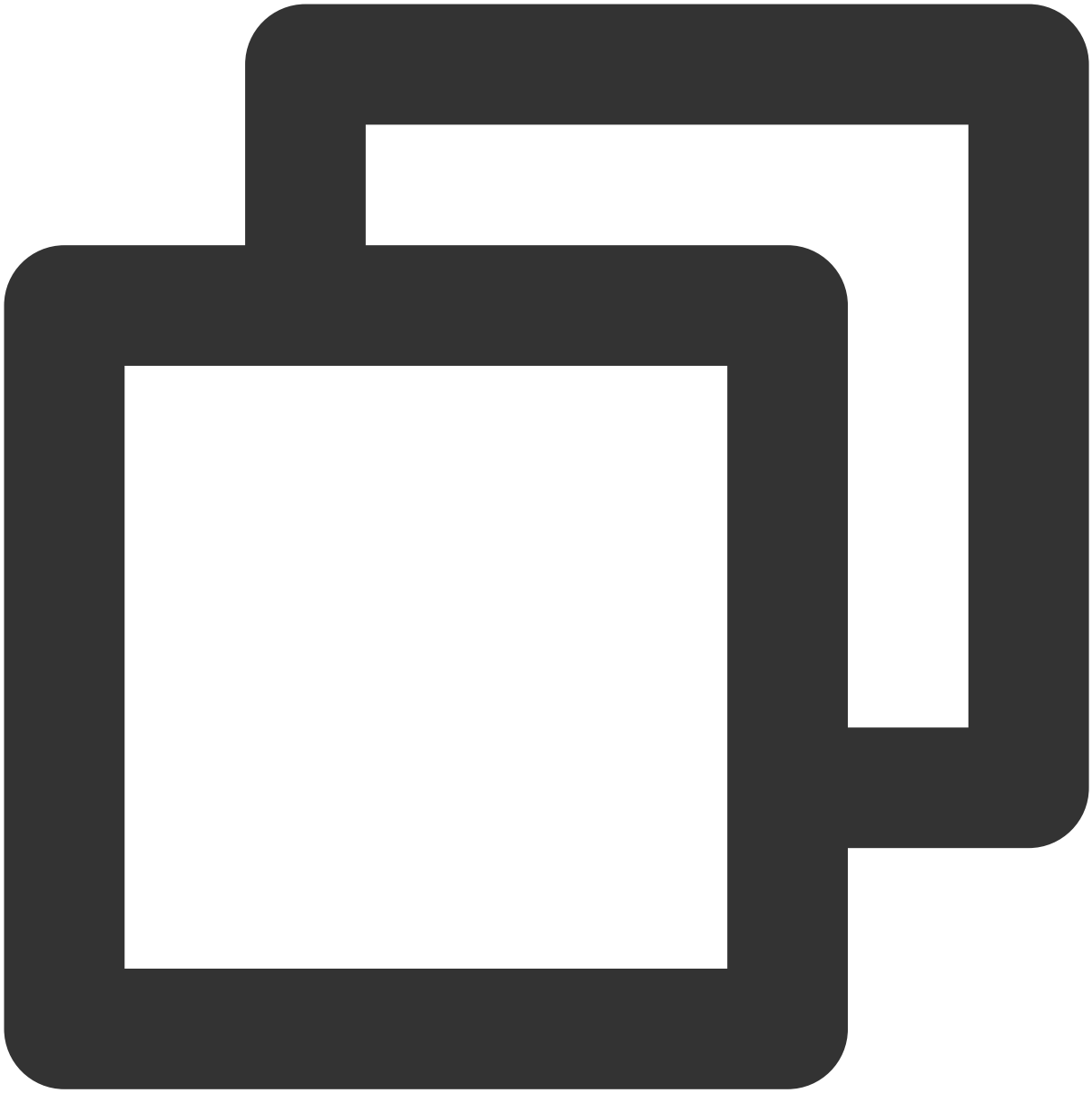
使用 COS Select，您必须具有 `cos:GetObject` 的授权。

CSV、JSON 对象需要以 UTF-8 格式编码。

COS Select 支持检索 GZIP 或者 BZIP2 压缩的 CSV、JSON 对象。

COS Select 支持检索 SSE-COS 加密的 CSV、JSON 对象。

示例代码



```
String bucket = "examplebucket-1250000000";
// 对象必须为 JSON 或者 csv 格式的文件
String cosPath = "exampleobject";
final String expression = "Select * from COSObject";

SelectObjectContentRequest selectObjectContentRequest = new SelectObjectContentRequ
        bucket, cosPath, expression, true,
        new InputSerialization(CompressionType.NONE, new JSONInput(JSONType.DOCUMEN
        new OutputSerialization(new JSONOutput(","))
);
```

```
// 设置查询结果回调，可能会回调多次
selectObjectContentRequest.setSelectObjectContentProgressListener(new SelectObjectC
    @Override
    public void onProcess(SelectObjectContentEvent event) {

    }
});
cosXmlService.selectObjectContentAsync(selectObjectContentRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        SelectObjectContentResult selectObjectContentResult =
                (SelectObjectContentResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

异地容灾
版本控制

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于版本控制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

SDK API 参考

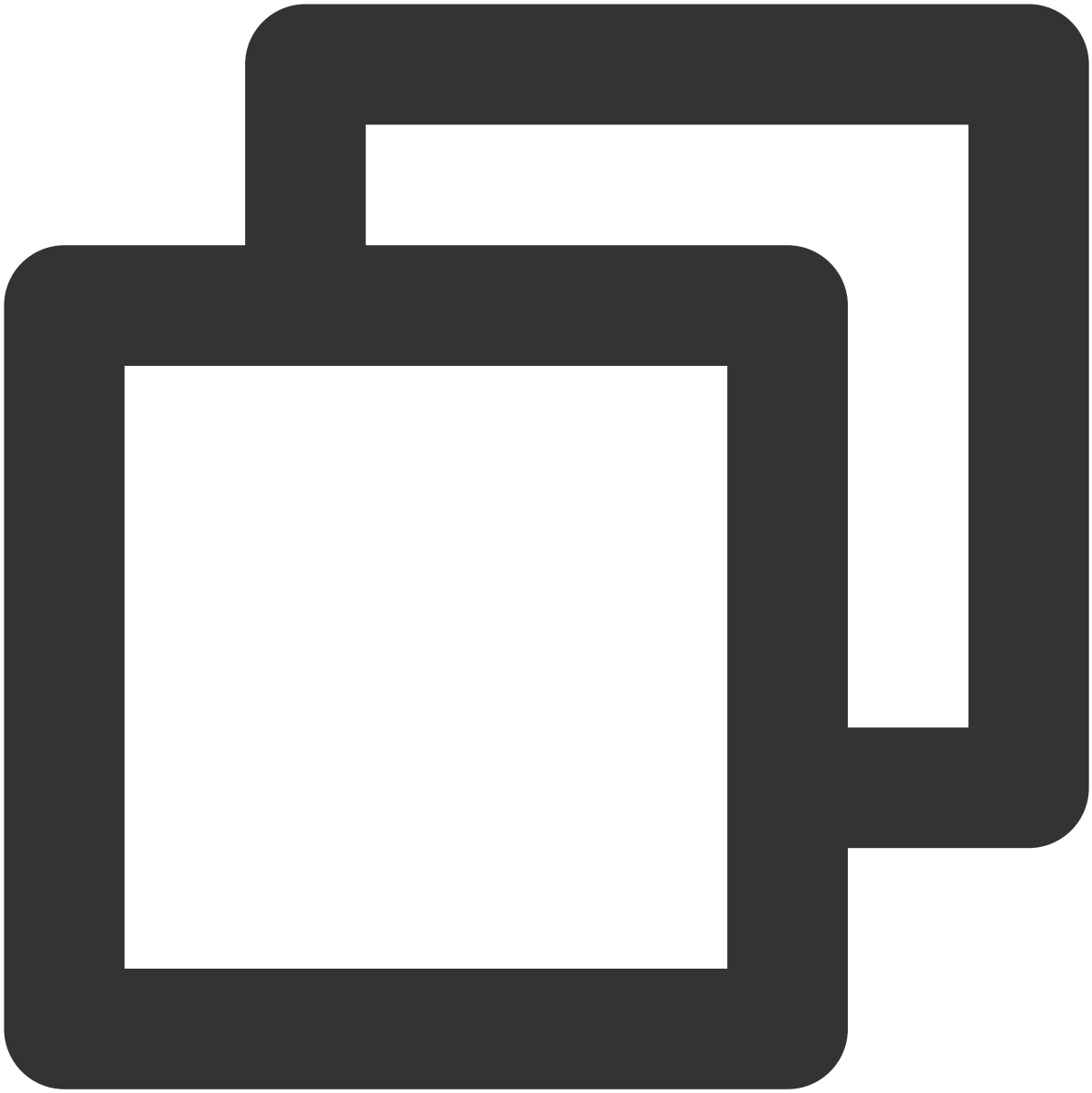
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置版本控制

功能说明

设置指定存储桶的版本控制功能。开启版本控制功能后，只能暂停，不能关闭。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
PutBucketVersioningRequest putBucketVersioningRequest =
        new PutBucketVersioningRequest(bucket);
//true：开启版本控制; false：暂停版本控制
putBucketVersioningRequest.setEnableVersion(true);

cosXmlService.putBucketVersionAsync(putBucketVersioningRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
```

```
            PutBucketVersioningResult putBucketVersioningResult =
                    (PutBucketVersioningResult) result;
        }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest cosXmlRequest,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

查询版本控制

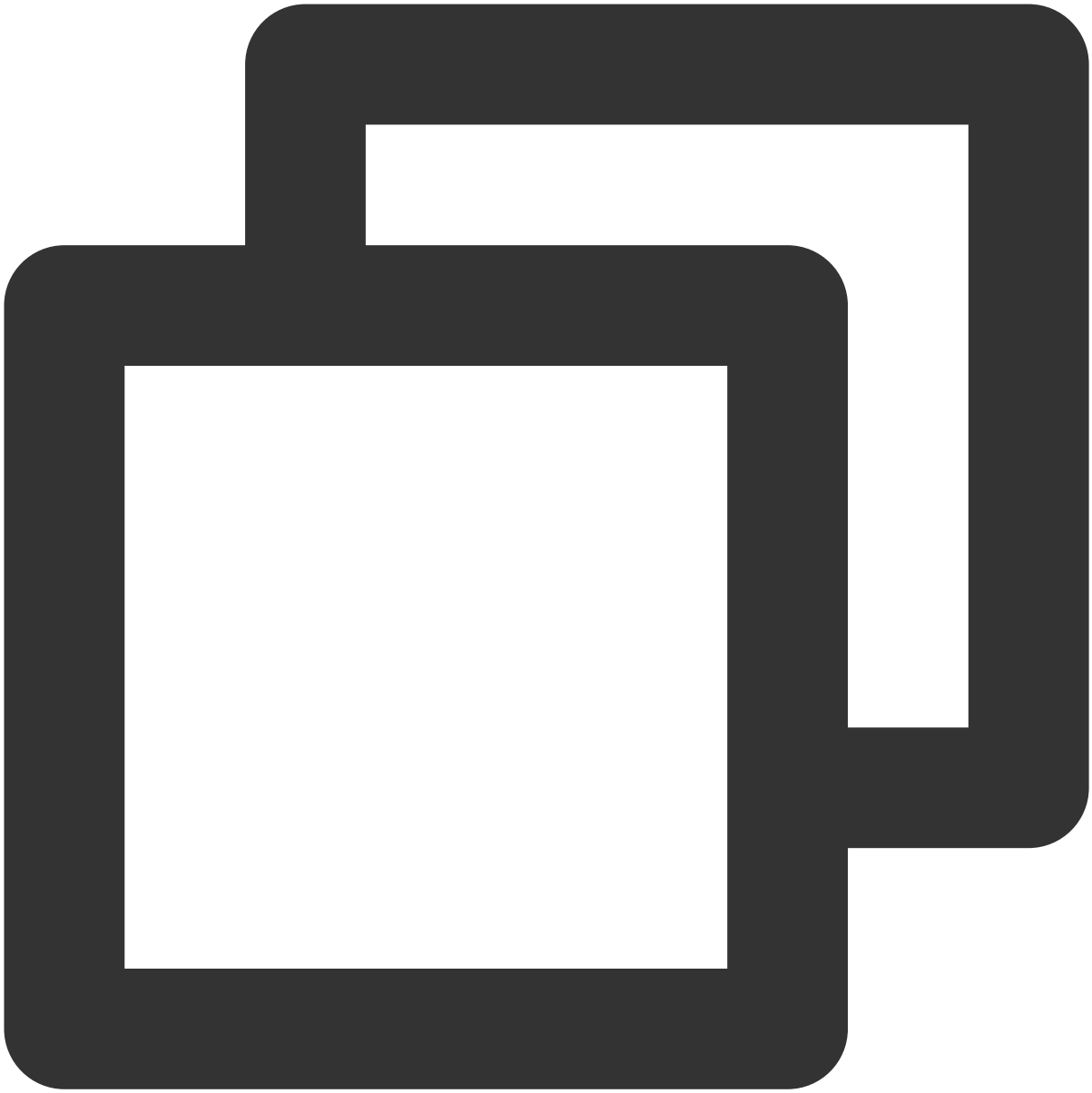
功能说明

查询指定存储桶的版本控制信息。

获取存储桶版本控制的状态，需要有该存储桶的读权限。

有三种版本控制状态：未启用版本控制、启用版本控制和暂停版本控制。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
GetBucketVersioningRequest getBucketVersioningRequest =
        new GetBucketVersioningRequest(bucket);

cosXmlService.getBucketVersioningAsync(getBucketVersioningRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketVersioningResult getBucketVersioningResult =
                (GetBucketVersioningResult) result;
```

```
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

跨地域复制

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于跨地域复制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

SDK API 参考

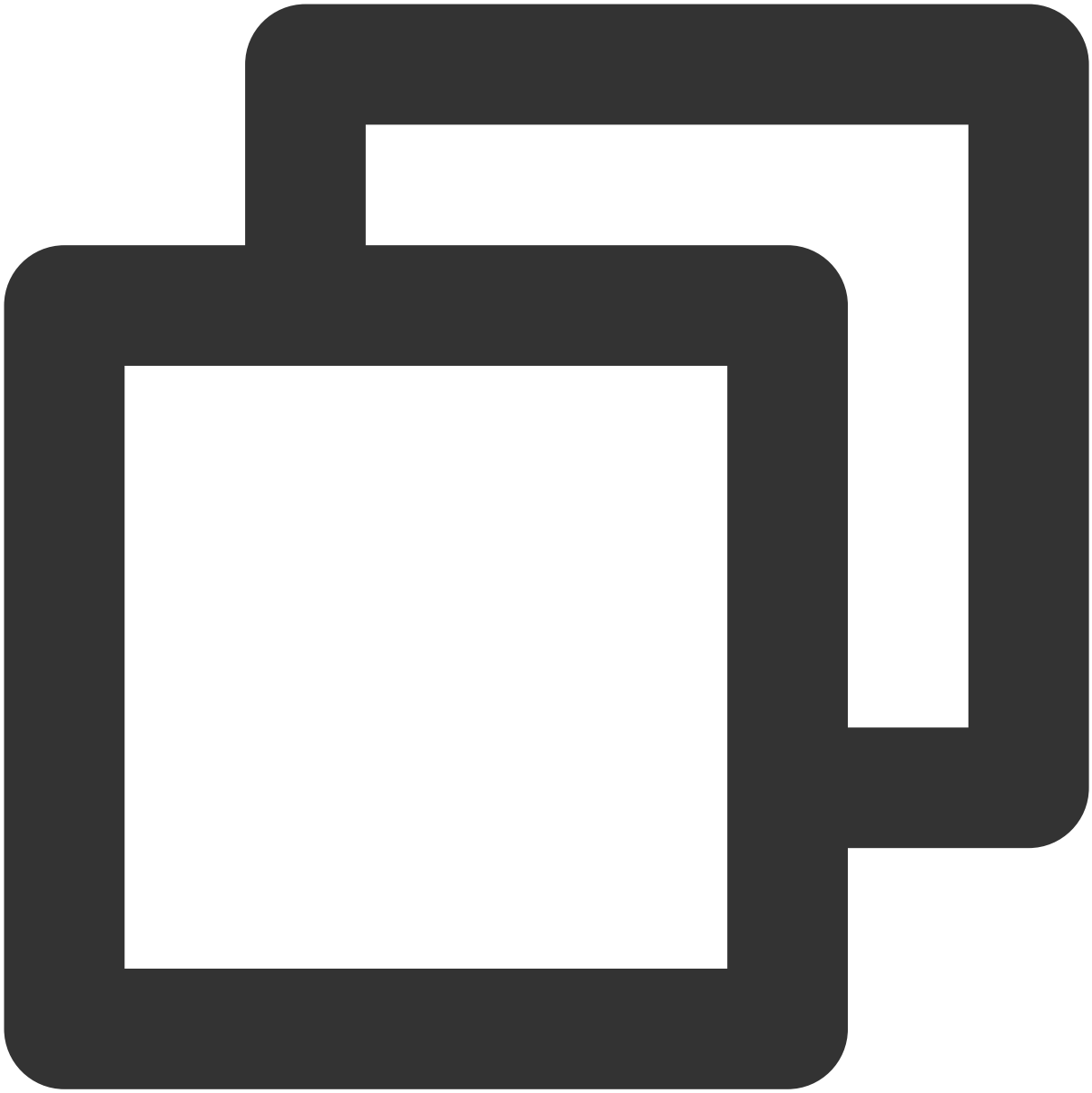
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置跨地域复制

功能说明

设置指定存储桶的跨地域复制规则。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
PutBucketReplicationRequest putBucketReplicationRequest =
        new PutBucketReplicationRequest(bucket);

String ownerUin = "100000000001"; //发起者身份标示：OwnerUin
String subUin = "100000000001"; //发起者身份标示：SubUin
putBucketReplicationRequest.setReplicationConfigurationWithRole(ownerUin,
        subUin);

PutBucketReplicationRequest.RuleStruct ruleStruct =
```

```
        new PutBucketReplicationRequest.RuleStruct();
//用来标注具体 Rule 的名称
ruleStruct.id = "replication_01";
//标识 Rule 是否生效。true：生效；false：不生效
ruleStruct.isEnable = true;
//目标存储桶地域信息
ruleStruct.region = "ap-beijing";
// 目标存储桶
ruleStruct.bucket = "destinationbucket-1250000000";
//前缀匹配策略
ruleStruct.prefix = "dir/";
putBucketReplicationRequest.setReplicationConfigurationWithRule(ruleStruct);

cosXmlService.putBucketReplicationAsync(putBucketReplicationRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutBucketReplicationResult putBucketReplicationResult =
                (PutBucketReplicationResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

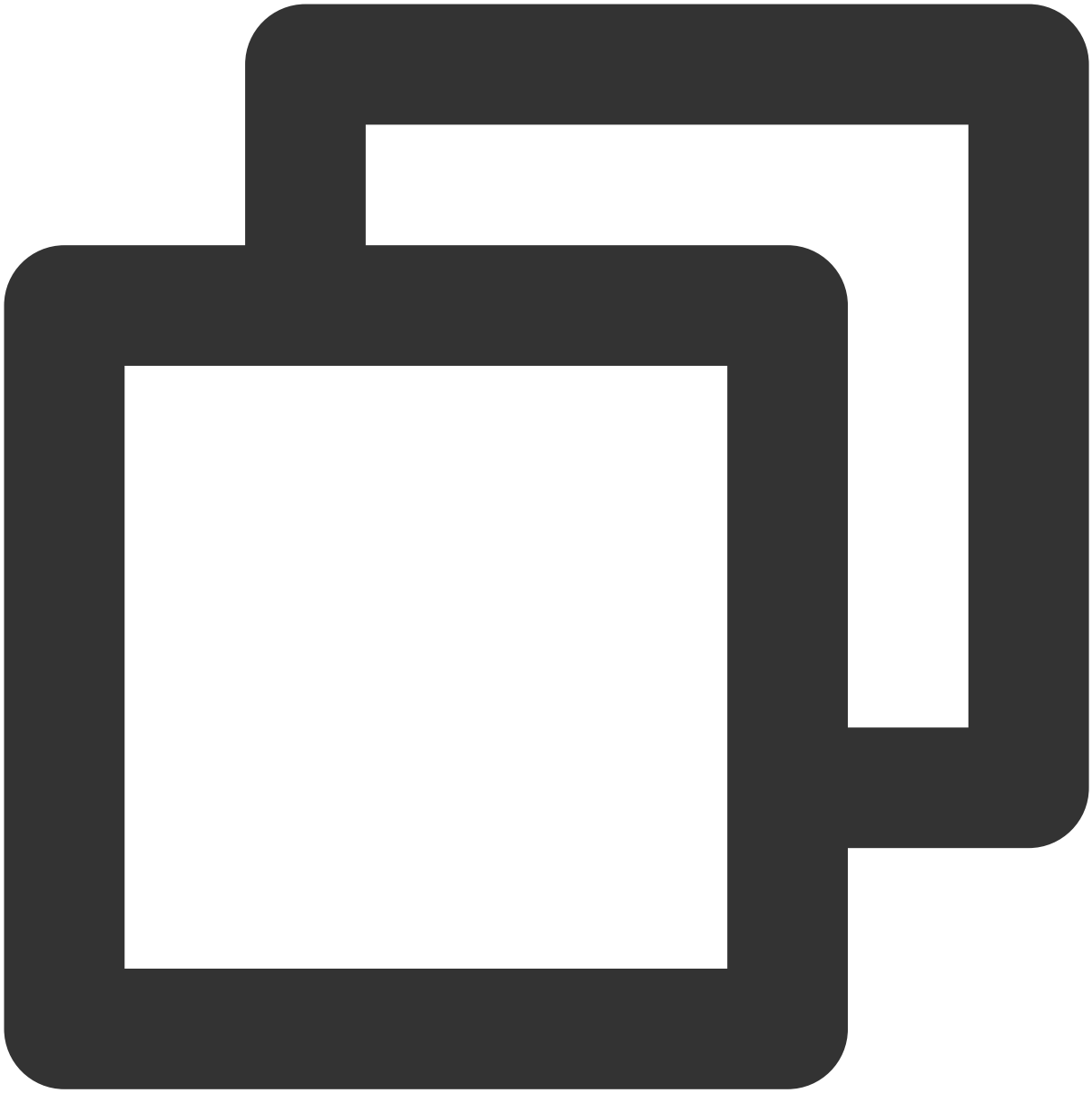
更多完整示例，请前往 GitHub 查看。

查询跨地域复制

功能说明

查询指定存储桶的跨地域复制规则。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
GetBucketReplicationRequest getBucketReplicationRequest =
        new GetBucketReplicationRequest(bucket);

cosXmlService.getBucketReplicationAsync(getBucketReplicationRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketReplicationResult getBucketReplicationResult =
                (GetBucketReplicationResult) result;
```

```
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

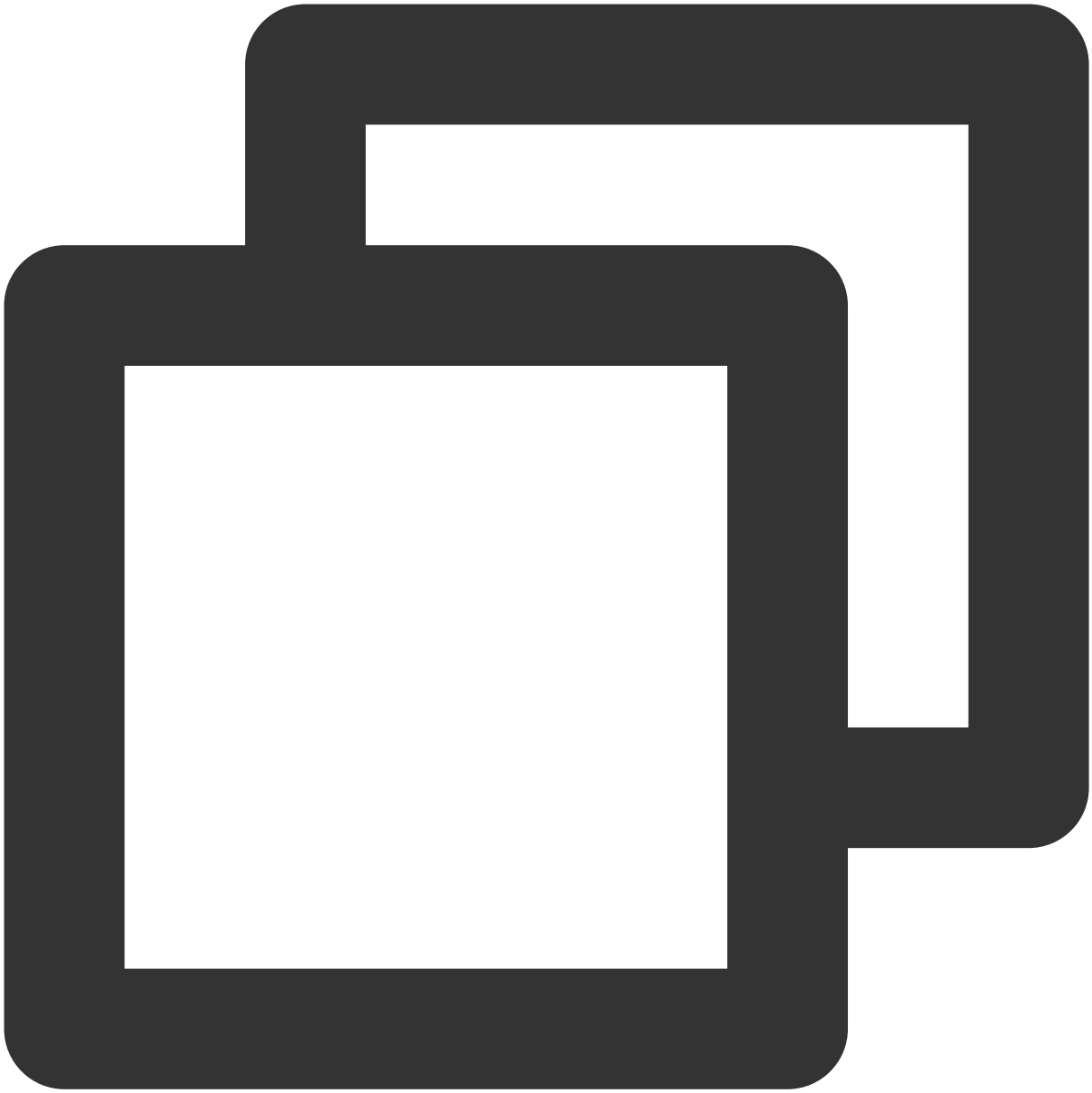
更多完整示例，请前往 GitHub 查看。

删除跨地域复制

功能说明

删除指定存储桶的跨地域复制规则。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
DeleteBucketReplicationRequest deleteBucketReplicationRequest =
        new DeleteBucketReplicationRequest(bucket);

cosXmlService.deleteBucketReplicationAsync(deleteBucketReplicationRequest,
        new CosXmlResultListener() {
            @Override
            public void onSuccess(CosXmlRequest request, CosXmlResult result) {
                DeleteBucketReplicationResult deleteBucketReplicationResult =
                        (DeleteBucketReplicationResult) result;
```

```
            }

            // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFa
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest cosXmlRequest,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
                    if (clientException != null) {
                        clientException.printStackTrace();
                    } else {
                        serviceException.printStackTrace();
                    }
                }
            });
```

说明

更多完整示例，请前往 GitHub 查看。

数据管理
生命周期

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于生命周期的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket lifecycle	设置生命周期	设置存储桶的生命周期管理的配置
GET Bucket lifecycle	查询生命周期	查询存储桶生命周期管理的配置
DELETE Bucket lifecycle	删除生命周期	删除存储桶生命周期管理的配置

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

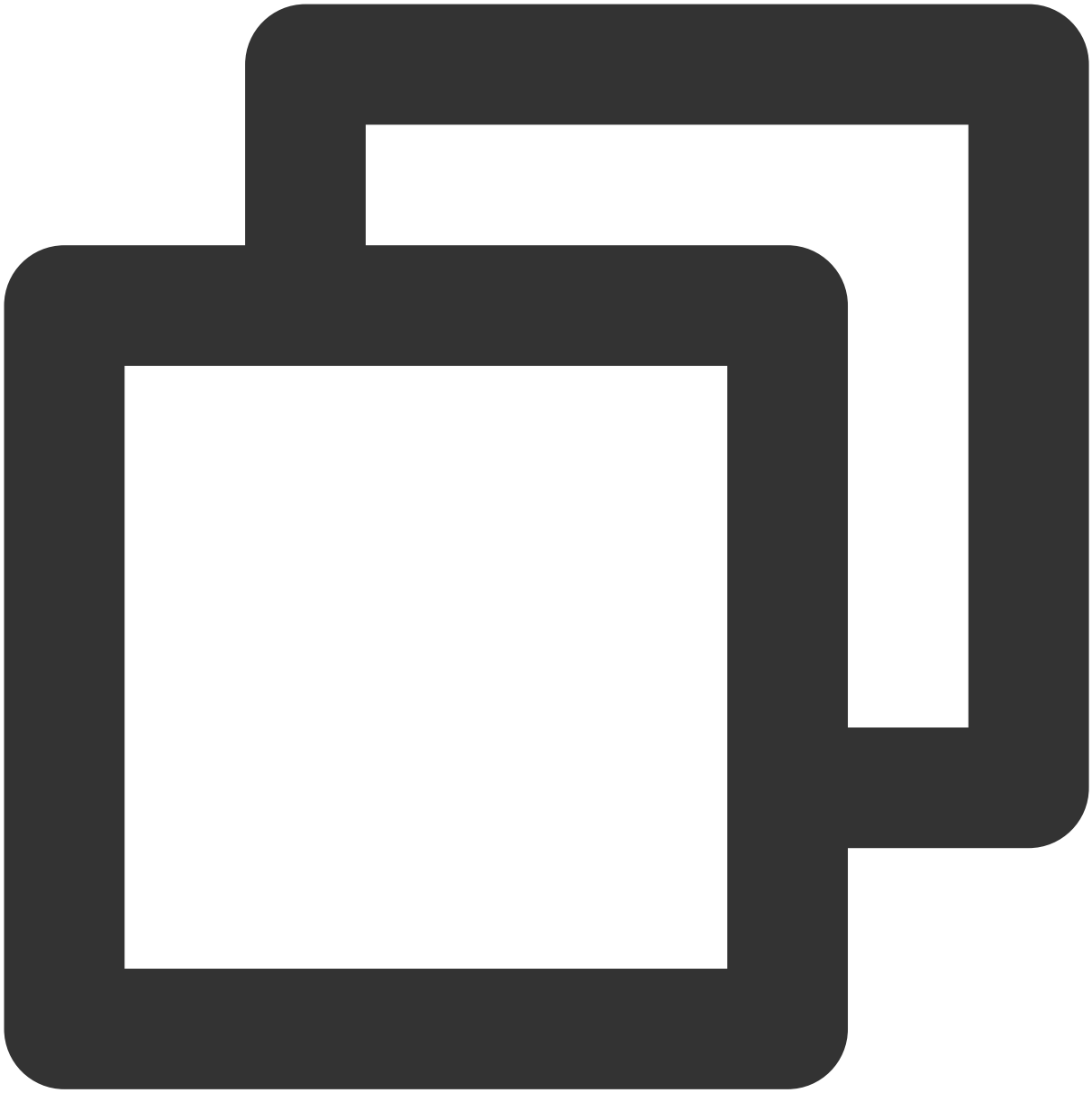
设置生命周期

功能说明

设置指定存储桶的生命周期配置信息。

示例代码

版权所有：腾讯云计算（北京）有限责任公司



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
PutBucketLifecycleRequest putBucketLifecycleRequest =
        new PutBucketLifecycleRequest(bucket);

// 声明周期配置规则信息
LifecycleConfiguration.Rule rule = new LifecycleConfiguration.Rule();
rule.id = "Lifecycle ID";
LifecycleConfiguration.Filter filter = new LifecycleConfiguration.Filter();
// 指定规则所适用的前缀
filter.prefix = "dir/";
```

```
rule.filter = filter;
// 指明规则是否启用
rule.status = "Enabled";
// 指明规则对应的动作在对象最后的修改日期过后多少天操作
LifecycleConfiguration.Transition transition =
        new LifecycleConfiguration.Transition();
transition.days = 100;
transition.storageClass = COSStorageClass.STANDARD.getStorageClass();
rule.transition = transition;

putBucketLifecycleRequest.setRuleList(rule);

cosXmlService.putBucketLifecycleAsync(putBucketLifecycleRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutBucketLifecycleResult putBucketLifecycleResult =
                (PutBucketLifecycleResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

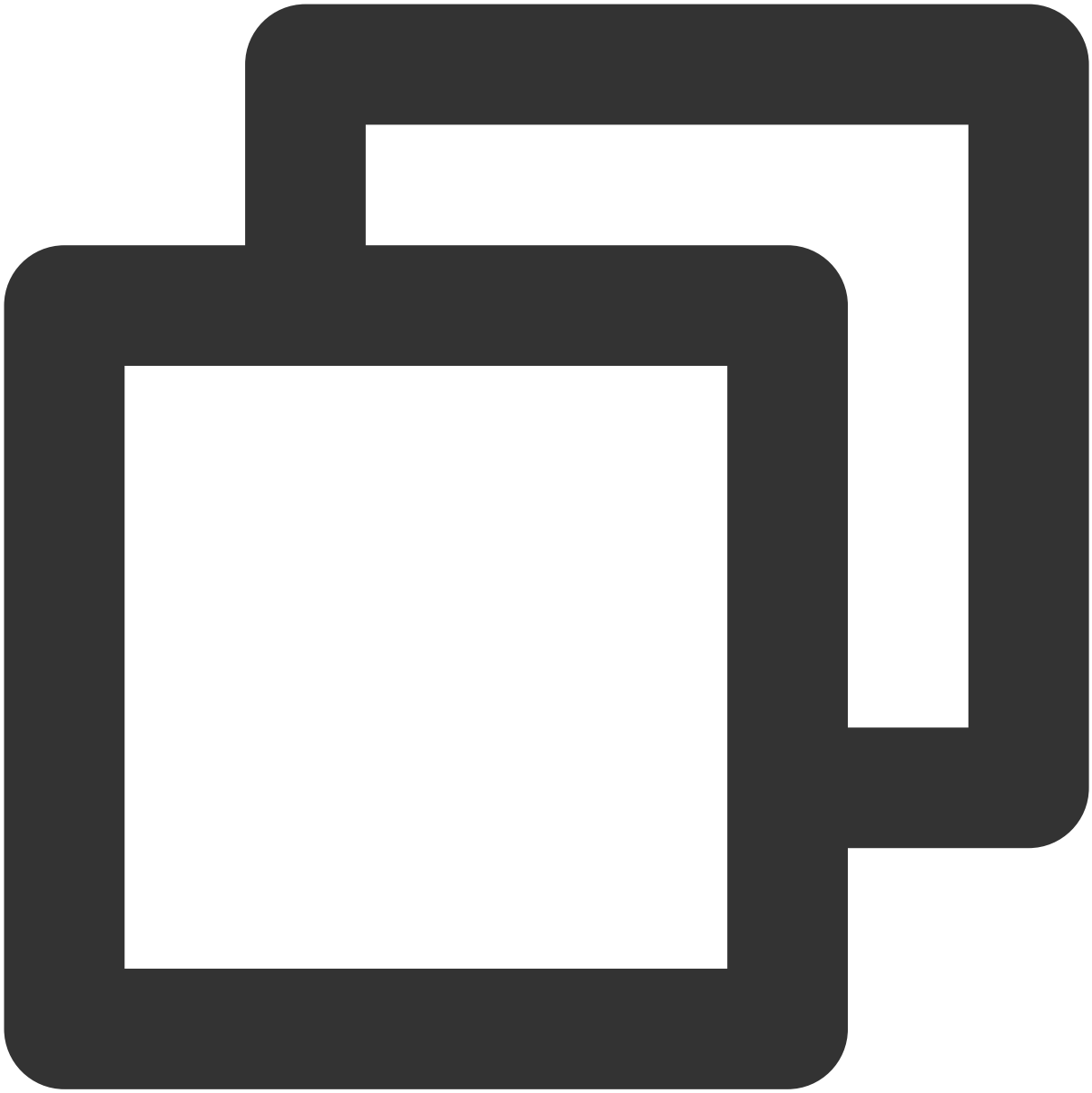
更多完整示例，请前往 GitHub 查看。

查询生命周期

功能说明

查询存储桶的生命周期管理配置。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
GetBucketLifecycleRequest getBucketLifecycleRequest =
        new GetBucketLifecycleRequest(bucket);

cosXmlService.getBucketLifecycleAsync(getBucketLifecycleRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketLifecycleResult getBucketLifecycleResult =
                (GetBucketLifecycleResult) result;
```

```
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

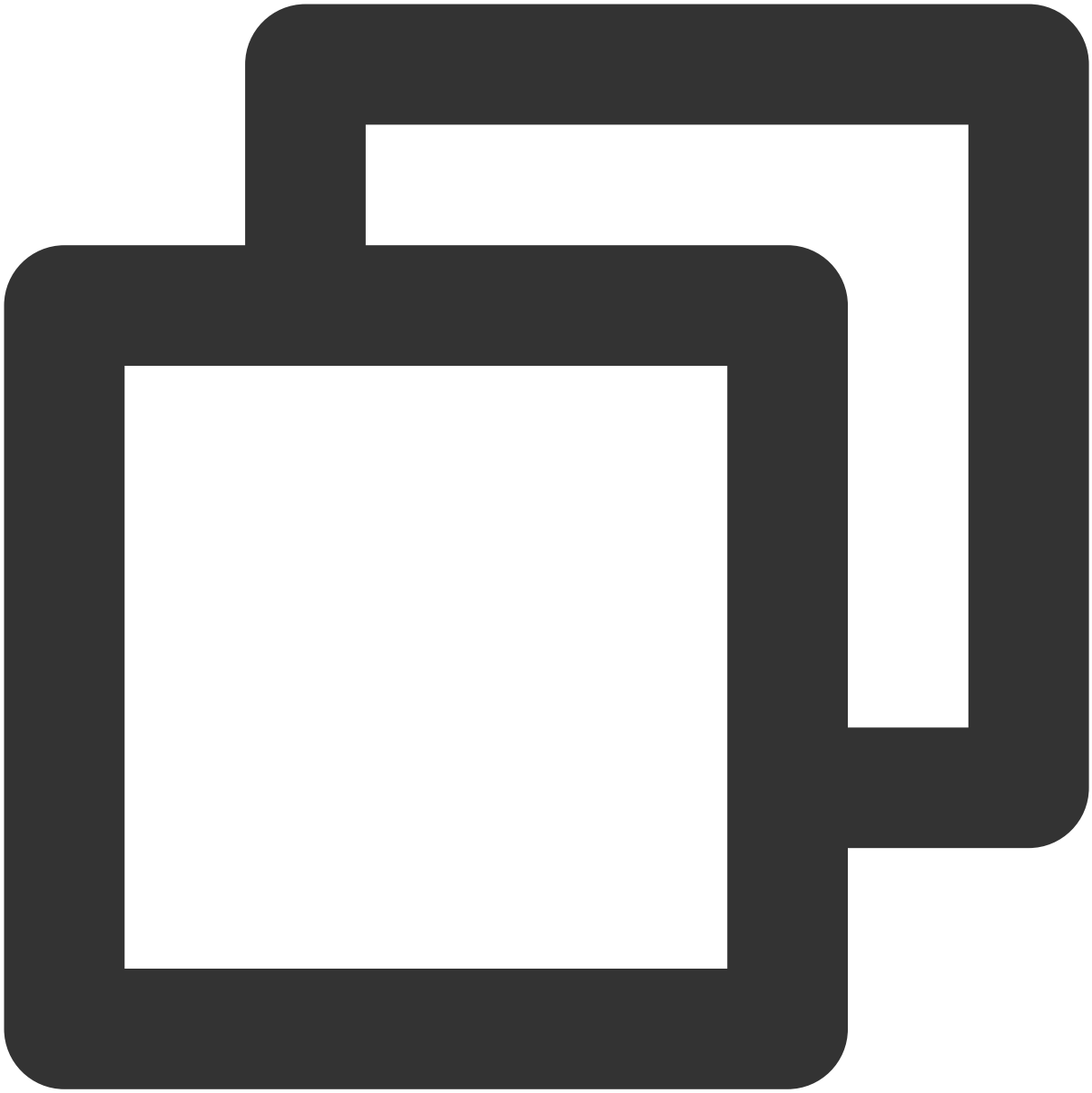
更多完整示例，请前往 GitHub 查看。

删除生命周期

功能说明

删除存储桶生命周期管理的配置。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
DeleteBucketLifecycleRequest deleteBucketLifecycleRequest =
        new DeleteBucketLifecycleRequest(bucket);

cosXmlService.deleteBucketLifecycleAsync(deleteBucketLifecycleRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        DeleteBucketLifecycleResult deleteBucketLifecycleResult =
                (DeleteBucketLifecycleResult) result;
```

```
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

日志管理

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于日志管理的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket logging	设置日志管理	为源存储桶开启日志记录
GET Bucket logging	查询日志管理	查询源存储桶的日志配置信息

SDK API 参考

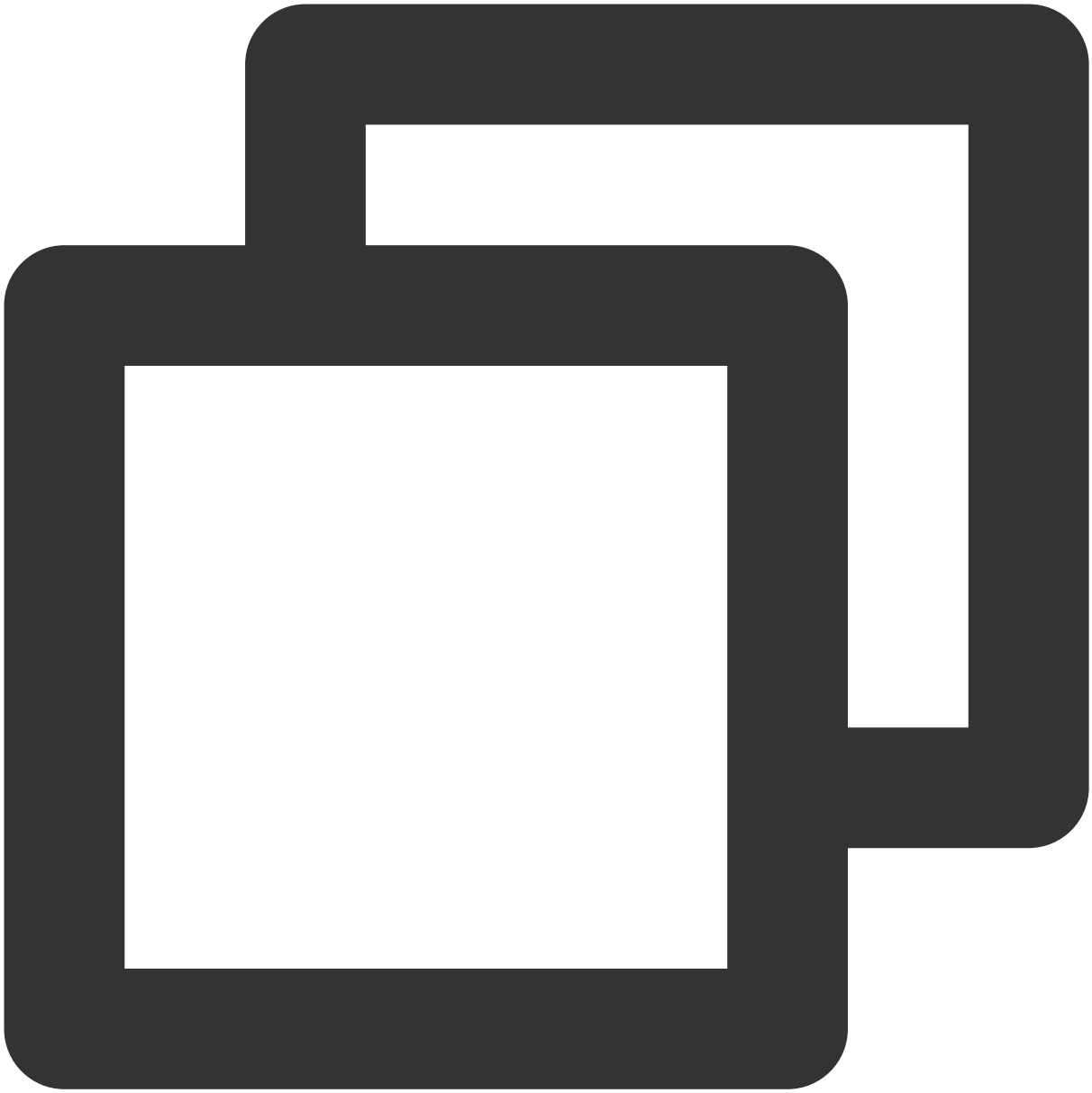
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置日志管理

功能说明

PUT Bucket logging 用于为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中。

示例代码



```
String srcBucket = "examplebucket-1250000000"; //格式：BucketName-APPID
String targetBucket = "examplebucket-1250000000"; //格式：BucketName-APPID
PutBucketLoggingRequest putBucketLoggingRequest =
        new PutBucketLoggingRequest(srcBucket);
// 目标存储桶
putBucketLoggingRequest.setTargetBucket(targetBucket);
// 日志存储的指定位置
putBucketLoggingRequest.setTargetPrefix("dir/");

cosXmlService.putBucketLoggingAsync(putBucketLoggingRequest,
        new CosXmlResultListener() {
```

```
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutBucketLoggingResult putBucketLoggingResult =
                (PutBucketLoggingResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

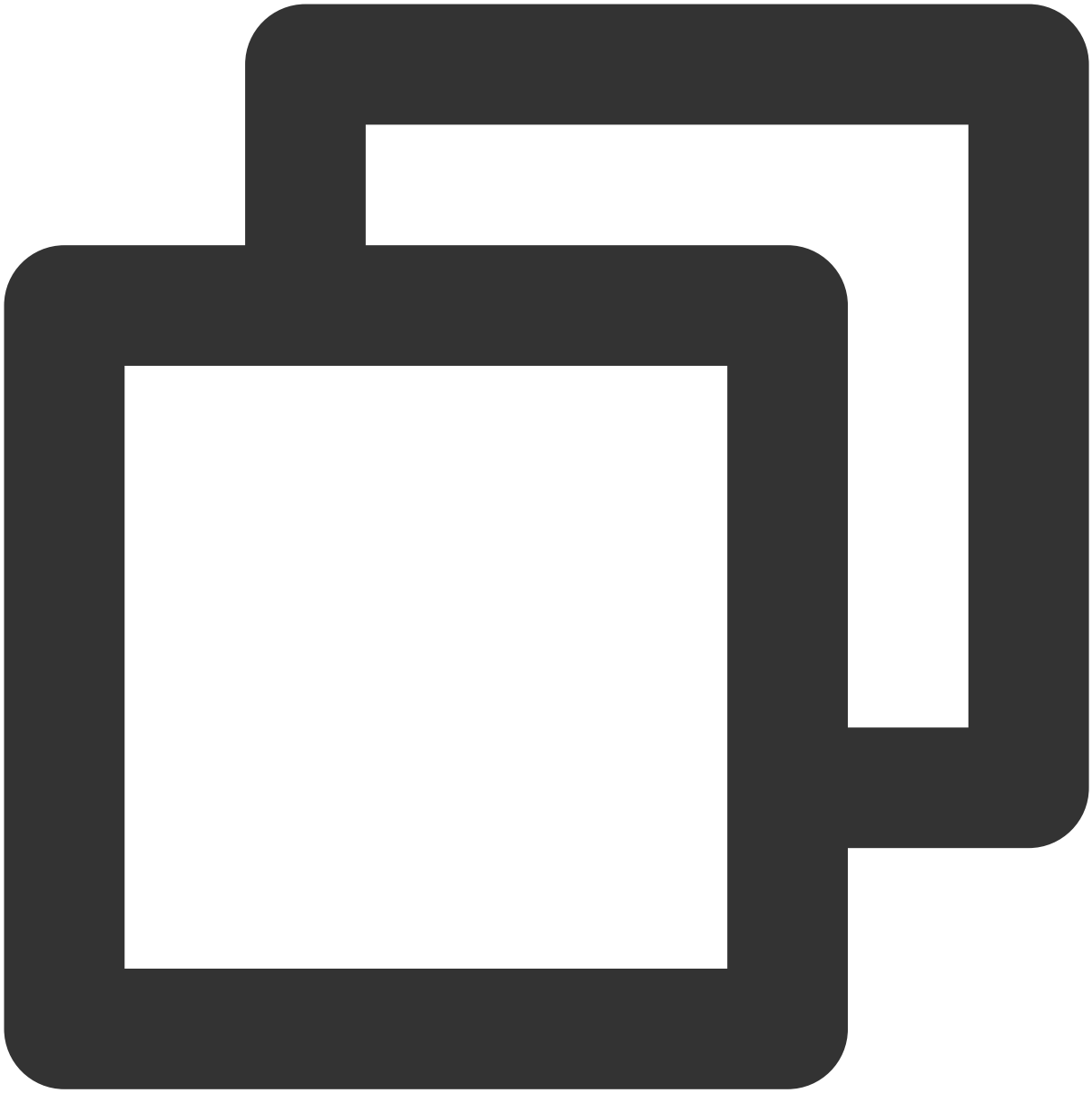
更多完整示例，请前往 GitHub 查看。

查询日志管理

功能说明

GET Bucket logging 用于查询指定存储桶的日志配置信息。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
GetBucketLoggingRequest getBucketLoggingRequest =
        new GetBucketLoggingRequest(bucket);

cosXmlService.getBucketLoggingAsync(getBucketLoggingRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketLoggingResult getBucketLoggingResult =
                (GetBucketLoggingResult) result;
```

```
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

对象标签

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于对象标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Object tagging	设置对象标签	为已上传的对象设置标签
GET Object tagging	查询对象标签	查询指定对象下已有的对象标签
DELETE Object tagging	删除对象标签	删除指定对象下已有的对象标签

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

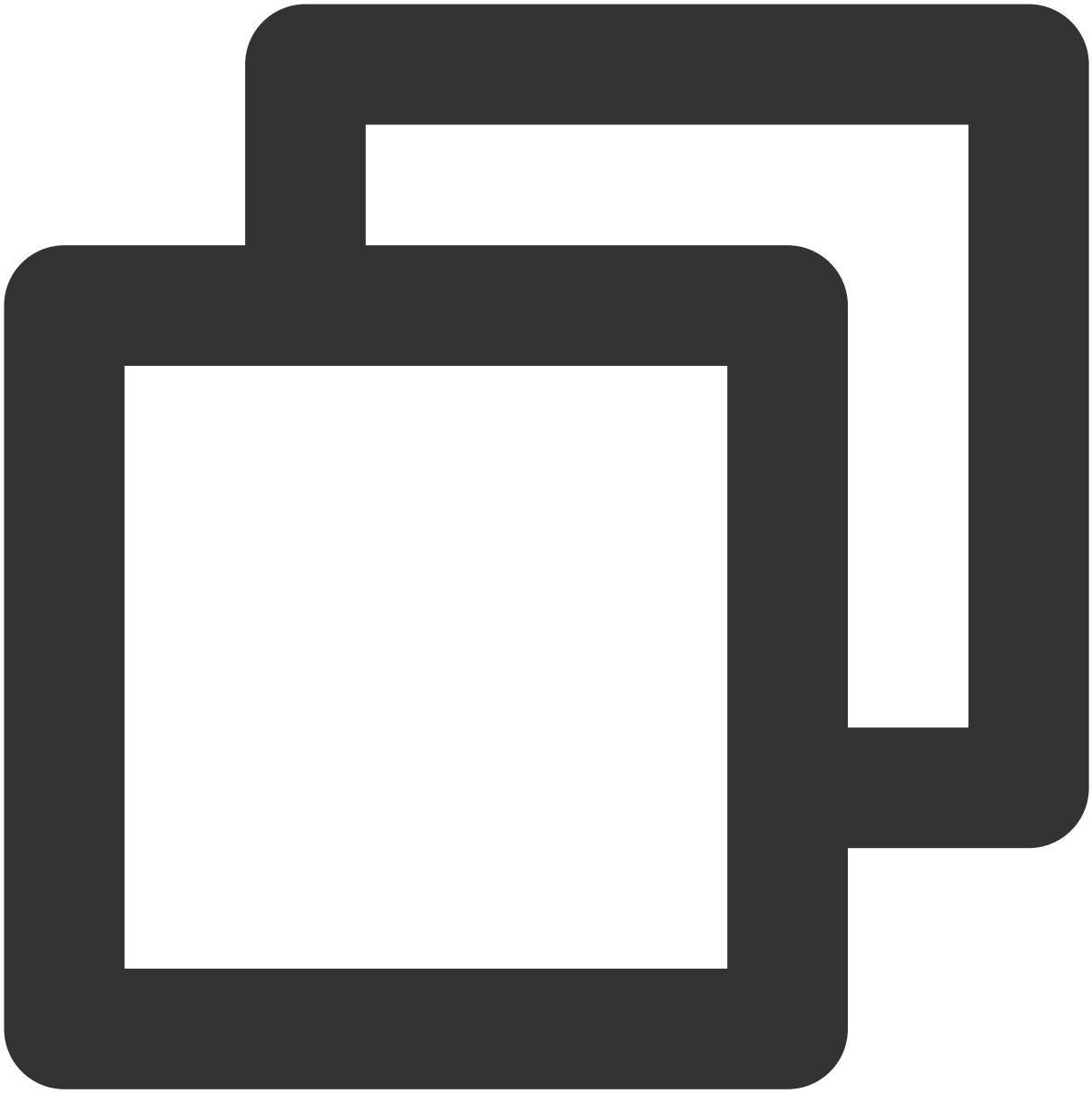
设置对象标签

上传时添加标签

功能说明

上传对象时，可以通过为请求添加特定的 Header 来给对象打标签，例如设置 x-cos-tagging 的值为 Key1=Value1&Key2=Value2，标签集合中的 Key 和 Value 必须先进行 URL 编码。

示例代码



```
// 初始化 TransferConfig，这里使用默认配置，如果需要定制，请参考 SDK 接口文档
TransferConfig transferConfig = new TransferConfig.Builder().build();
//初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String srcPath = new File(context.getCacheDir(), "exampleobject")
        .toString(); //本地文件的绝对路径
PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, srcPath);
```

```
try {
    // 设置对象标签，标签集合中的 Key 和 Value 必须先进行 URL 编码
    putObjectRequest.setRequestHeaders("x-cos-tagging", "Key1=Value&Key2=Value2", f
} catch (CosXmlClientException e) {
    e.printStackTrace();
}
// 若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传；否则，赋值 null
String uploadId = null;
// 上传文件
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(bucket, cosPath,
        srcPath, uploadId);
//设置返回结果回调
cosxmlUploadTask.setCosXmlResultListener(new CosXmlResultListener() {

    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        COSXMLUploadTask.COSXMLUploadTaskResult uploadResult =
                (COSXMLUploadTask.COSXMLUploadTaskResult) result;
    }
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

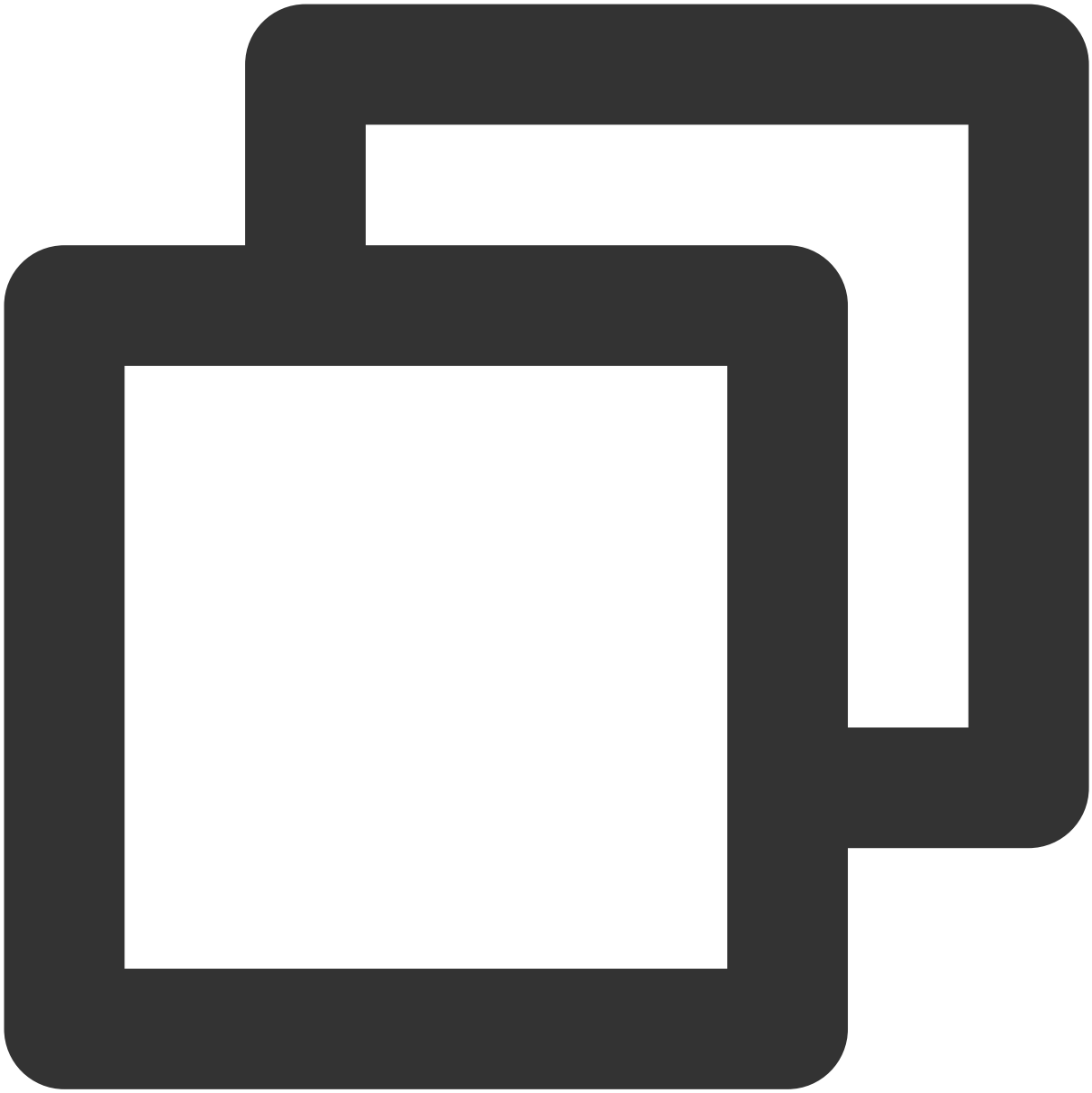
更多完整示例，请前往 GitHub 查看。

为已存在的对象添加标签

功能说明

COS 支持为已存在的对象设置标签。通过为对象添加键值对作为对象标签，可以协助您分组管理已有的对象资源。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
PutObjectTaggingRequest putObjectTaggingRequest = new PutObjectTaggingRequest(bucke
putObjectTaggingRequest.addTag("key", "value");
try {
    PutObjectTaggingResult putObjectTaggingResult = cosXmlService.putObjectTagging(
} catch (CosXmlClientException clientException) {
    clientException.printStackTrace();
} catch (CosXmlServiceException serviceException) {
    serviceException.printStackTrace();
```

```
}
```

说明

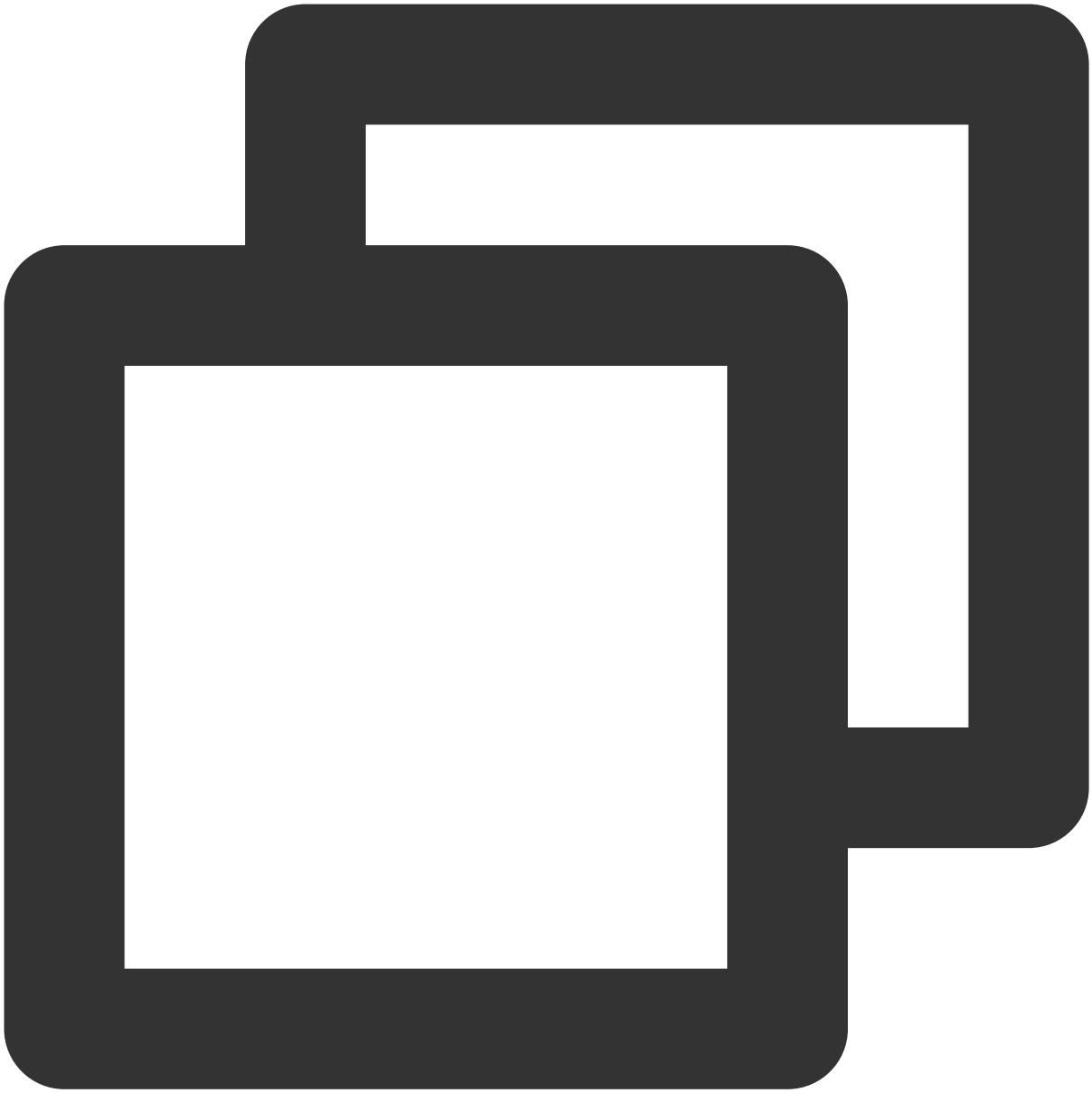
更多完整示例，请前往 GitHub 查看。

查询对象标签

功能说明

查询指定对象下已有的对象标签。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
GetObjectTaggingRequest getObjectTaggingRequest = new GetObjectTaggingRequest(bucke
try {
    GetObjectTaggingResult getObjectTaggingResult = cosXmlService.getObjectTagging(
} catch (CosXmlClientException clientException) {
    clientException.printStackTrace();
} catch (CosXmlServiceException serviceException) {
    serviceException.printStackTrace();
}
```

说明

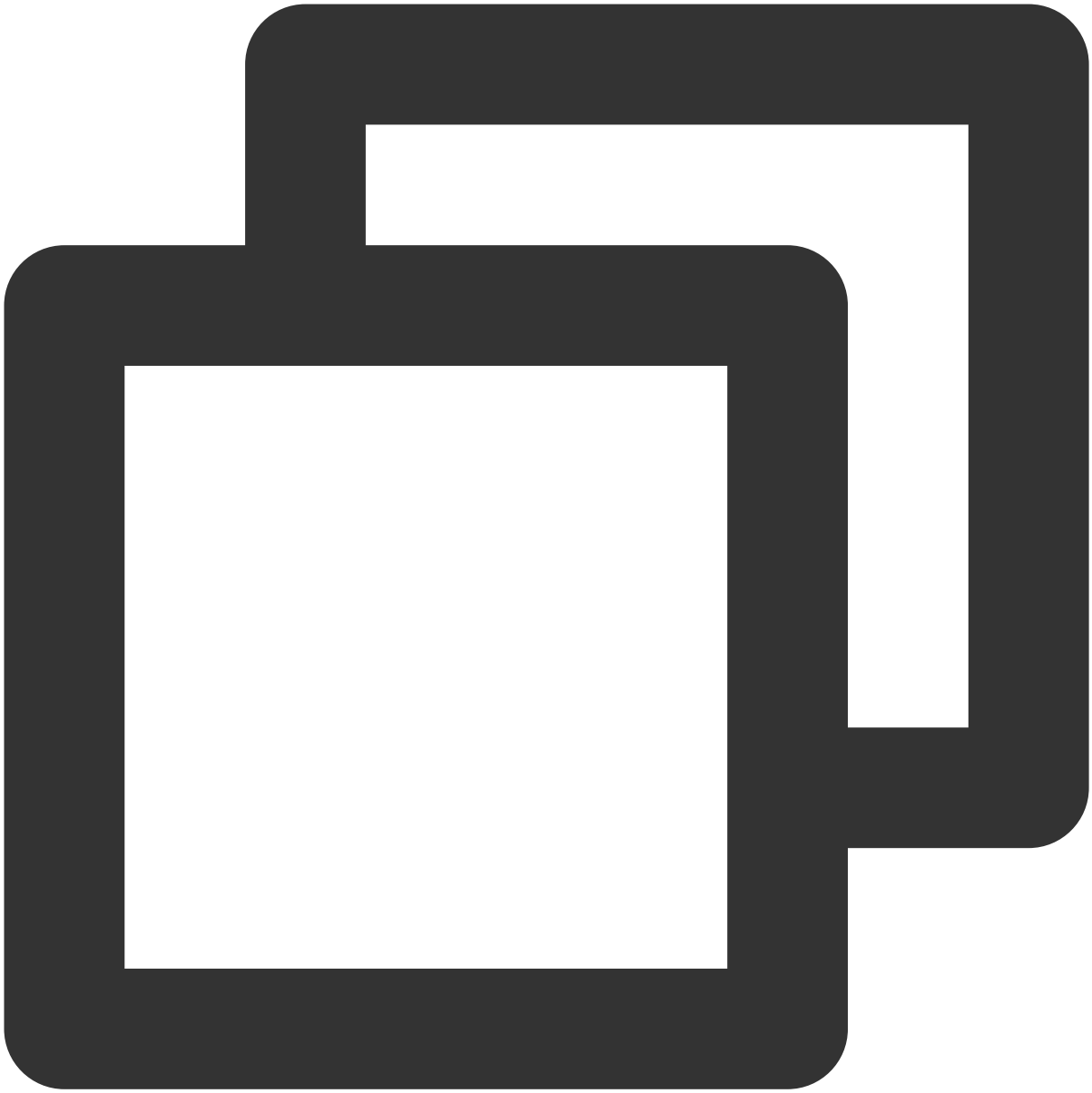
更多完整示例，请前往 GitHub 查看。

删除对象标签

功能说明

删除指定对象下已有的对象标签。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
DeleteObjectTaggingRequest deleteObjectTaggingRequest = new DeleteObjectTaggingRequ
try {
    DeleteObjectTaggingResult deleteObjectTaggingResult = cosXmlService.deleteObjec
} catch (CosXmlClientException clientException) {
    clientException.printStackTrace();
} catch (CosXmlServiceException serviceException) {
    serviceException.printStackTrace();
}
```

说明

更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

存储桶标签

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket tagging	设置存储桶标签	为已存在的存储桶设置标签
GET Bucket tagging	查询存储桶标签	查询指定存储桶下已有的存储桶标签
DELETE Bucket tagging	删除存储桶标签	删除指定的存储桶标签

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

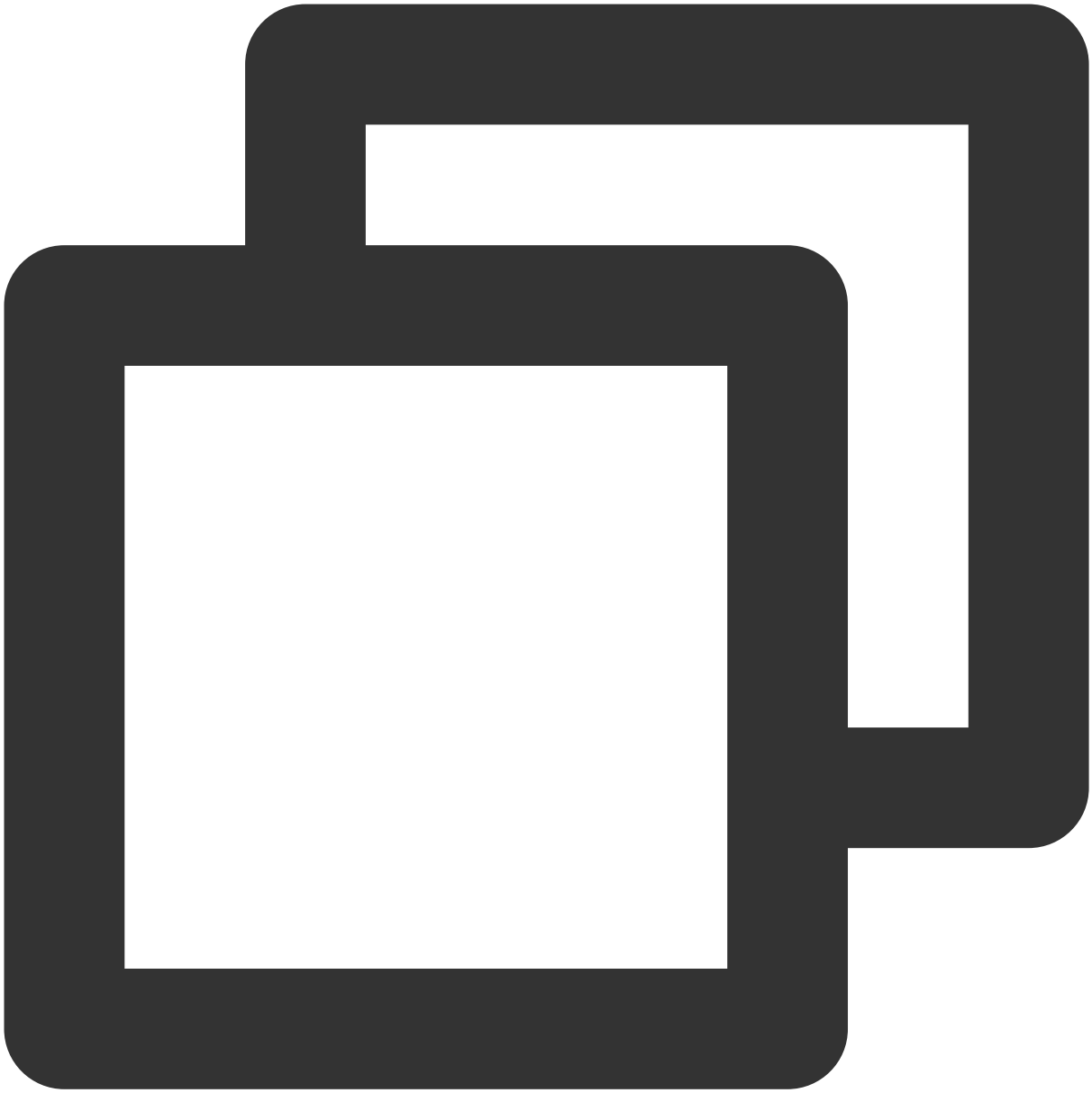
设置存储桶标签

功能说明

PUT Bucket tagging 用于为已存在的存储桶设置标签。

示例代码

版权所有：腾讯云计算（北京）有限责任公司



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
PutBucketTaggingRequest putBucketTaggingRequest =
        new PutBucketTaggingRequest(bucket);
// 设置标签
putBucketTaggingRequest.addTag("key", "value");
putBucketTaggingRequest.addTag("hello", "world");

cosXmlService.putBucketTaggingAsync(putBucketTaggingRequest,
        new CosXmlResultListener() {
    @Override
```

```
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutBucketTaggingResult putBucketTaggingResult =
                (PutBucketTaggingResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

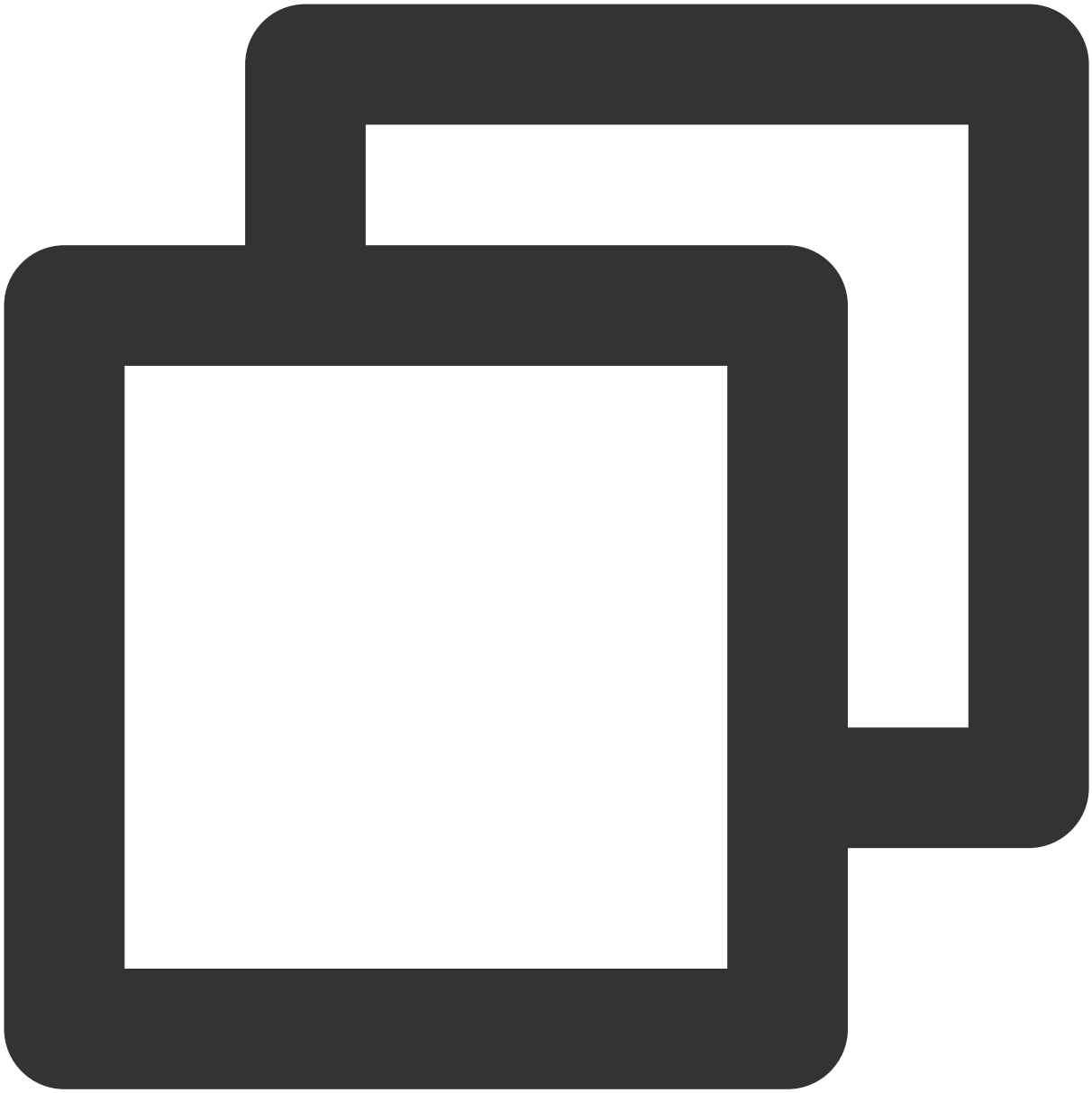
更多完整示例，请前往 GitHub 查看。

查询存储桶标签

功能说明

GET Bucket tagging 用于查询指定存储桶下已有的存储桶标签。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
GetBucketTaggingRequest getBucketTaggingRequest =
        new GetBucketTaggingRequest(bucket);

cosXmlService.getBucketTaggingAsync(getBucketTaggingRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketTaggingResult getBucketTaggingResult =
                (GetBucketTaggingResult) result;
```

```
        }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest cosXmlRequest,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

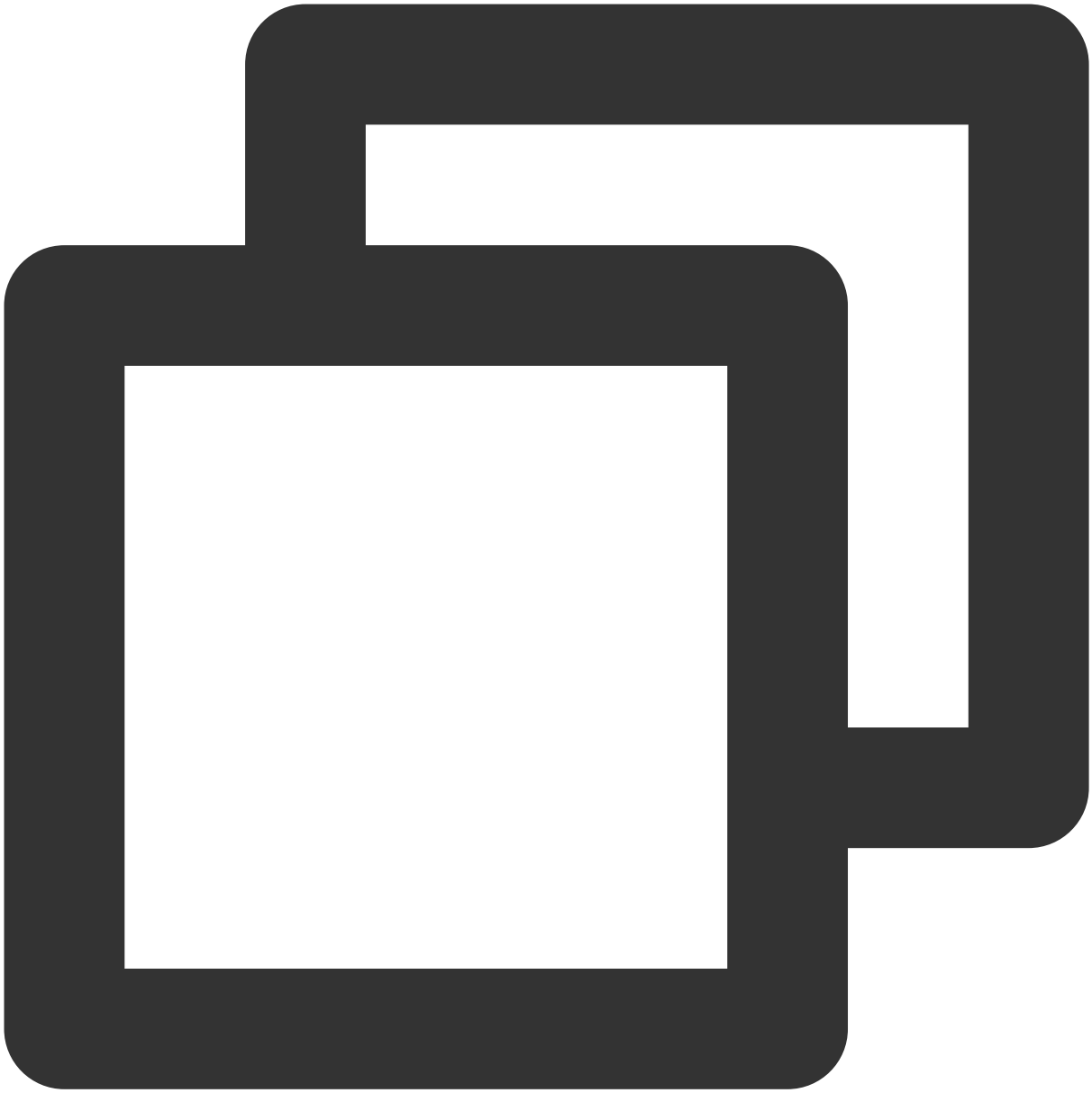
更多完整示例，请前往 GitHub 查看。

删除存储桶标签

功能说明

DELETE Bucket tagging 用于删除指定存储桶下已有的存储桶标签。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
DeleteBucketTaggingRequest deleteBucketTaggingRequest =
        new DeleteBucketTaggingRequest(bucket);

cosXmlService.deleteBucketTaggingAsync(deleteBucketTaggingRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        DeleteBucketTaggingResult getBucketTaggingResult =
                (DeleteBucketTaggingResult) result;
```

```
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }

    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

静态网站

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket website	设置静态网站	设置存储桶的静态网站配置
GET Bucket website	查询静态网站配置	查询存储桶的静态网站配置
DELETE Bucket website	删除静态网站配置	删除存储桶的静态网站配置

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

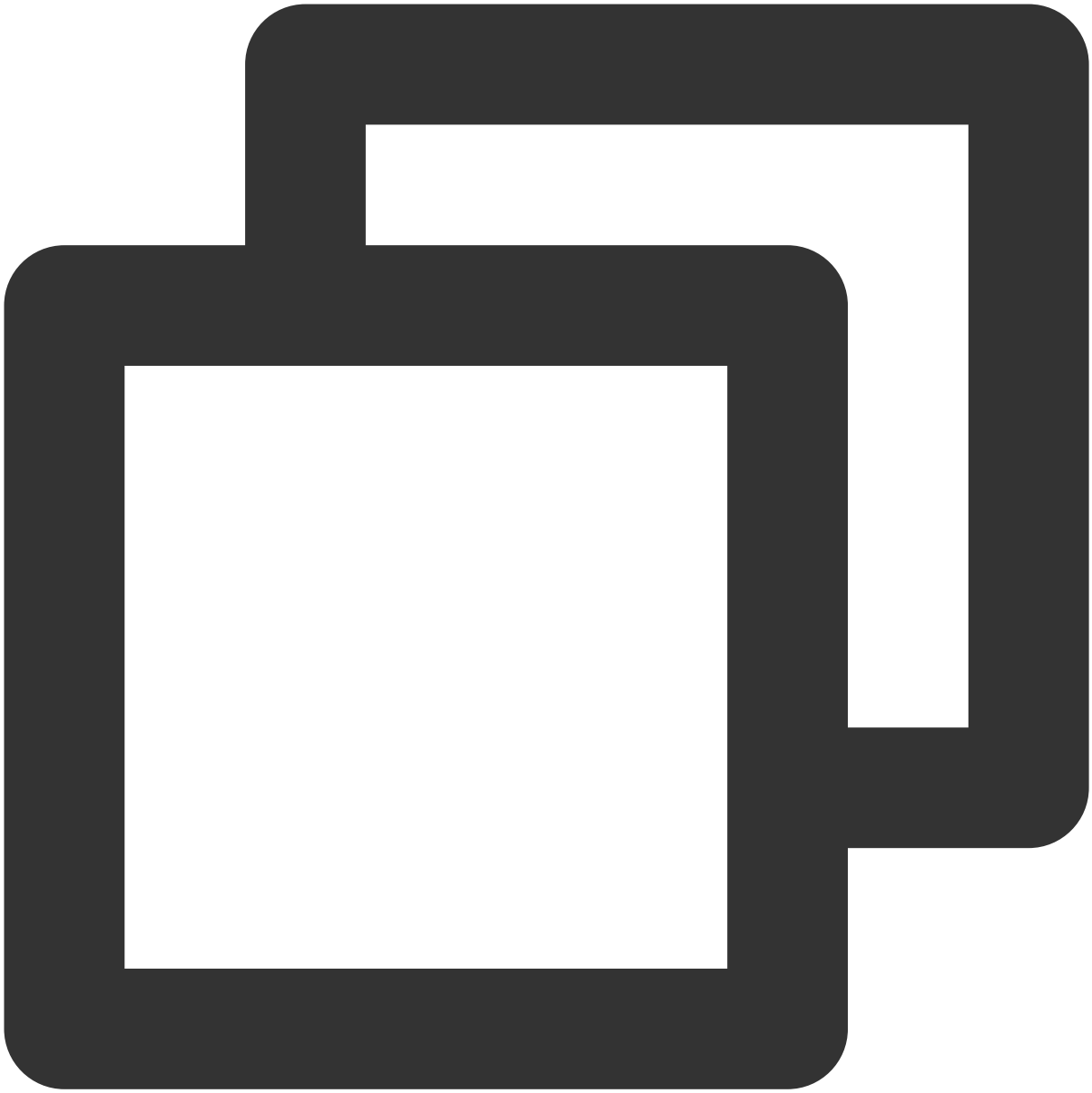
设置静态网站

功能说明

PUT Bucket website 用于为存储桶配置静态网站。

示例代码

版权所有：腾讯云计算（北京）有限责任公司



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
PutBucketWebsiteRequest putBucketWebsiteRequest =
        new PutBucketWebsiteRequest(bucket);
// 设置 index 文档
putBucketWebsiteRequest.setIndexDocument("index.html");

cosXmlService.putBucketWebsiteAsync(putBucketWebsiteRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
```

```
            PutBucketWebsiteResult putBucketWebsiteResult =
                    (PutBucketWebsiteResult) result;
        }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest cosXmlRequest,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

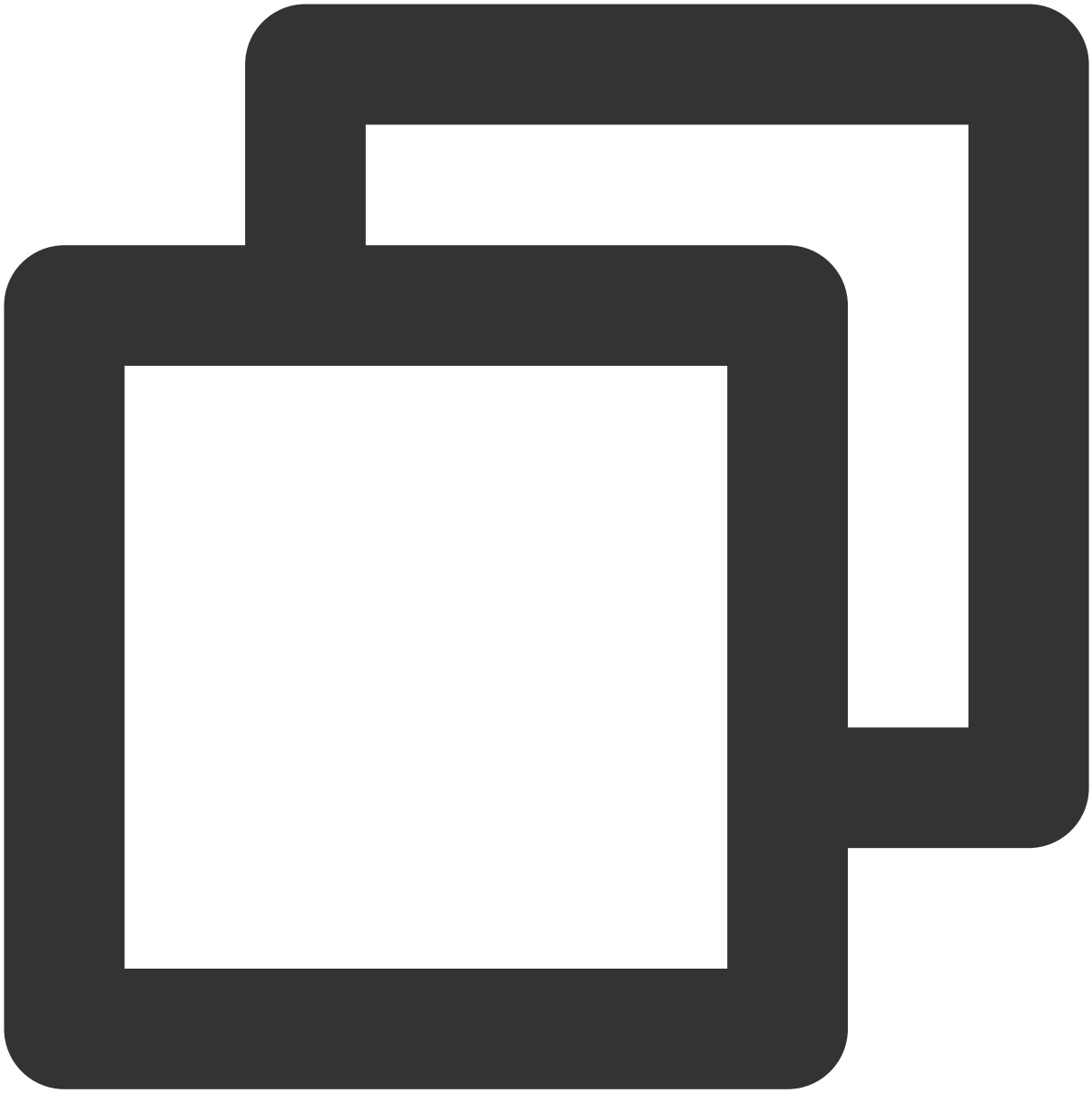
更多完整示例，请前往 GitHub 查看。

查询静态网站配置

功能说明

GET Bucket website 用于查询与存储桶关联的静态网站配置信息。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
GetBucketWebsiteRequest getBucketWebsiteRequest =
        new GetBucketWebsiteRequest(bucket);
cosXmlService.getBucketWebsiteAsync(getBucketWebsiteRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketWebsiteResult getBucketWebsiteResult =
                (GetBucketWebsiteResult) result;
    }
```

```
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

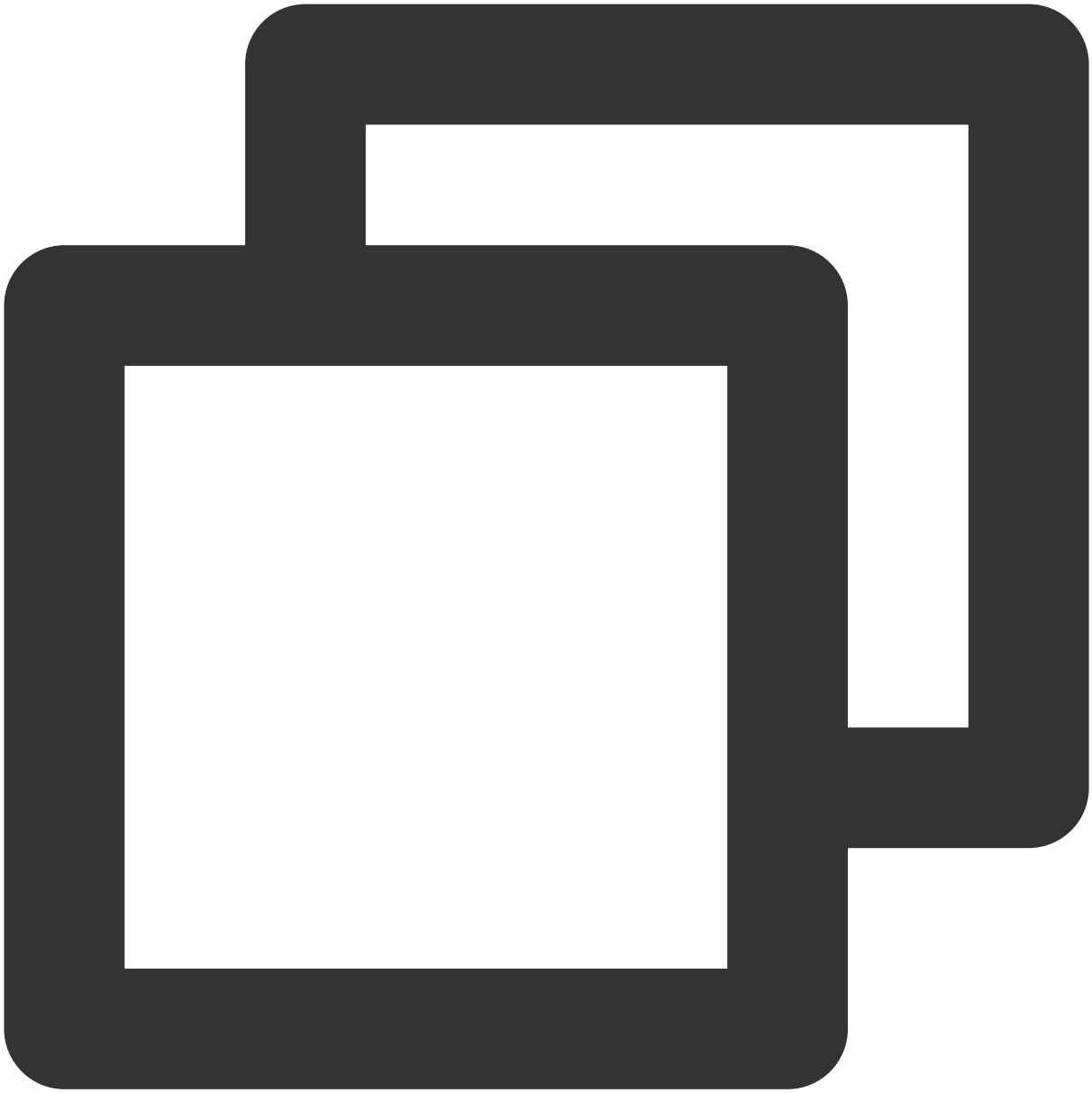
更多完整示例，请前往 GitHub 查看。

删除静态网站配置

功能说明

DELETE Bucket website 用于删除存储桶中的静态网站配置。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
DeleteBucketWebsiteRequest deleteBucketWebsiteRequest =
        new DeleteBucketWebsiteRequest(bucket);

cosXmlService.deleteBucketWebsiteAsync(deleteBucketWebsiteRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        DeleteBucketWebsiteResult getBucketWebsiteResult =
                (DeleteBucketWebsiteResult) result;
```

```
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

清单

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于清单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket inventory	设置清单任务	设置存储桶的清单任务
GET Bucket inventory	查询清单任务	查询存储桶的清单任务
DELETE Bucket inventory	删除清单任务	删除存储桶的清单任务

SDK API 参考

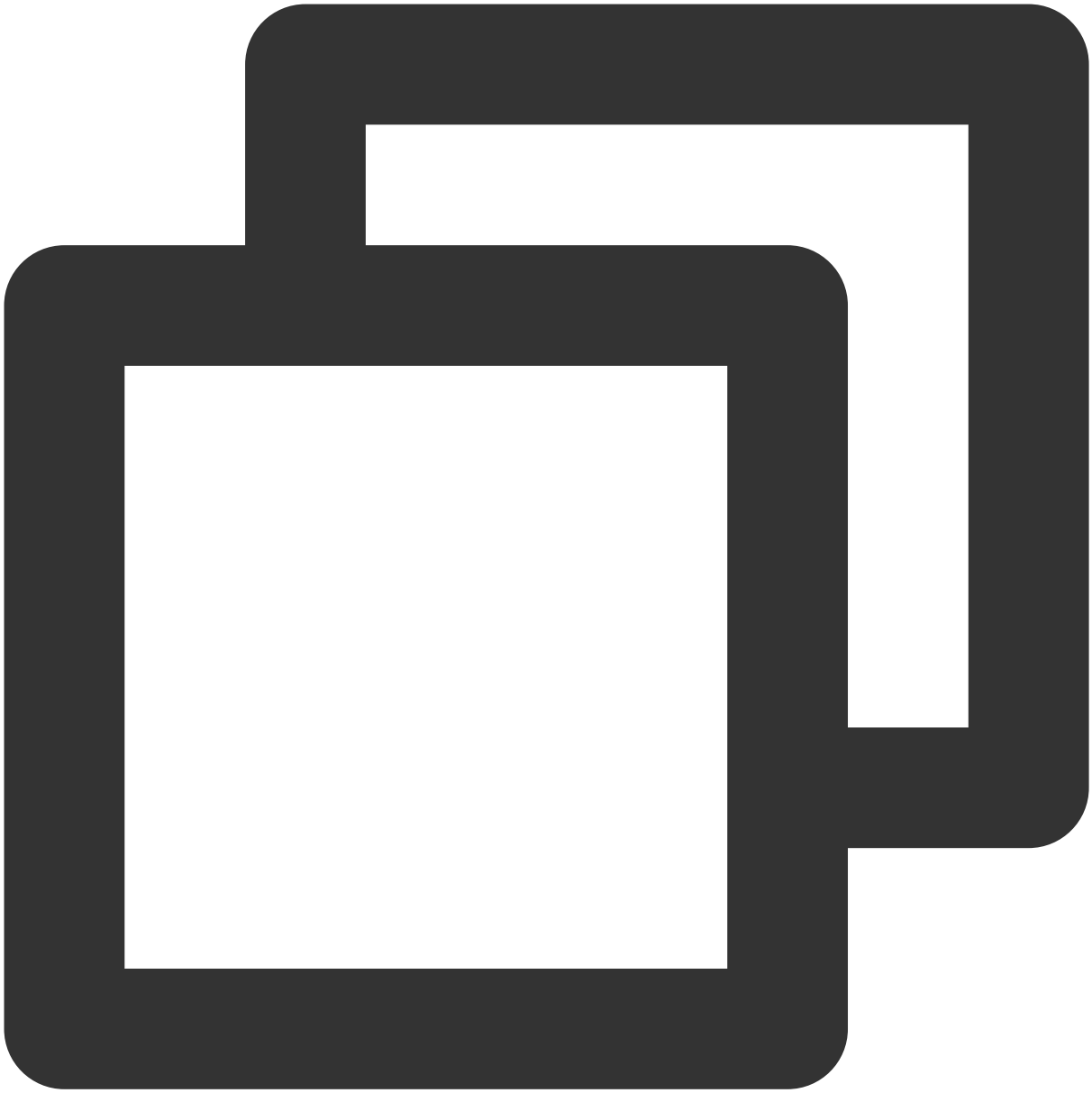
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置清单任务

功能说明

PUT Bucket inventory 用于在存储桶中创建清单任务。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
PutBucketInventoryRequest putBucketInventoryRequest =
        new PutBucketInventoryRequest(bucket);
putBucketInventoryRequest.setInventoryId("exampleInventoryId");
// 是否在清单中包含对象版本：
// 如果设置为 All，清单中将会包含所有对象版本，
// 并在清单中增加VersionId，IsLatest，DeleteMarker 这几个字段
// 如果设置为 Current，则清单中不包含对象版本信息
putBucketInventoryRequest.setIncludedObjectVersions(InventoryConfiguration
        .IncludedObjectVersions.ALL);
```

```
// 备份频率
putBucketInventoryRequest.setScheduleFrequency(InventoryConfiguration
        .SCHEDULE_FREQUENCY_DAILY);
// 备份路径
putBucketInventoryRequest.setDestination("CSV", "1000000000",
        "examplebucket-1250000000", "region", "dir/");

cosXmlService.putBucketInventoryAsync(putBucketInventoryRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutBucketInventoryResult putBucketInventoryResult =
                (PutBucketInventoryResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

错误码说明

该请求可能会发生的一些常见的特殊错误如下：

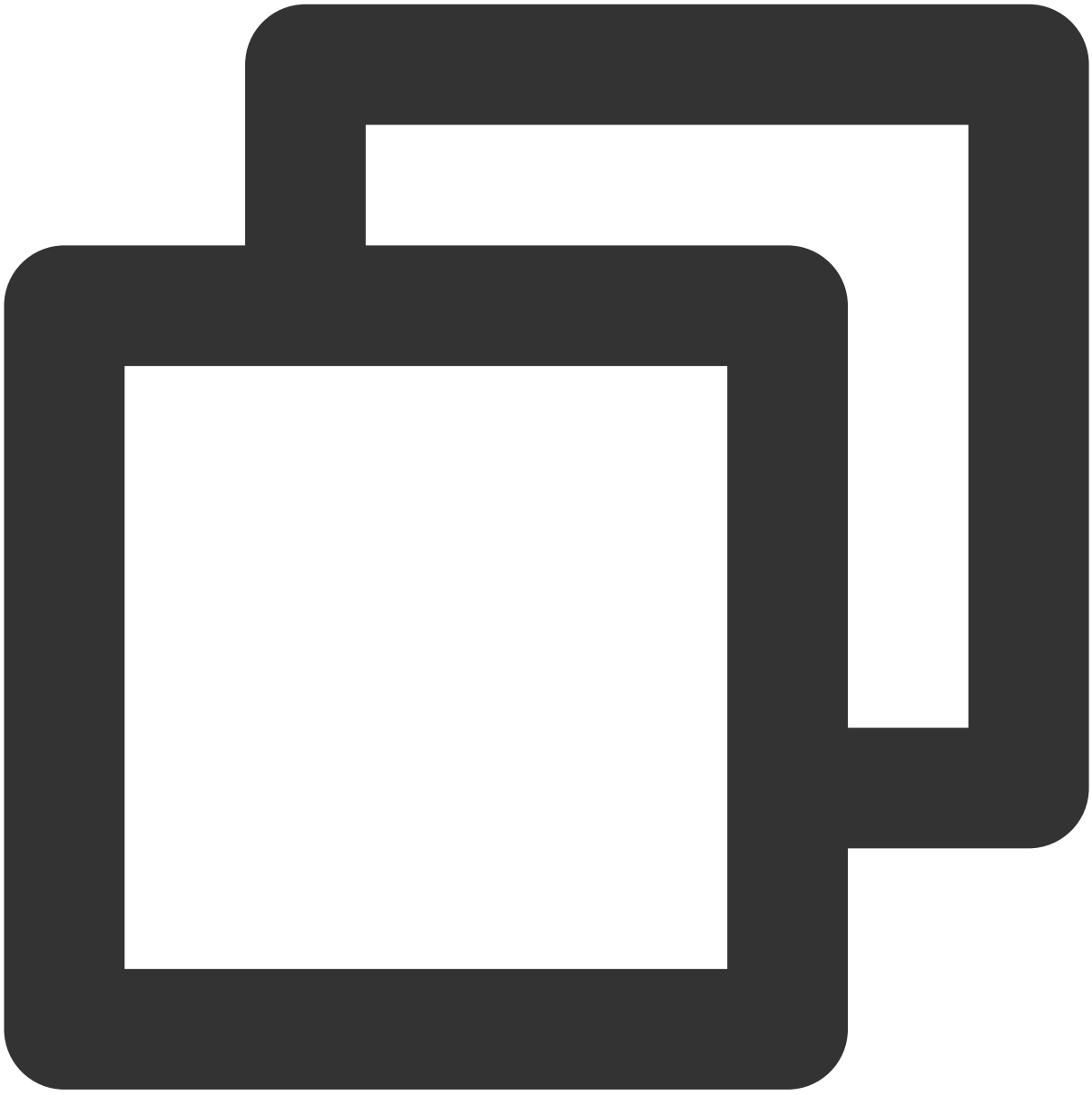
错误码	描述	状态码
InvalidArgument	不合法的参数值	HTTP 400 Bad Request
TooManyConfigurations	清单数量已经达到1000条的上限	HTTP 400 Bad Request
AccessDenied	未授权的访问。您可能不具备访问该存储桶的权限	HTTP 403 Forbidden

查询清单任务

功能说明

GET Bucket inventory 用于查询存储桶中用户的清单任务信息。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
GetBucketInventoryRequest getBucketInventoryRequest =
        new GetBucketInventoryRequest(bucket);
getBucketInventoryRequest.setInventoryId("exampleInventoryId");
```

```
cosXmlService.getBucketInventoryAsync(getBucketInventoryRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketInventoryResult getBucketInventoryResult =
                (GetBucketInventoryResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

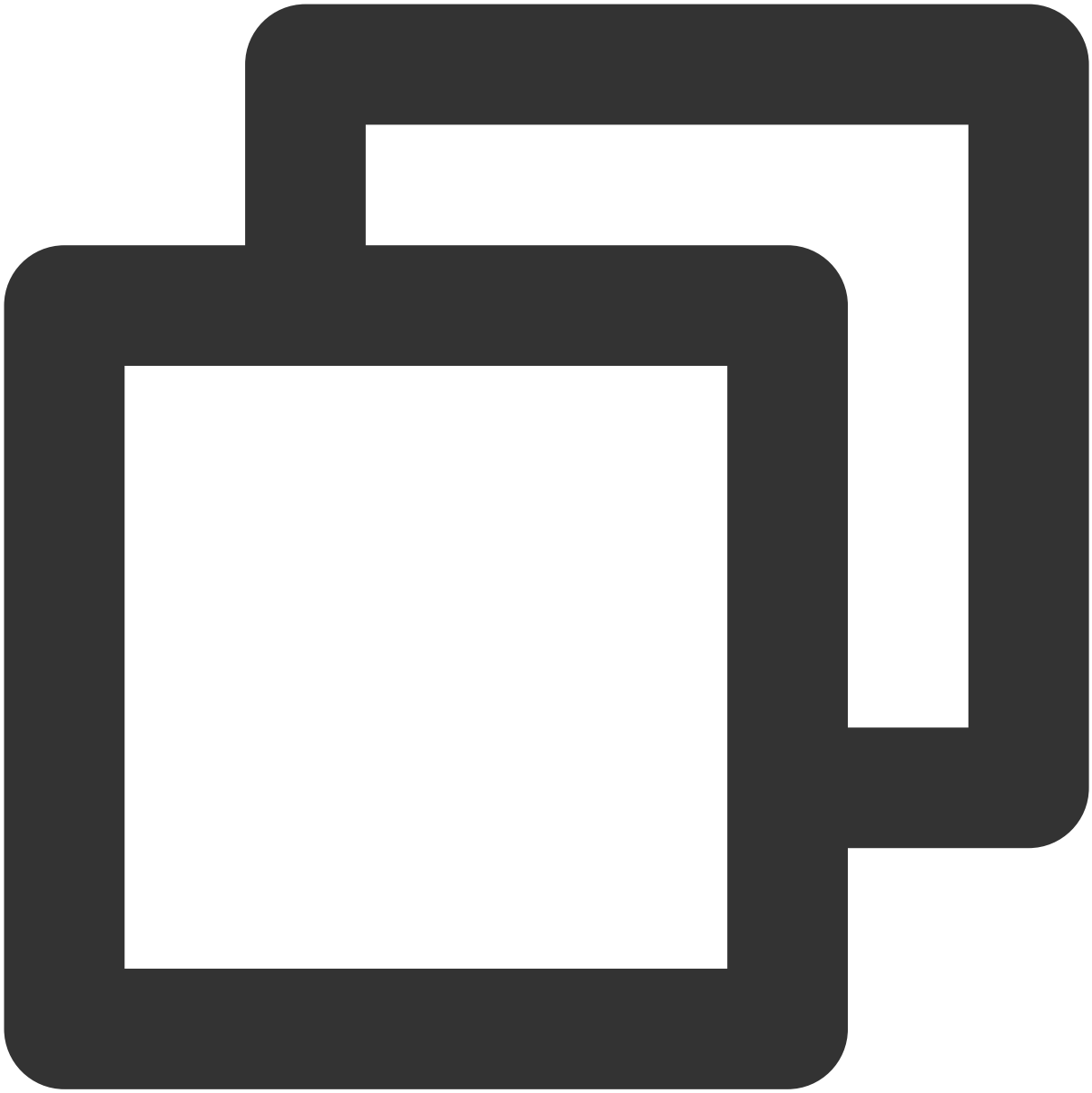
更多完整示例，请前往 GitHub 查看。

删除清单任务

功能说明

DELETE Bucket inventory 用于删除存储桶中指定的清单任务。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
DeleteBucketInventoryRequest deleteBucketInventoryRequest =
        new DeleteBucketInventoryRequest(bucket);
deleteBucketInventoryRequest.setInventoryId("exampleInventoryId");

cosXmlService.deleteBucketInventoryAsync(deleteBucketInventoryRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        DeleteBucketInventoryResult deleteBucketInventoryResult =
```

```
                (DeleteBucketInventoryResult) result;
        }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest cosXmlRequest,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

访问管理
跨域访问

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于跨域访问的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域名访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域名访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域名访问配置信息

SDK API 参考

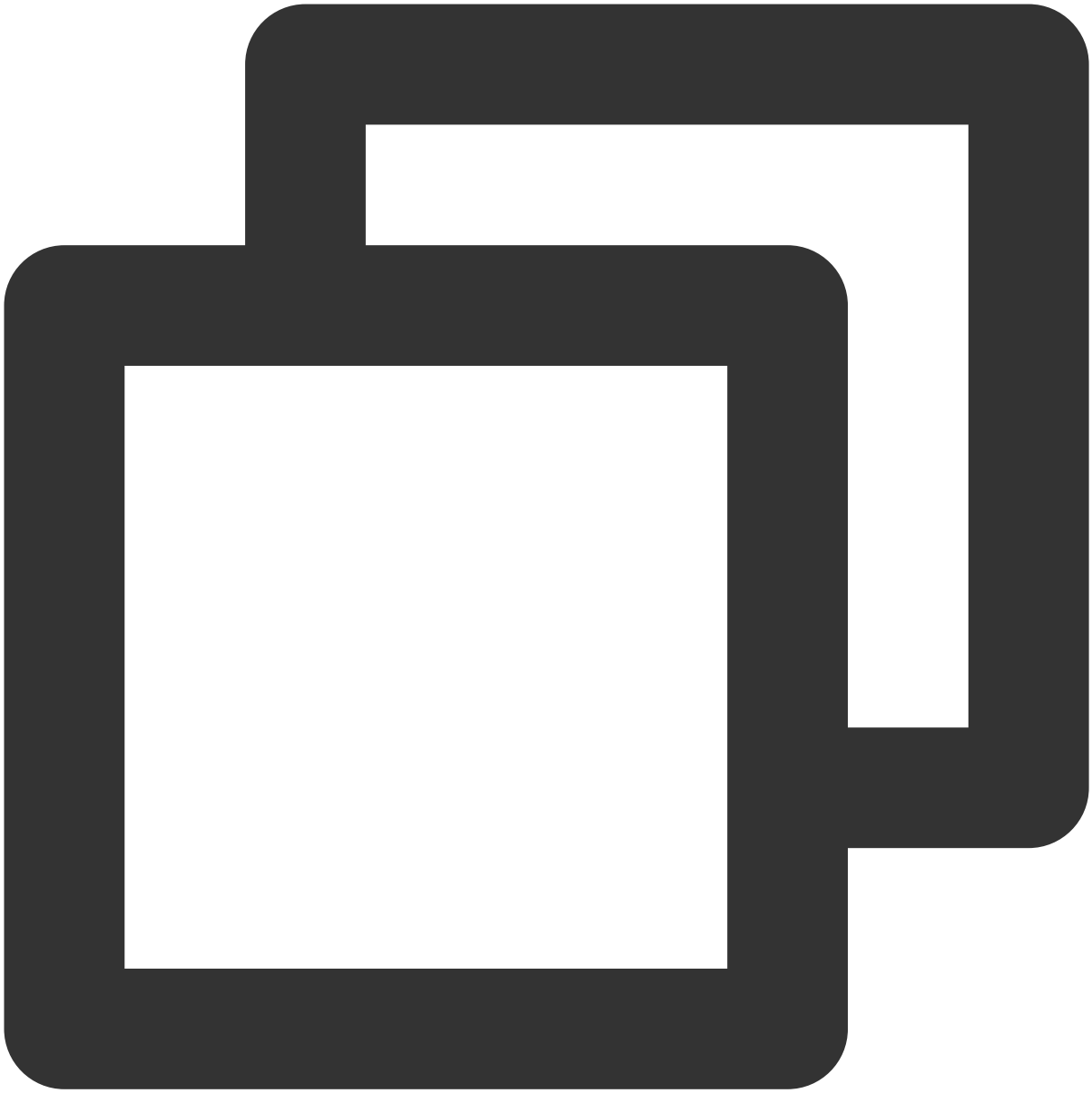
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置跨域配置

功能说明

设置指定存储桶的跨域名访问配置信息（PUT Bucket cors）。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
PutBucketCORSRequest putBucketCORSRequest = new PutBucketCORSRequest(bucket);

CORSConfiguration.CORSRule corsRule = new CORSConfiguration.CORSRule();

// 配置规则的 ID
corsRule.id = "123";
// 允许的访问来源，支持通配符 *，格式为：协议://域名[:端口]
corsRule.allowedOrigin = "https://cloud.tencent.com";
// 设置 OPTIONS 请求得到结果的有效期
```

```
corsRule.maxAgeSeconds = 5000;

List<String> methods = new LinkedList<>();
methods.add("PUT");
methods.add("POST");
methods.add("GET");
// 允许的 HTTP 操作，例如：GET, PUT, HEAD, POST, DELETE
corsRule.allowedMethod = methods;

List<String> headers = new LinkedList<>();
headers.add("host");
headers.add("content-type");
// 在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用的 HTTP 请求头部，支持通配符 *
corsRule.allowedHeader = headers;

List<String> exposeHeaders = new LinkedList<>();
exposeHeaders.add("x-cos-meta-1");
// 设置浏览器可以接收到的来自服务端的自定义头部信息
corsRule.exposeHeader = exposeHeaders;

putBucketCORSRequest.addCORSRule(corsRule);

cosXmlService.putBucketCORSAsync(putBucketCORSRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutBucketCORSResult putBucketCORSResult = (PutBucketCORSResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

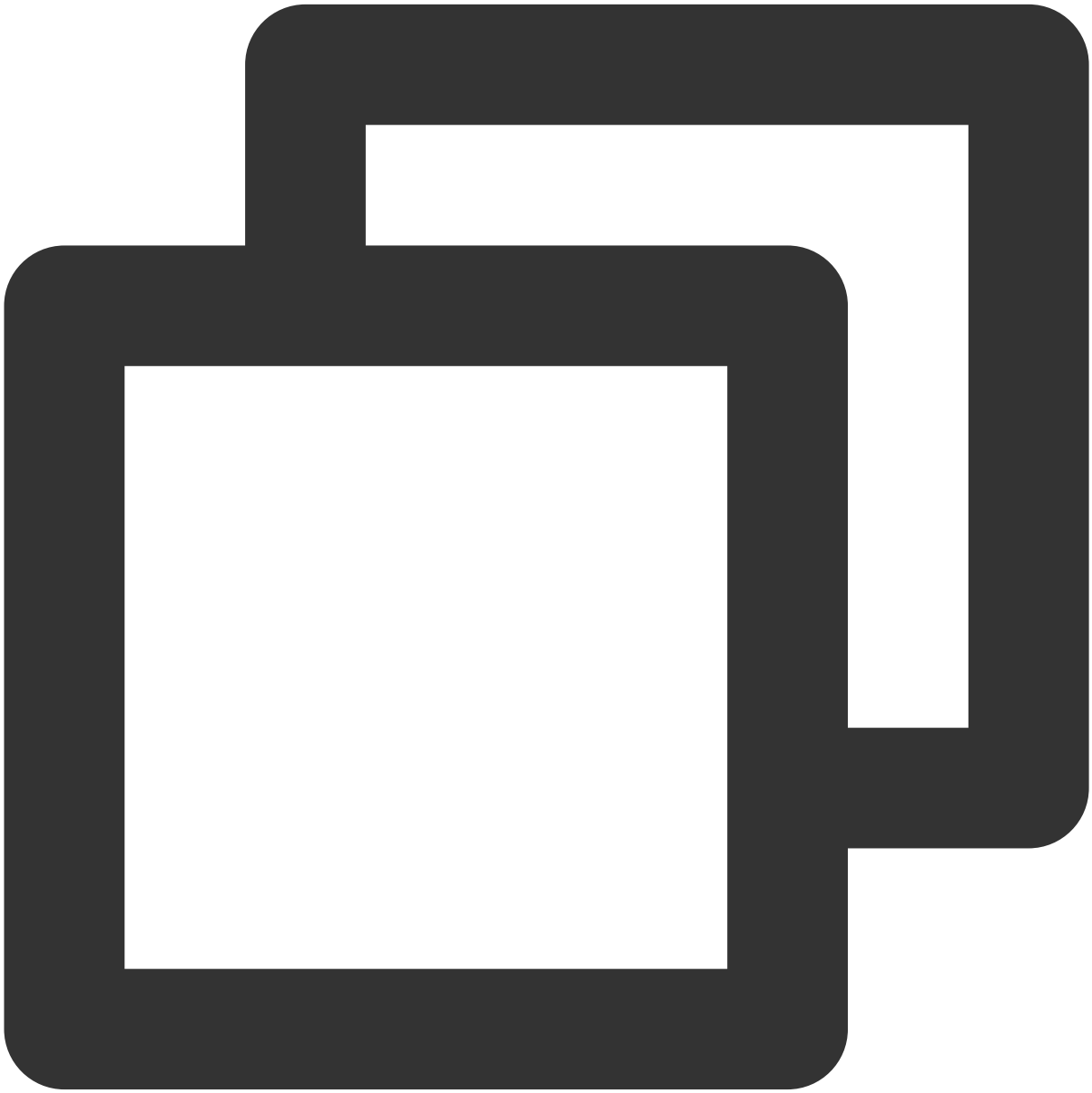
更多完整示例，请前往 GitHub 查看。

查询跨域配置

功能说明

查询指定存储桶的跨域名访问配置信息（GET Bucket cors）。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
GetBucketCORSRequest getBucketCORSRequest = new GetBucketCORSRequest(bucket);
```

版权所有：腾讯云计算（北京）有限责任公司

```
cosXmlService.getBucketCORSAsync(getBucketCORSRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketCORSResult getBucketCORSResult = (GetBucketCORSResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

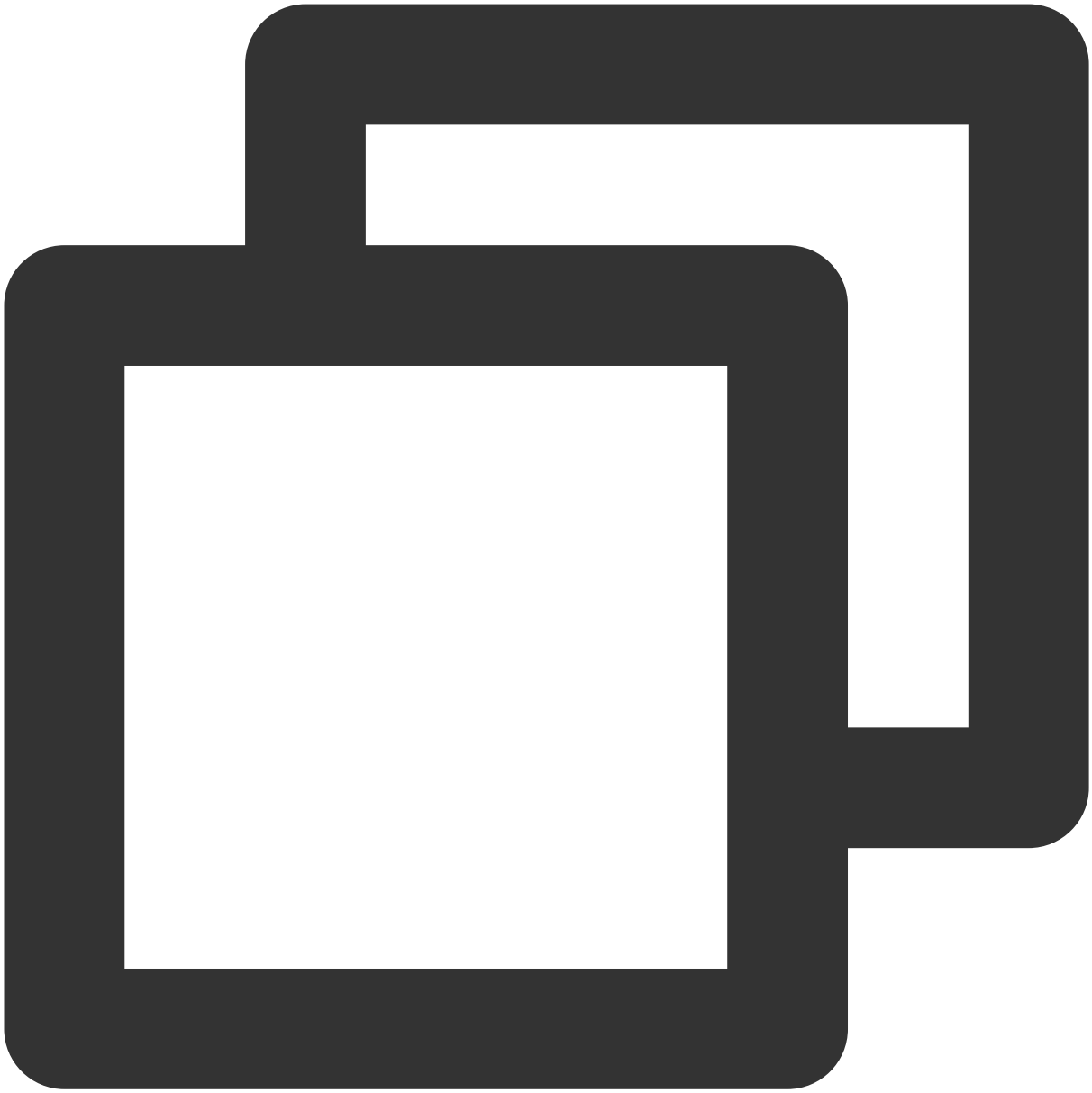
更多完整示例，请前往 GitHub 查看。

删除跨域配置

功能说明

删除指定存储桶的跨域名访问配置（DELETE Bucket cors）。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
DeleteBucketCORSRequest deleteBucketCORSRequest =
        new DeleteBucketCORSRequest(bucket);
cosXmlService.deleteBucketCORSAsync(deleteBucketCORSRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        DeleteBucketCORSResult deleteBucketCORSResult =
                (DeleteBucketCORSResult) result;
    }
```

```
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

自定义域名

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于自定义域名的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket domain	设置自定义域名	设置存储桶的自定义域名信息
GET Bucket domain	查询自定义域名	查询存储桶的自定义域名信息
DELETE Bucket domain	删除自定义域名	删除存储桶的自定义域名配置信息

SDK API 参考

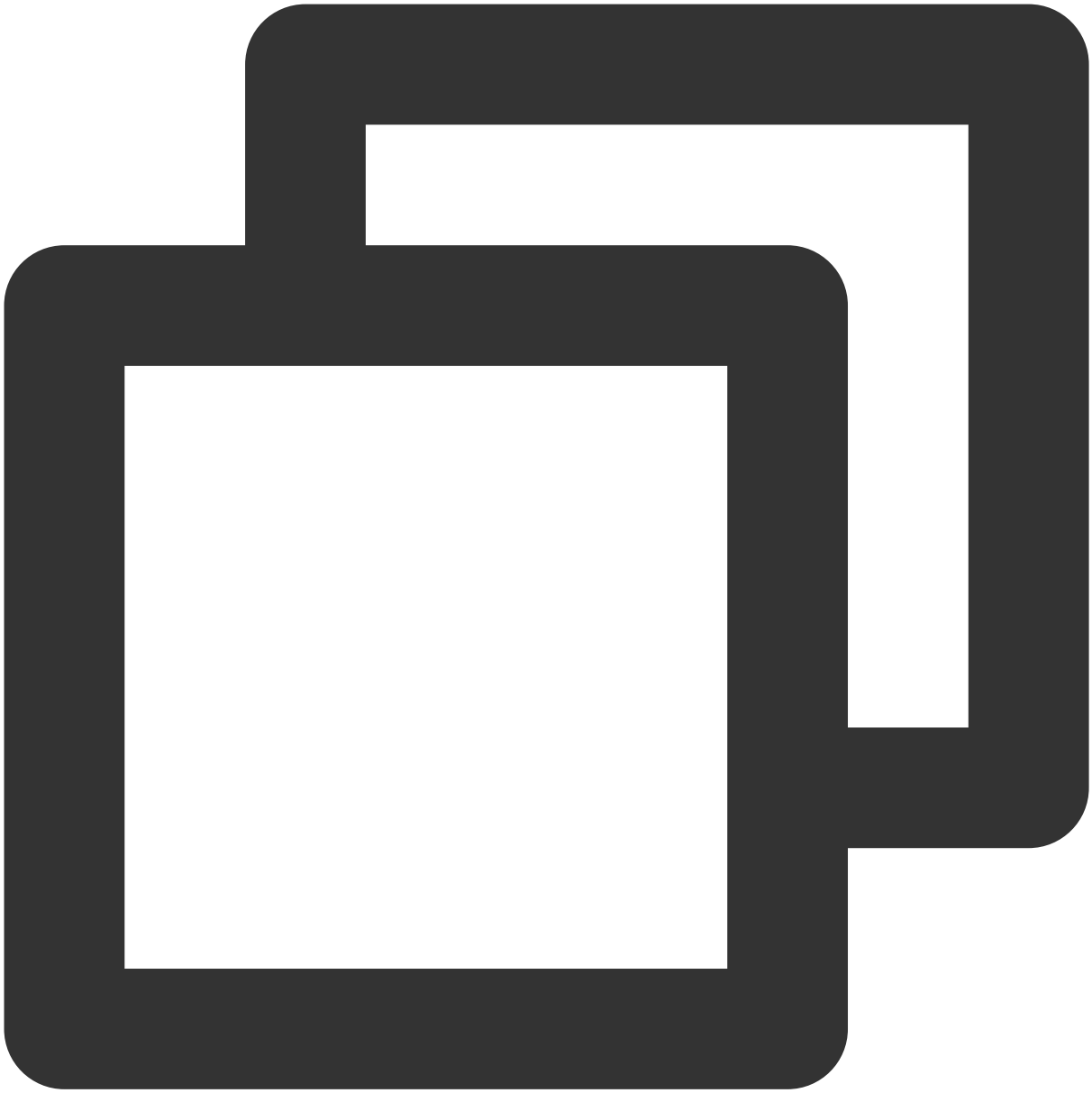
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置自定义域名

功能说明

PUT Bucket domain 用于为存储桶配置自定义域名。

示例代码



```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
PutBucketDomainRequest putBucketDomainRequest =
        new PutBucketDomainRequest(bucket);
DomainConfiguration.DomainRule domainRule = new DomainConfiguration.DomainRule(
        DomainConfiguration.STATUS_ENABLED,
        "www.example.com",
        DomainConfiguration.TYPE_REST
);
domainRule.forcedReplacement = DomainConfiguration.REPLACE_CNAME;
putBucketDomainRequest.addDomainRule(domainRule);
```

```
cosXmlService.putBucketDomainAsync(putBucketDomainRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutBucketDomainResult putBucketDomainResult =
                (PutBucketDomainResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

返回错误码说明

该请求可能会发生的一些常见的特殊错误如下：

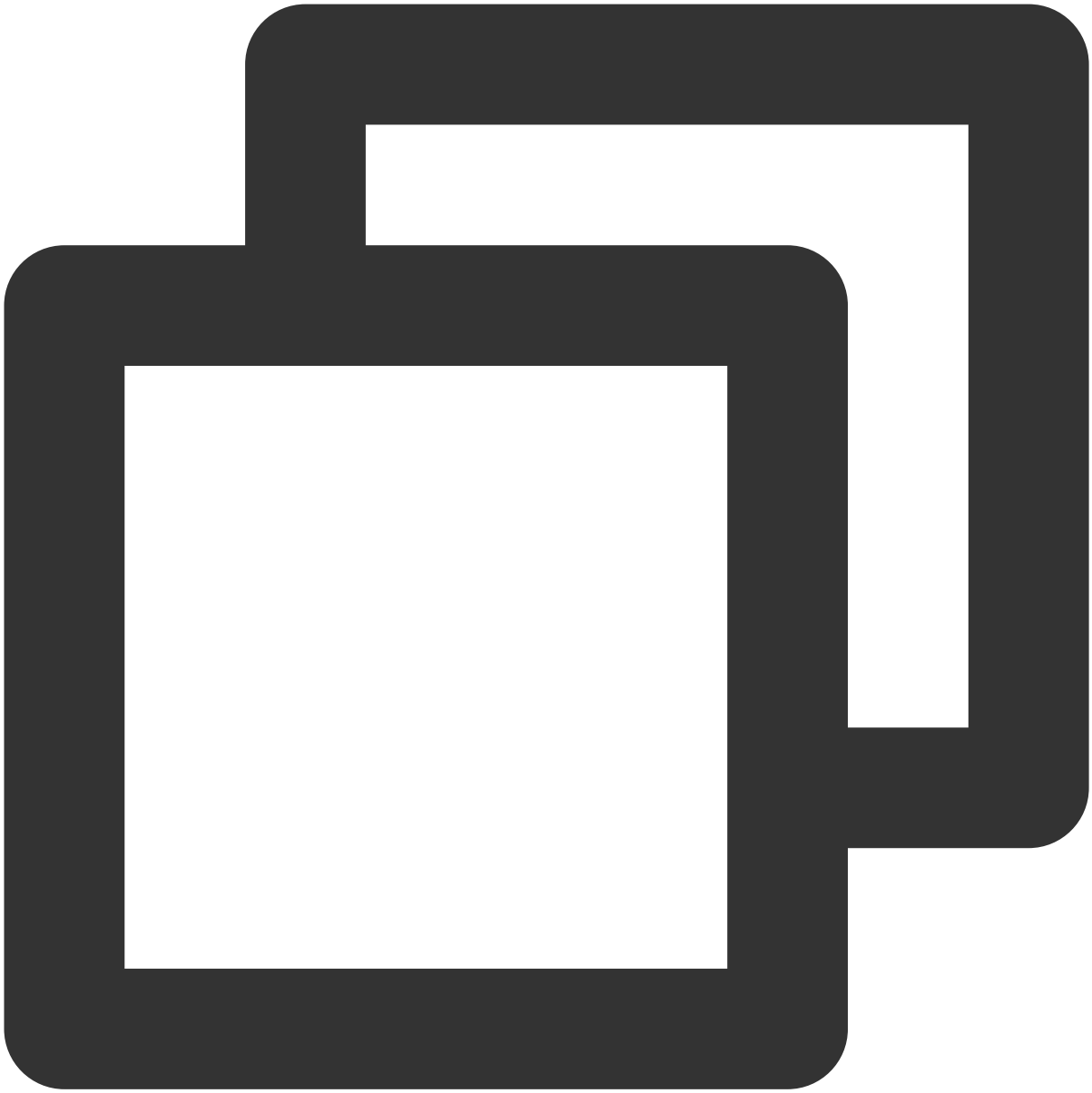
状态码	说明
HTTP 409 Conflict	该域名记录已存在，且请求中没有设置强制覆盖。或者该域名记录不存在，且请求中设置了强制覆盖
HTTP 451 Unavailable For Legal Reasons	该域名是中国境内域名，并且没有备案

查询自定义域名

功能说明

GET Bucket domain 用于查询存储桶的自定义域名信息。

示例代码



```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
GetBucketDomainRequest getBucketDomainRequest =
        new GetBucketDomainRequest(bucket);
cosXmlService.getBucketDomainAsync(getBucketDomainRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketDomainResult getBucketTaggingResult =
                (GetBucketDomainResult) result;
    }
```

```
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

返回参数说明

参数名称	描述	类型
x-cos-domain-txt-verification	域名校验信息，该字段是一个 MD5 校验值，原串格式为：cos[Region][BucketName-APPID][BucketCreateTime]，其中 Region 为存储桶所在地域，BucketCreateTime 为存储桶 GMT 创建时间	String

删除自定义域名

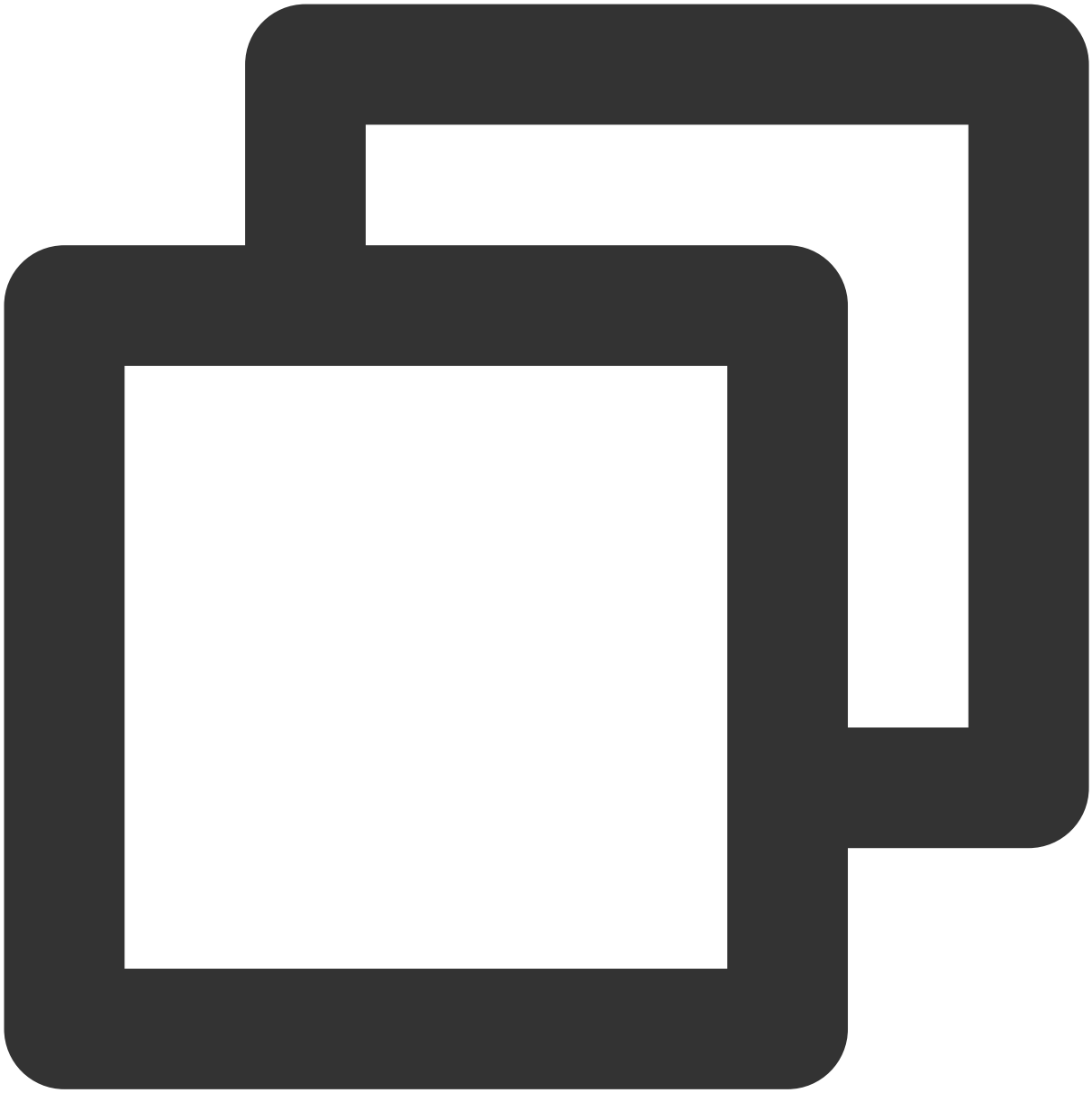
功能说明

DELETE Bucket domain 用于删除存储桶的自定义域名信息。

注意

COS Android SDK 版本需要大于等于 v5.9.8。

示例代码



```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
DeleteBucketDomainRequest deleteBucketDomainRequest =
        new DeleteBucketDomainRequest(bucket);
cosXmlService.deleteBucketDomainAsync(deleteBucketDomainRequest,
        new CosXmlResultListener() {
            @Override
            public void onSuccess(CosXmlRequest request, CosXmlResult result) {
                // 详细字段请查看api文档或者SDK源码
                DeleteBucketDomainResult deleteBucketDomainResult =
                        (DeleteBucketDomainResult) result;
```

```
            }
            // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFa
            // clientException 的类型为 CosXmlClientException?, serviceException 的类
            @Override
            public void onFail(CosXmlRequest cosXmlRequest,
                               @Nullable CosXmlClientException clientException,
                               @Nullable CosXmlServiceException serviceException) {
                if (clientException != null) {
                    clientException.printStackTrace();
                } else {
                    serviceException.printStackTrace();
                }
            }
        });
```

说明

更多完整示例，请前往 GitHub 查看。

访问控制

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于存储桶、对象的访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

存储桶 ACL

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶的访问权限控制列表（ACL）
GET Bucket acl	查询存储桶 ACL	查询指定存储桶的访问权限控制列表（ACL）

对象 ACL

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表
GET Object acl	查询对象 ACL	查询对象的访问控制列表

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

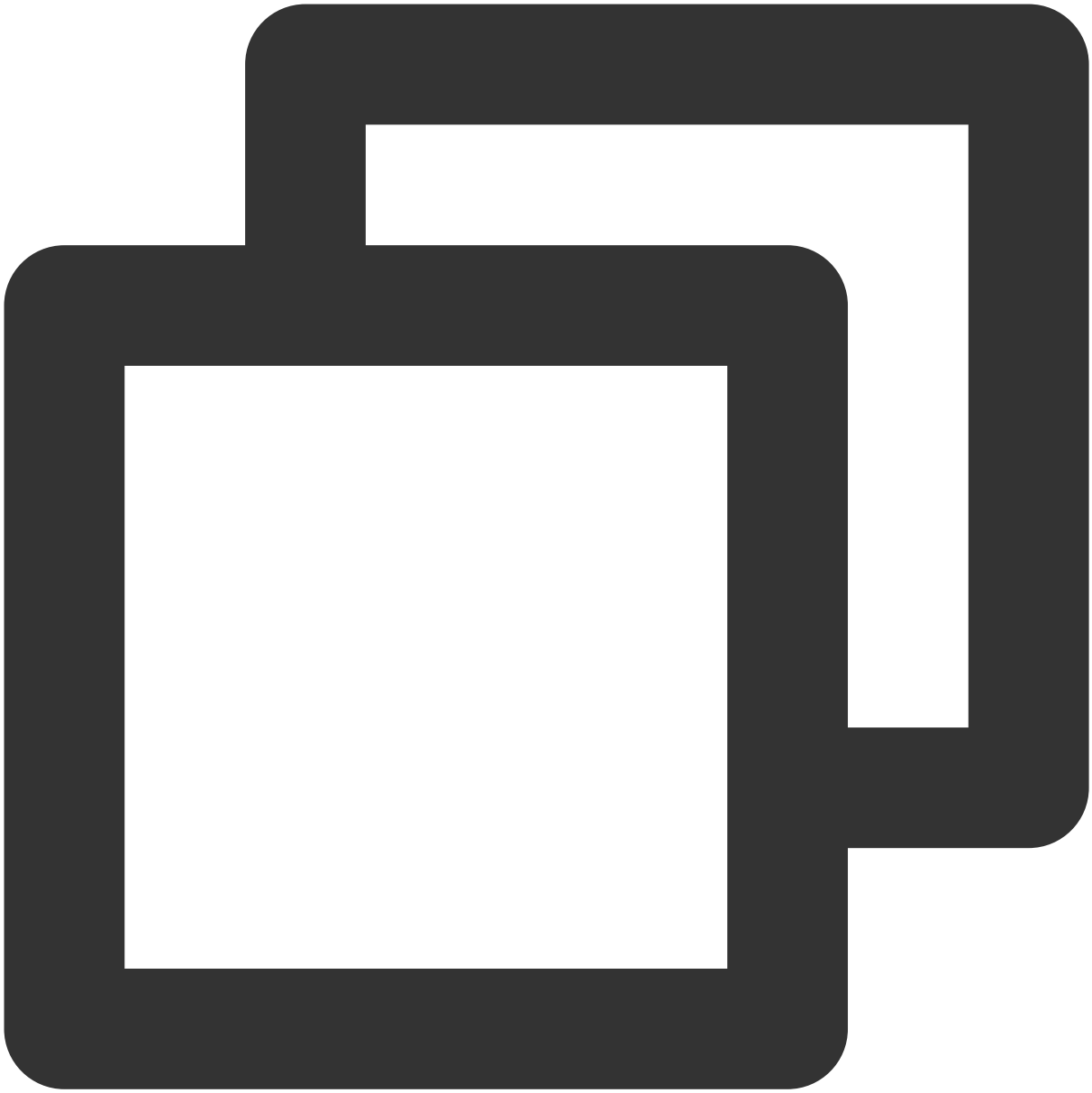
存储桶 ACL

设置存储桶 ACL

功能说明

设置指定存储桶的访问权限控制列表（ACL）。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
PutBucketACLRequest putBucketACLRequest = new PutBucketACLRequest(bucket);

// 设置 bucket 访问权限
putBucketACLRequest.setXCOSACL("public-read");

// 赋予被授权者读的权限
ACLAccount readACLS = new ACLAccount();
readACLS.addAccount("100000000001", "100000000001");
putBucketACLRequest.setXCOSGrantRead(readACLS);
```

```
// 赋予被授权者写的权限
ACLAccount writeACLS = new ACLAccount();
writeACLS.addAccount("100000000001", "100000000001");
putBucketACLRequest.setXCOSGrantWrite(writeACLS);

// 赋予被授权者读写的权限
ACLAccount writeandReadACLS = new ACLAccount();
writeandReadACLS.addAccount("100000000001", "100000000001");
putBucketACLRequest.setXCOSReadWrite(writeandReadACLS);

cosXmlService.putBucketACLAsync(putBucketACLRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutBucketACLResult putBucketACLResult = (PutBucketACLResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

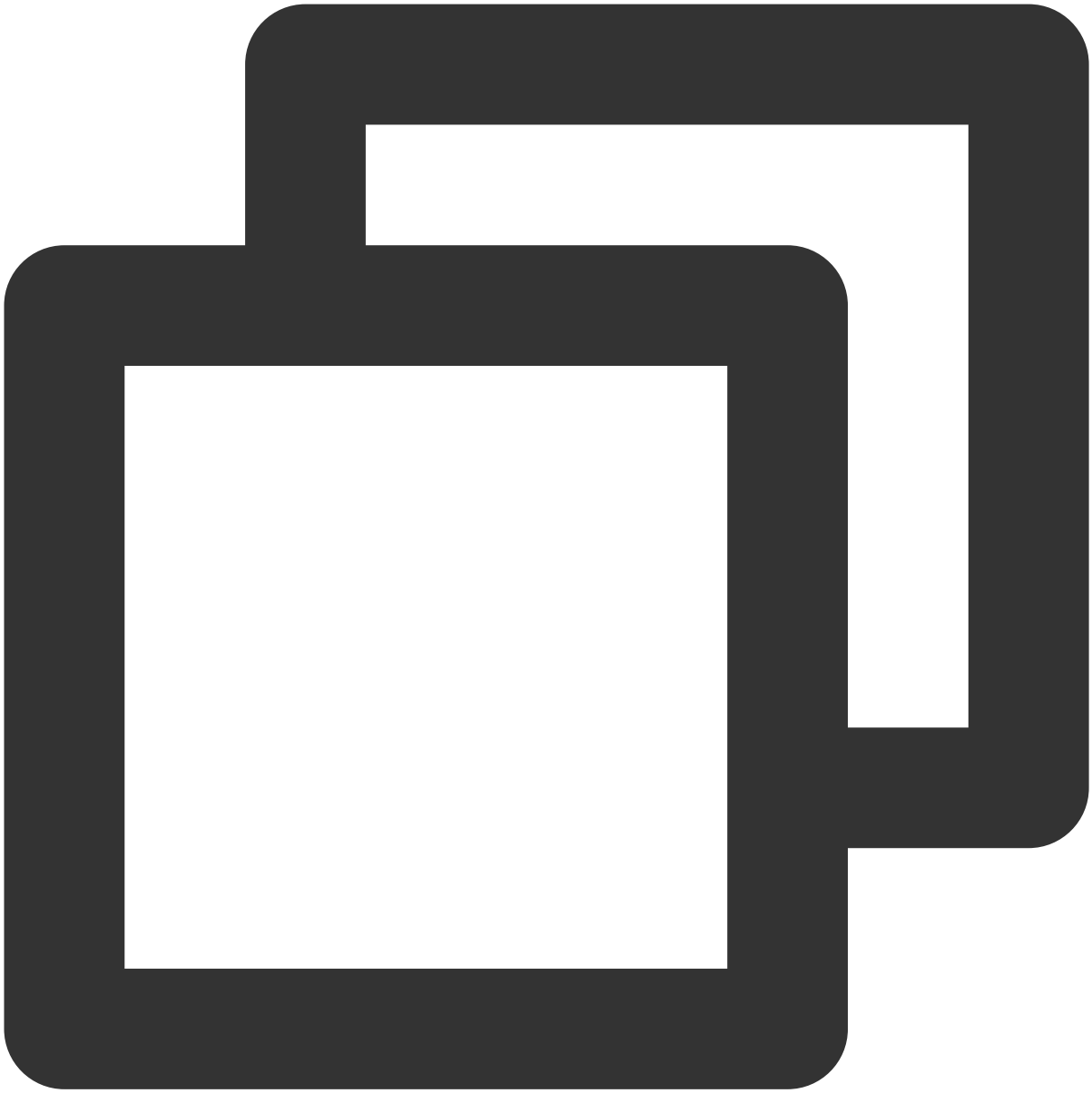
更多完整示例，请前往 GitHub 查看。

查询存储桶 ACL

功能说明

查询指定存储桶的访问权限控制列表（ACL）。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
GetBucketACLRequest getBucketACLRequest = new GetBucketACLRequest(bucket);
cosXmlService.getBucketACLAsync(getBucketACLRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketACLResult getBucketACLResult = (GetBucketACLResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
```

```
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

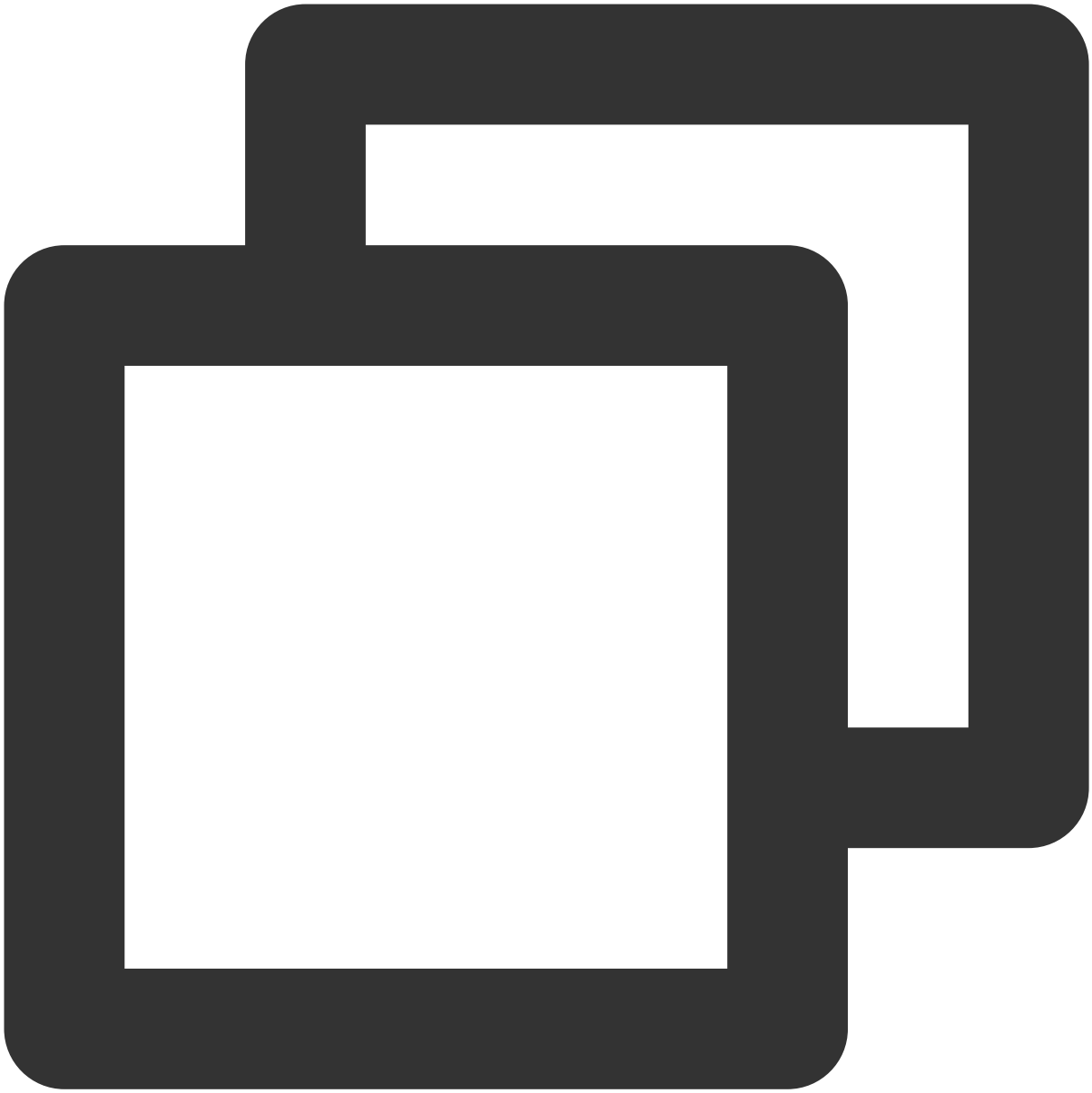
对象 ACL

设置对象 ACL

功能说明

设置存储桶中某个对象的访问控制列表（ACL）。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键。
PutObjectACLRequest putObjectACLRequest = new PutObjectACLRequest(bucket,
        cosPath);

// 设置 对象 访问权限
putObjectACLRequest.setXCOSACL("public-read");

// 赋予被授权者读的权限
ACLAccount readACLS = new ACLAccount();
```

```
readACLS.addAccount("100000000001", "100000000001");
putObjectACLRequest.setXCOSGrantRead(readACLS);

// 赋予被授权者读写的权限
ACLAccount writeandReadACLS = new ACLAccount();
writeandReadACLS.addAccount("100000000001", "100000000001");
putObjectACLRequest.setXCOSReadWrite(writeandReadACLS);

cosXmlService.putObjectACLAsync(putObjectACLRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutObjectACLResult putObjectACLResult = (PutObjectACLResult) result;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

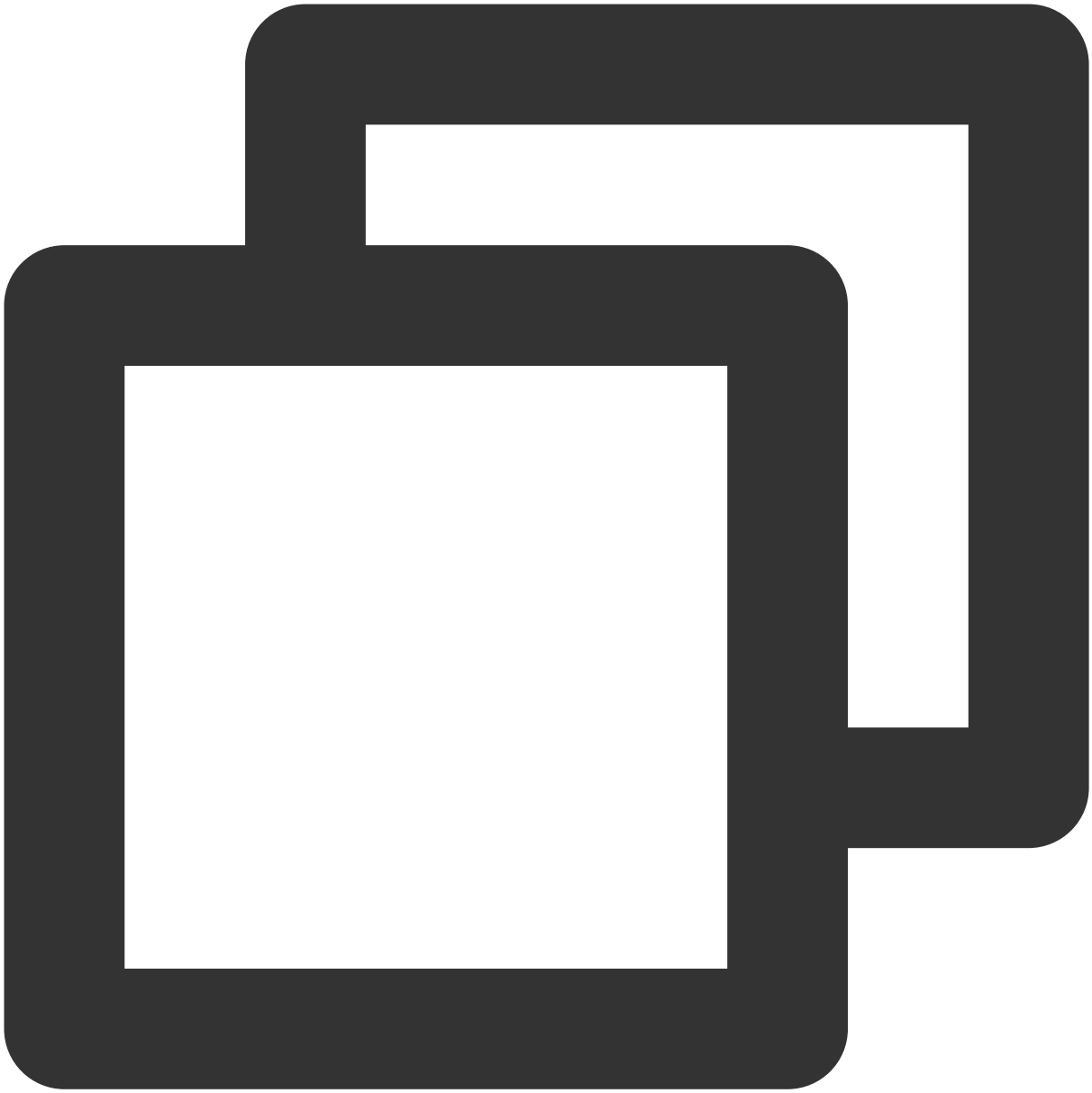
更多完整示例，请前往 GitHub 查看。

查询对象 ACL

功能说明

查询对象的访问控制列表。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键。
GetObjectACLRequest getBucketACLRequest = new GetObjectACLRequest(bucket,
        cosPath);
cosXmlService.getObjectACLAsync(getBucketACLRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetObjectACLResult getObjectACLResult = (GetObjectACLResult) result;
    }
```

```
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

防盗链

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于防盗链的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket referer	设置防盗链配置	设置存储桶的防盗链
GET Bucket referer	查询防盗链配置	查询存储桶的防盗链配置信息

SDK API 参考

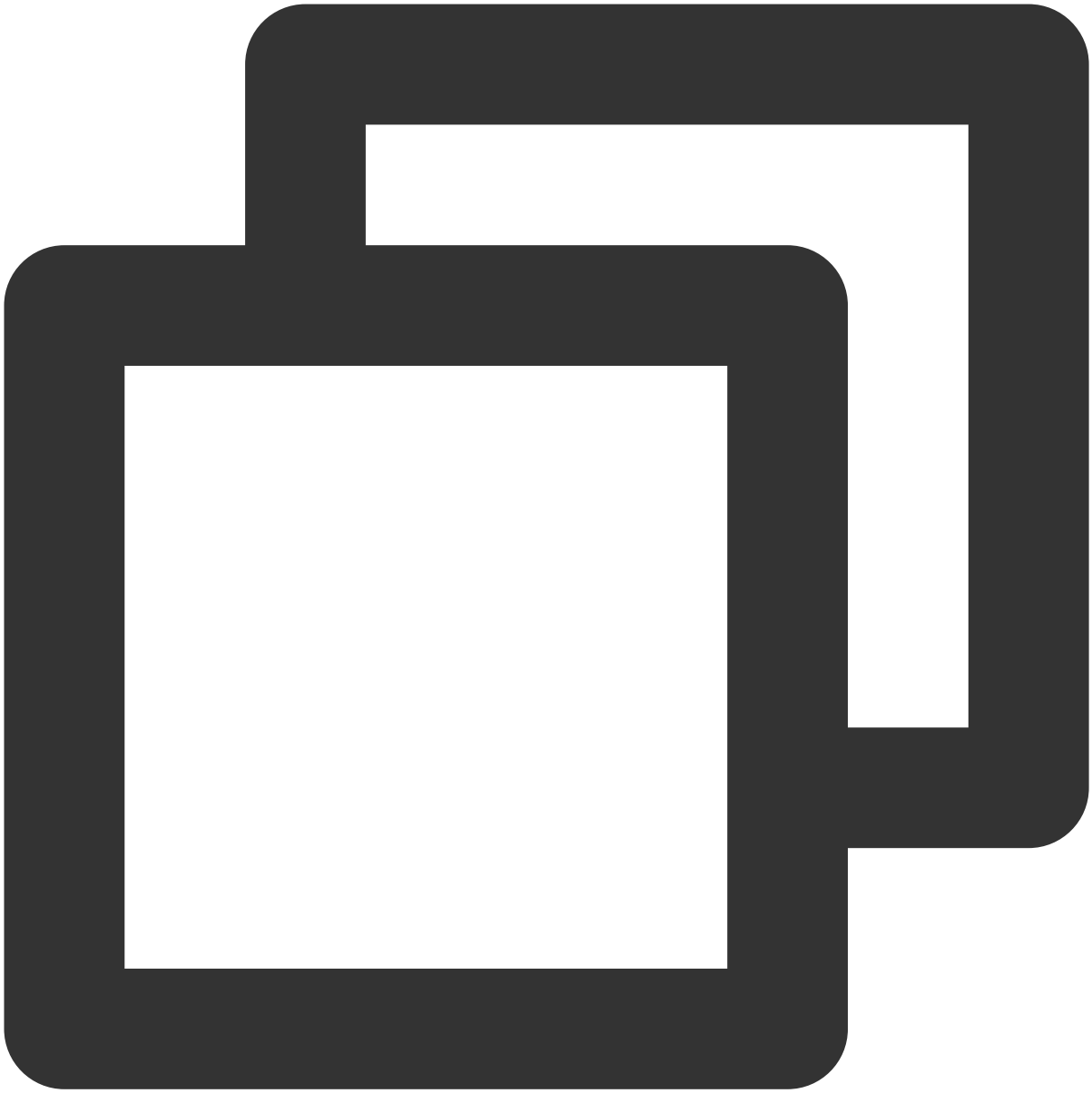
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置防盗链配置

功能说明

设置指定存储桶的防盗链配置信息（PUT Bucket referer）。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
PutBucketRefererRequest putBucketRefererRequest = new PutBucketRefererRequest(
        bucket, true, RefererConfiguration.RefererType.White);
putBucketRefererRequest.setAllowEmptyRefer(false);
ArrayList<RefererConfiguration.Domain> domainList = new ArrayList<>();
domainList.add(new RefererConfiguration.Domain("*.qq.com"));
domainList.add(new RefererConfiguration.Domain("*.qcloud.com"));
domainList.add(new RefererConfiguration.Domain("*.google.com"));
putBucketRefererRequest.setDomainList(domainList);
cosXmlService.putBucketRefererAsync(putBucketRefererRequest,
```

```
        new CosXmlResultListener() {
            @Override
            public void onSuccess(CosXmlRequest request, CosXmlResult result) {
                // 详细字段请查看api文档或者SDK源码
                PutBucketRefererResult putBucketRefererResult =
                        (PutBucketRefererResult) result;
            }
            // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFa
            // clientException 的类型为 CosXmlClientException?, serviceException 的类
            @Override
            public void onFail(CosXmlRequest cosXmlRequest,
                               @Nullable CosXmlClientException clientException,
                               @Nullable CosXmlServiceException serviceException) {
                if (clientException != null) {
                    clientException.printStackTrace();
                } else {
                    serviceException.printStackTrace();
                }
            }
        });
```

说明

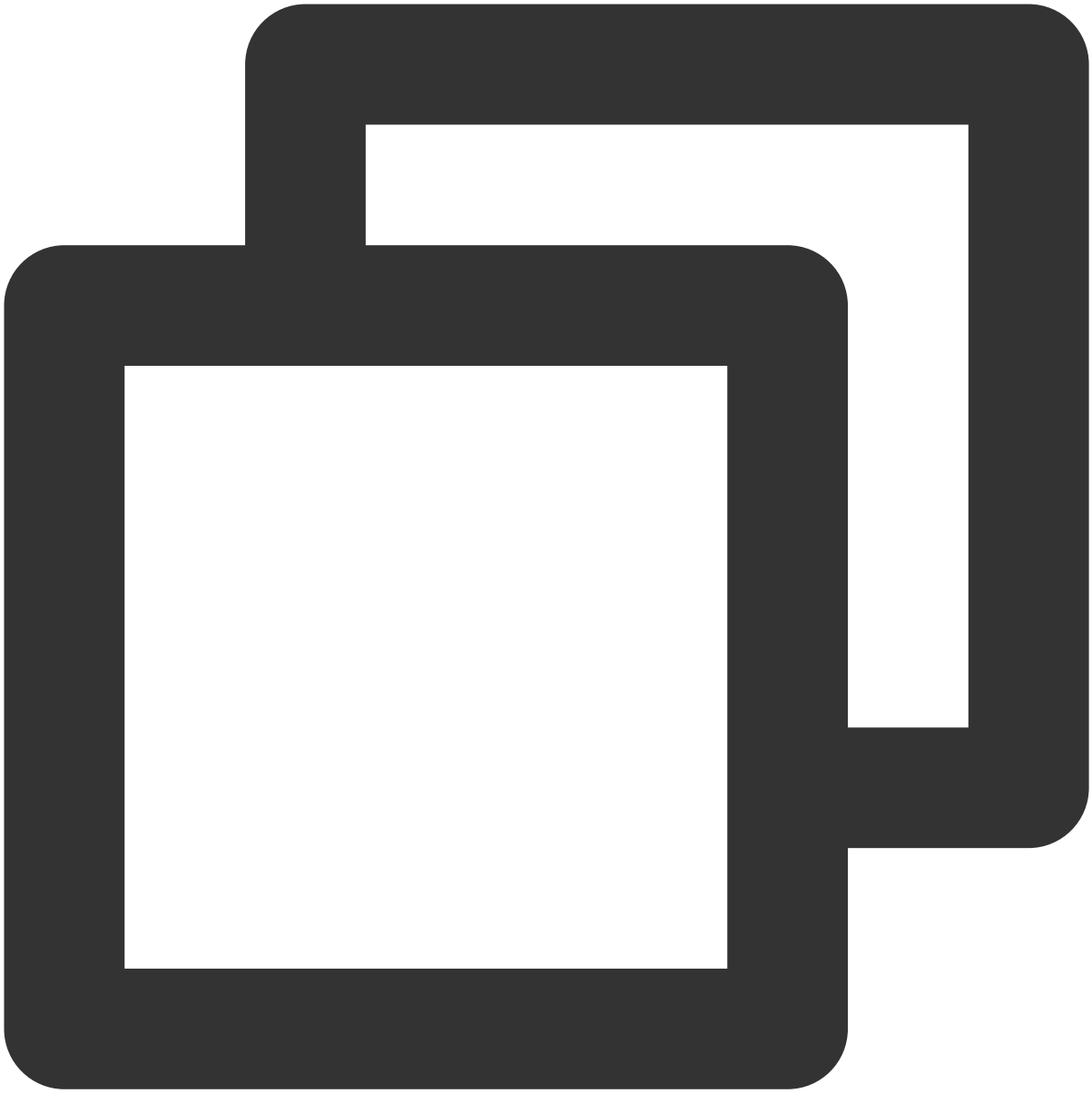
更多完整示例，请前往 GitHub 查看。

查询防盗链配置

功能说明

查询指定存储桶的防盗链配置信息（GET Bucket referer）。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
GetBucketRefererRequest getBucketRefererRequest = new GetBucketRefererRequest(bucke
cosXmlService.getBucketRefererAsync(getBucketRefererRequest,
        new CosXmlResultListener() {
            @Override
            public void onSuccess(CosXmlRequest request, CosXmlResult result) {
                // 详细字段请查看api文档或者SDK源码
                GetBucketRefererResult getBucketRefererResult =
                        (GetBucketRefererResult) result;
            }
```

```
            // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFa
            // clientException 的类型为 CosXmlClientException?, serviceException 的类
            @Override
            public void onFail(CosXmlRequest cosXmlRequest,
                               @Nullable CosXmlClientException clientException,
                               @Nullable CosXmlServiceException serviceException) {
                if (clientException != null) {
                    clientException.printStackTrace();
                } else {
                    serviceException.printStackTrace();
                }
            }
        });
```

说明

更多完整示例，请前往 GitHub 查看。

存储桶策略

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于存储桶策略的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket policy	设置存储桶策略	设置指定存储桶的权限策略
GET Bucket policy	查询存储桶策略	查询指定存储桶的权限策略
DELETE Bucket policy	删除存储桶策略	删除指定存储桶的权限策略

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置存储桶策略

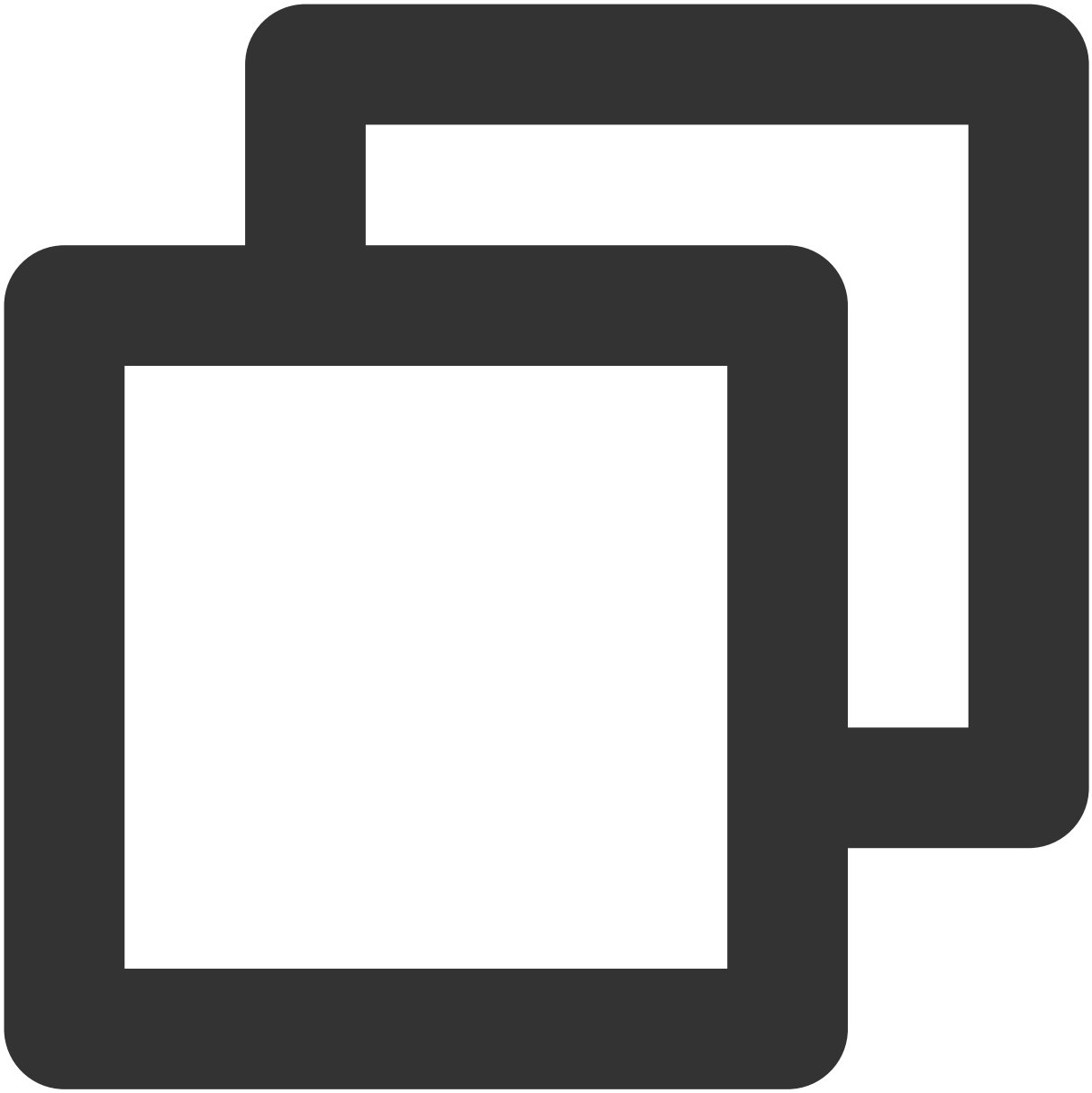
功能说明

设置指定存储桶的权限策略（PUT Bucket policy）。

注意

COS Android SDK 版本需要大于等于 v5.9.8。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
// 权限策略，详情请参见 访问管理策略语法 https://cloud.tencent.com/document/product/436/1
String policy = "{\\n" +
        "  \\"Statement\\": [\\n" +
        "    {\\n" +
        "      \\"Principal\\": {\\n" +
        "        \\"qcs\\": [\\n" +
        "          \\"qcs::cam::uin/100000000001:uin/100000000011\\"\\n" +
        "        ]\\n" +
        "      },\\n" +
```

```
        "      \\"Effect\\": \\"allow\\",\\n" +
        "      \\"Action\\": [\\n" +
        "        \\"name/cos:GetBucket\\"\\n" +
        "      ],\\n" +
        "      \\"Resource\\": [\\n" +
        "        \\"qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*
        "      ]\\n" +
        "    }\\n" +
        "  ],\\n" +
        "  \\"version\\": \\"2.0\\"\\n" +
        "}";
PutBucketPolicyRequest putBucketPolicyRequest =
        new PutBucketPolicyRequest(bucket, policy);
cosXmlService.putBucketPolicyAsync(putBucketPolicyRequest,
        new CosXmlResultListener() {
            @Override
            public void onSuccess(CosXmlRequest request, CosXmlResult result) {
                // 详细字段请查看api文档或者SDK源码
                PutBucketPolicyResult putBucketPolicyResult =
                        (PutBucketPolicyResult) result;
            }
            // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFa
            // clientException 的类型为 CosXmlClientException?, serviceException 的类
            @Override
            public void onFail(CosXmlRequest cosXmlRequest,
                               @Nullable CosXmlClientException clientException,
                               @Nullable CosXmlServiceException serviceException) {
                if (clientException != null) {
                    clientException.printStackTrace();
                } else {
                    serviceException.printStackTrace();
                }
            }
        });
```

说明

更多完整示例，请前往 GitHub 查看。

查询存储桶策略

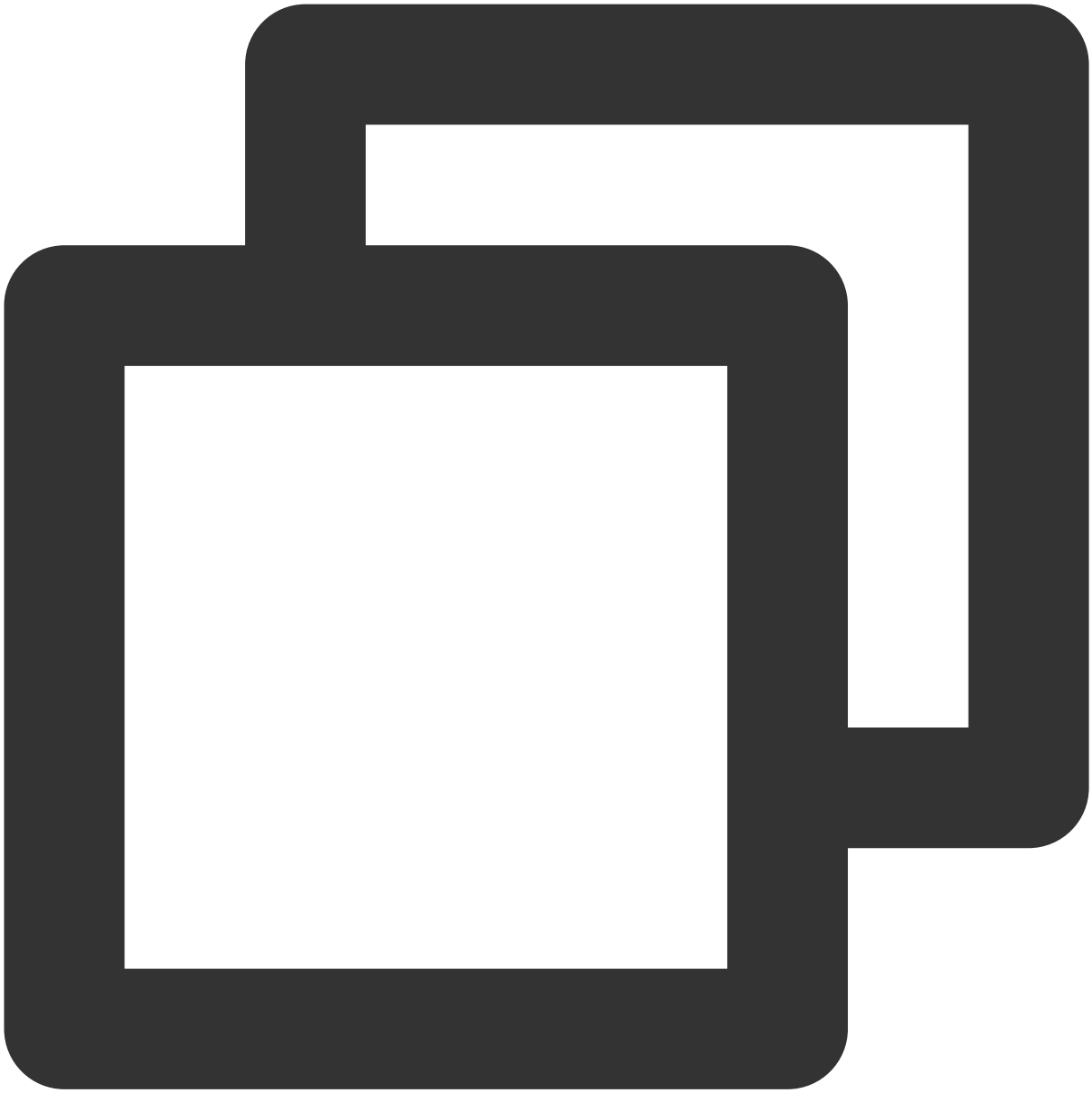
功能说明

查询指定存储桶的权限策略（GET Bucket policy）。

注意

COS Android SDK 版本需要大于等于 v5.9.8。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
final GetBucketPolicyRequest getBucketPolicyRequest =
        new GetBucketPolicyRequest(bucket);
cosXmlService.getBucketPolicyAsync(getBucketPolicyRequest,
        new CosXmlResultListener() {
            @Override
            public void onSuccess(CosXmlRequest request, CosXmlResult result) {
                // 详细字段请查看api文档或者SDK源码
```

```
                GetBucketPolicyResult getBucketPolicyResult =
                        (GetBucketPolicyResult) result;
                String policy = getBucketPolicyResult.policy;
            }
            // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFa
            // clientException 的类型为 CosXmlClientException?, serviceException 的类
            @Override
            public void onFail(CosXmlRequest cosXmlRequest,
                               @Nullable CosXmlClientException clientException,
                               @Nullable CosXmlServiceException serviceException) {
                if (clientException != null) {
                    clientException.printStackTrace();
                } else {
                    serviceException.printStackTrace();
                }
            }
        });
```

说明

更多完整示例，请前往 GitHub 查看。

删除存储桶策略

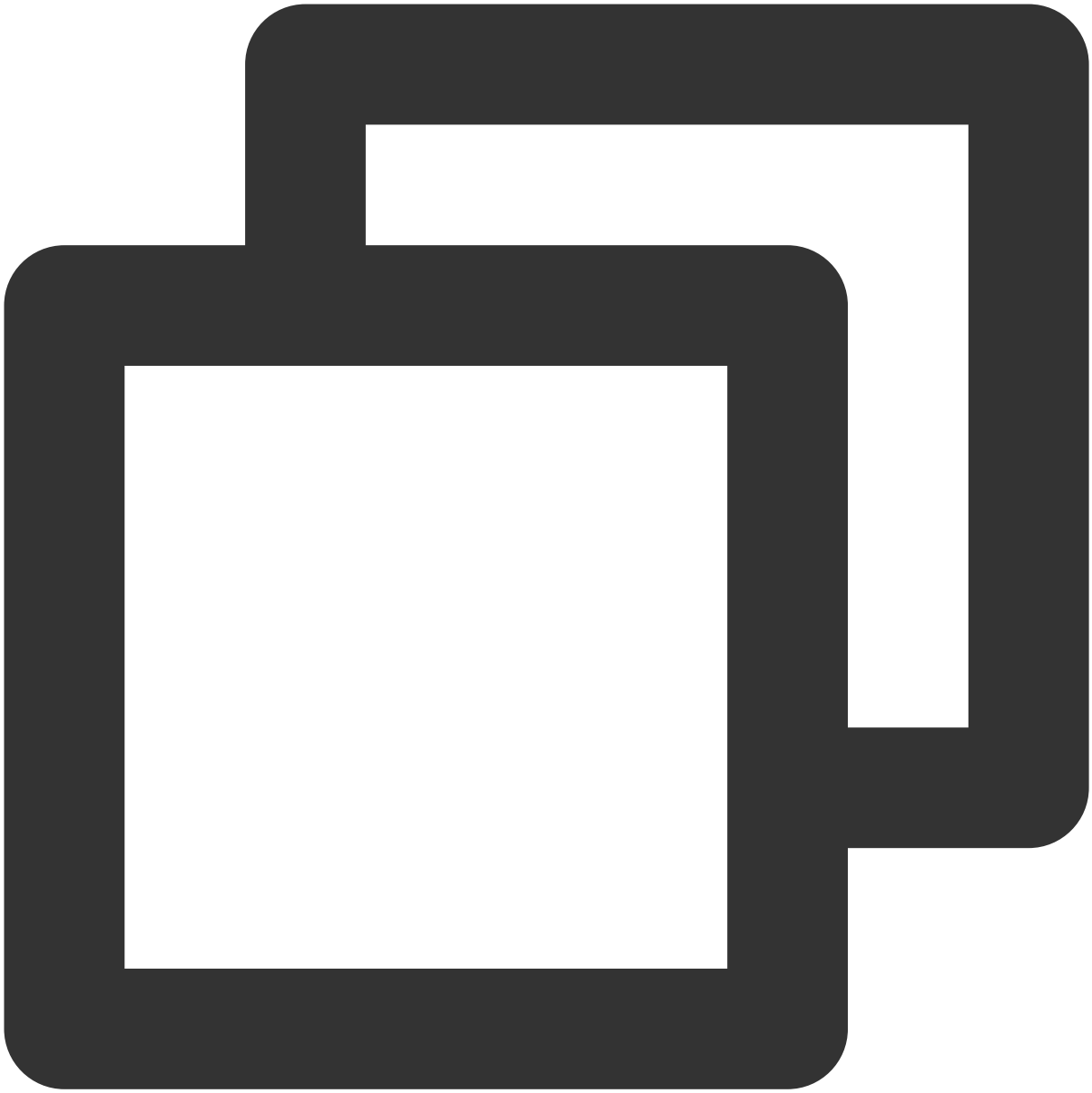
功能说明

删除指定存储桶的权限策略（DELETE Bucket policy）。

注意

COS Android SDK 版本需要大于等于 v5.9.8。

示例代码



```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
DeleteBucketPolicyRequest deleteBucketPolicyRequest =
        new DeleteBucketPolicyRequest(bucket);
cosXmlService.deleteBucketPolicyAsync(deleteBucketPolicyRequest,
        new CosXmlResultListener() {
            @Override
            public void onSuccess(CosXmlRequest request, CosXmlResult result) {
                // 详细字段请查看api文档或者SDK源码
                DeleteBucketPolicyResult deleteBucketPolicyResult =
                        (DeleteBucketPolicyResult) result;
```

```
            }
            // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFa
            // clientException 的类型为 CosXmlClientException?, serviceException 的类
            @Override
            public void onFail(CosXmlRequest cosXmlRequest,
                               @Nullable CosXmlClientException clientException,
                               @Nullable CosXmlServiceException serviceException) {
                if (clientException != null) {
                    clientException.printStackTrace();
                } else {
                    serviceException.printStackTrace();
                }
            }
        });
```

说明

更多完整示例，请前往 GitHub 查看。

数据校验

CRC64 校验

最近更新时间：2024-01-04 15:52:02

简介

数据在客户端和服务器间传输时可能会出现错误，对象存储（Cloud Object Storage，COS）除了可以通过 MD5 和自定义属性 验证数据完整性外，还可以通过 CRC64 检验码来进行数据校验。
COS 会对新上传的对象进行 CRC64 计算，并将结果作为对象的属性进行存储，随后在返回的响应头部中携带 x-cos-hash-crc64ecma，该头部表示上传对象的 CRC64 值，根据 ECMA-182标准计算得到。对于 CRC64 特性上线前就已经存在于 COS 的对象，COS 不会对其计算 CRC64 值，所以获取此类对象时不会返回其 CRC64 值。

操作说明

目前支持 CRC64 的 API 如下：
简单上传接口
PUT Object 和 POST Object ：用户可在返回的响应头中获得文件 CRC64 校验值。
分块上传接口
Upload Part：用户可以根据 COS 返回的 CRC64 值与本地计算的数值进行比较验证。
Complete Multipart Upload：如果每个分块都有 CRC64 属性，则会返回整个对象的 CRC64 值，如果某些分块不具备 CRC64 值，则不返回。
执行 Upload Part - Copy 时，会返回对应的 CRC64 值。
执行 PUT Object - Copy 时，如果源对象存在 CRC64 值，则返回 CRC64，否则不返回。
执行 HEAD Object 和 GET Object 时，如果对象存在 CRC64，则返回。用户可以根据 COS 返回的 CRC64 值和本地计算的 CRC64 进行比较验证。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

SDK 说明

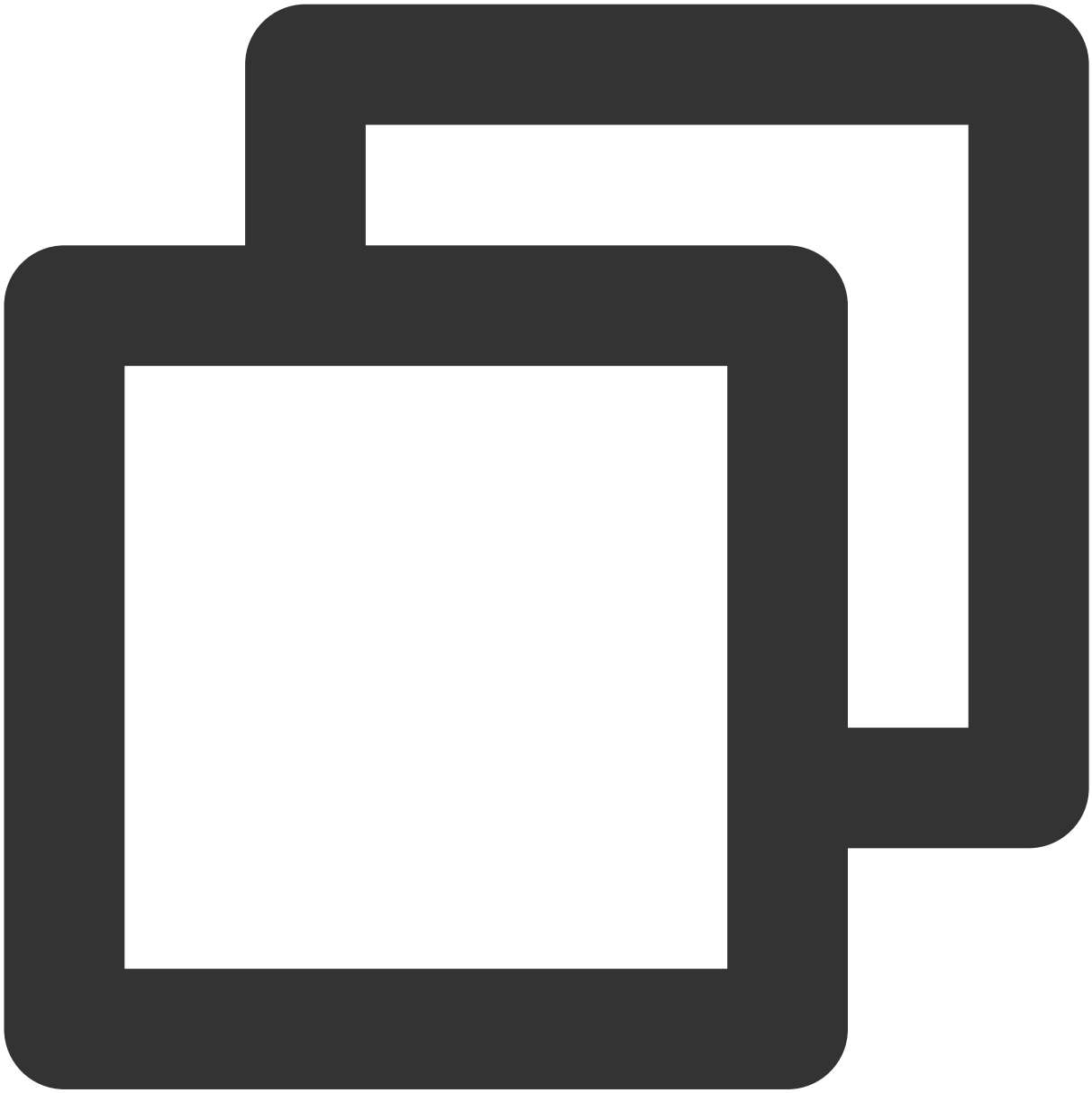
您在上传或者下载成功后，可以在响应头部中获取 CRC64 值。

版权所有：腾讯云计算（北京）有限责任公司

注意

COS Android SDK 版本需要大于等于 v5.7.5。

上传请求示例



```
// 1. 初始化 TransferService。在相同配置的情况下，您应该复用同一个 TransferService
TransferConfig transferConfig = new TransferConfig.Builder()
        .build();
TransferService transferService = new TransferService(cosXmlService, transferConfig

// 2. 初始化 PutObjectRequest
```

```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String srcPath = "examplefilepath"; //本地文件的绝对路径
PutObjectRequest putObjectRequest = new PutObjectRequest(bucket,
        cosPath, srcPath);

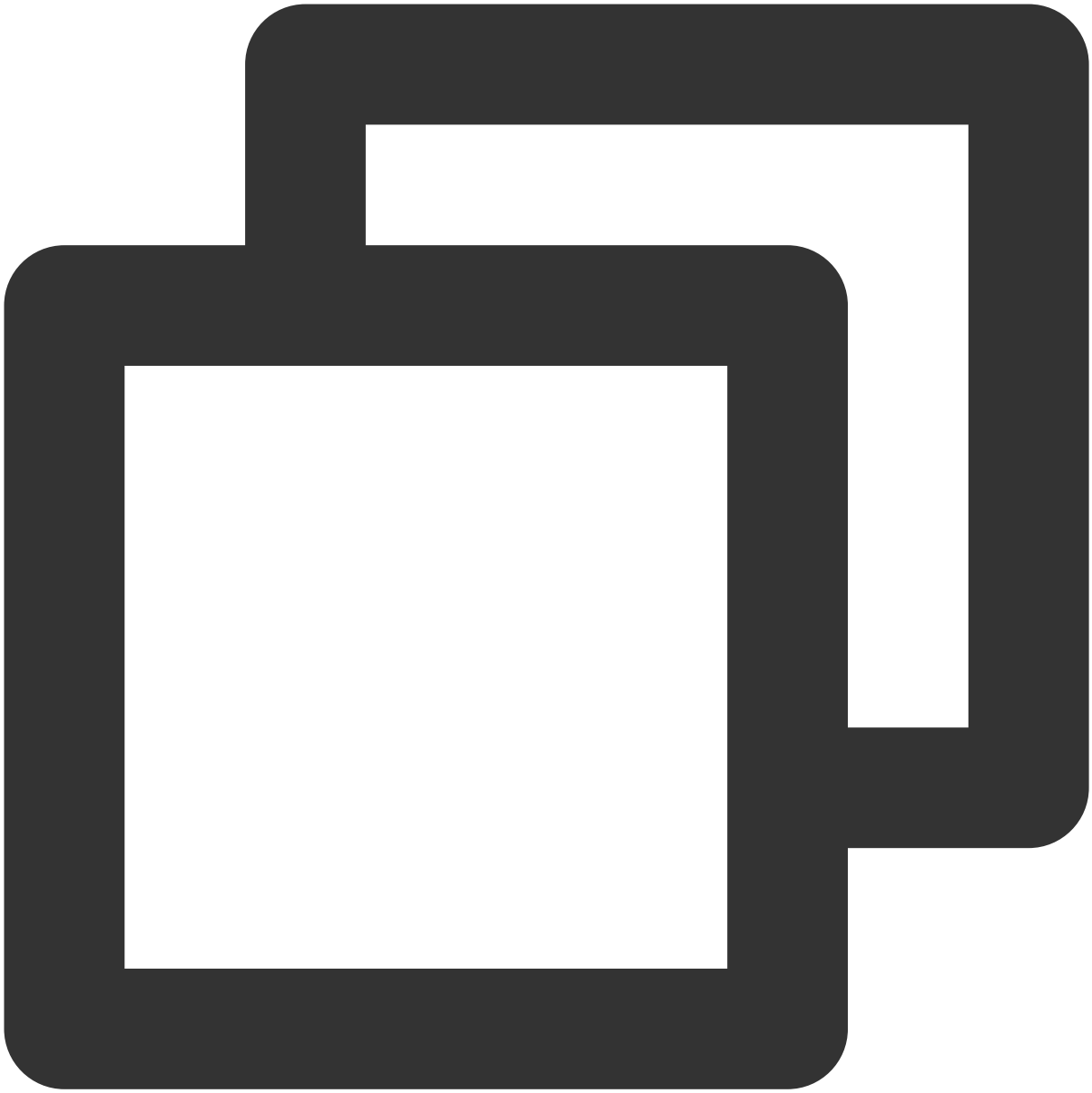
// 3. 调用 upload 方法上传文件
final COSUploadTask uploadTask = transferService.upload(putObjectRequest);
uploadTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        // 上传成功，可以在这里拿到文件的 CRC64
        String crc64 = result.getHeader("x-cos-hash-crc64ecma");
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

下载请求示例



```
// 1. 初始化 TransferService。在相同配置的情况下，您应该复用同一个 TransferService
TransferConfig transferConfig = new TransferConfig.Builder()
        .build();
TransferService transferService = new TransferService(cosXmlService, transferConfig

// 2. 初始化 GetObjectRequest
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String savePathDir = context.getCacheDir().toString(); //本地目录路径
//本地保存的文件名，若不填（null），则与 COS 上的文件名一样
```

```
String savedFileName = "exampleobject";
GetObjectRequest getObjectRequest = new GetObjectRequest(bucket,
        cosPath, savePathDir, savedFileName);

// 3. 调用 download 方法下载文件
final COSDownloadTask downloadTask = transferService.download(getObjectRequest);
downloadTask.setCosXmlResultListener(new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        // 下载成功，可以在这里拿到 COS 上的文件 CRC64
        String cosCRC64 = result.getHeader("x-cos-hash-crc64ecma");
    }

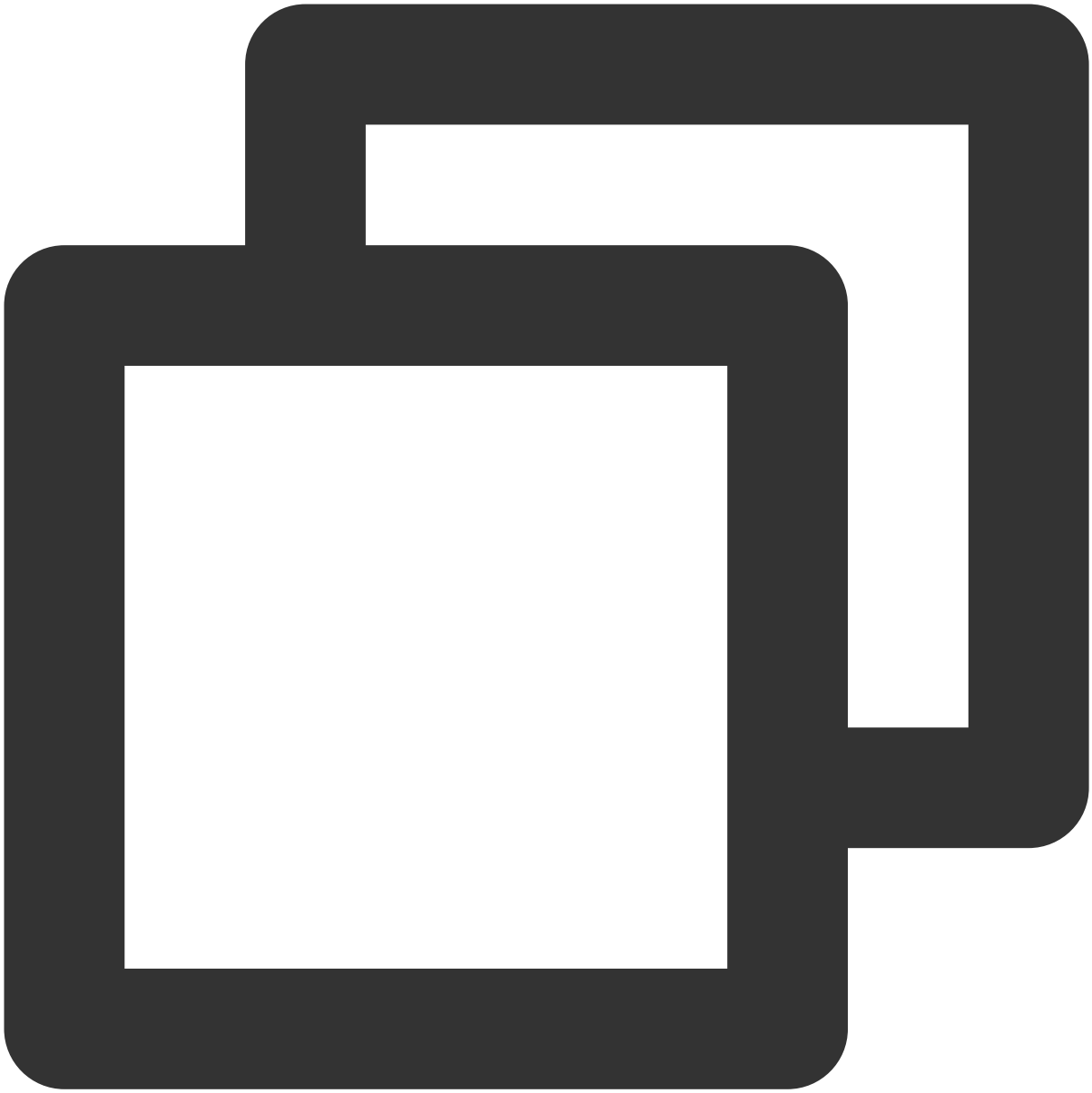
    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

CRC64 校验

通过 `TransferService` 进行上传和下载时，SDK 默认进行了数据校验的工作，如果您仍然希望能够自己进行 CRC64 校验，可以参考如下代码。



```
// 1. 参考以上上传或者下载请求示例代码获取 COS 上文件的 CRC64 值
String cosCRC64 = "examplecoscrc64";

// 2. 计算本地文件的 CRC64
File localFile = new File("examplefilepath");
String localCRC64 = DigestUtils.getCRC64String(localFile);

// 3. 比对 localCRC64 和 cosCRC64 是否一致
if (localCRC64.equals(cosCRC64)) {
    // CRC64 对比正确
}
```

说明

更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

图片处理
图片持久化处理

最近更新时间：2024-01-04 15:52:02

简介

腾讯云对象存储（Cloud Object Storage，COS）集成了 数据万象（Cloud Infinite，CI）专业的一体化多媒体解决方案，涵盖以下图片处理功能，详情可见 图片处理概述。

服务	功能	说明
基础图片处理服务	缩放	等比缩放、设定目标宽高缩放等多种方式
	裁剪	普通裁剪、缩放裁剪、内切圆、人脸智能裁剪
	旋转	自适应旋转、普通旋转
	格式转换	格式转换、GIF 格式优化、渐进显示
	质量变换	针对 JPG 和 WEBP 图片进行质量变换
	高斯模糊	对图片进行模糊处理
	锐化	对图片进行锐化处理
	添加水印	图片水印、文字水印
	获取图片信息	基本信息、EXIF 信息、主色调
	去除元信息	包括 EXIF 信息
	快速缩略模板	快速实现图片格式转换、缩略、剪裁等功能，生成缩略图
	样式设置	设置图片的样式，方便管理不同需求的图片

SDK API 参考

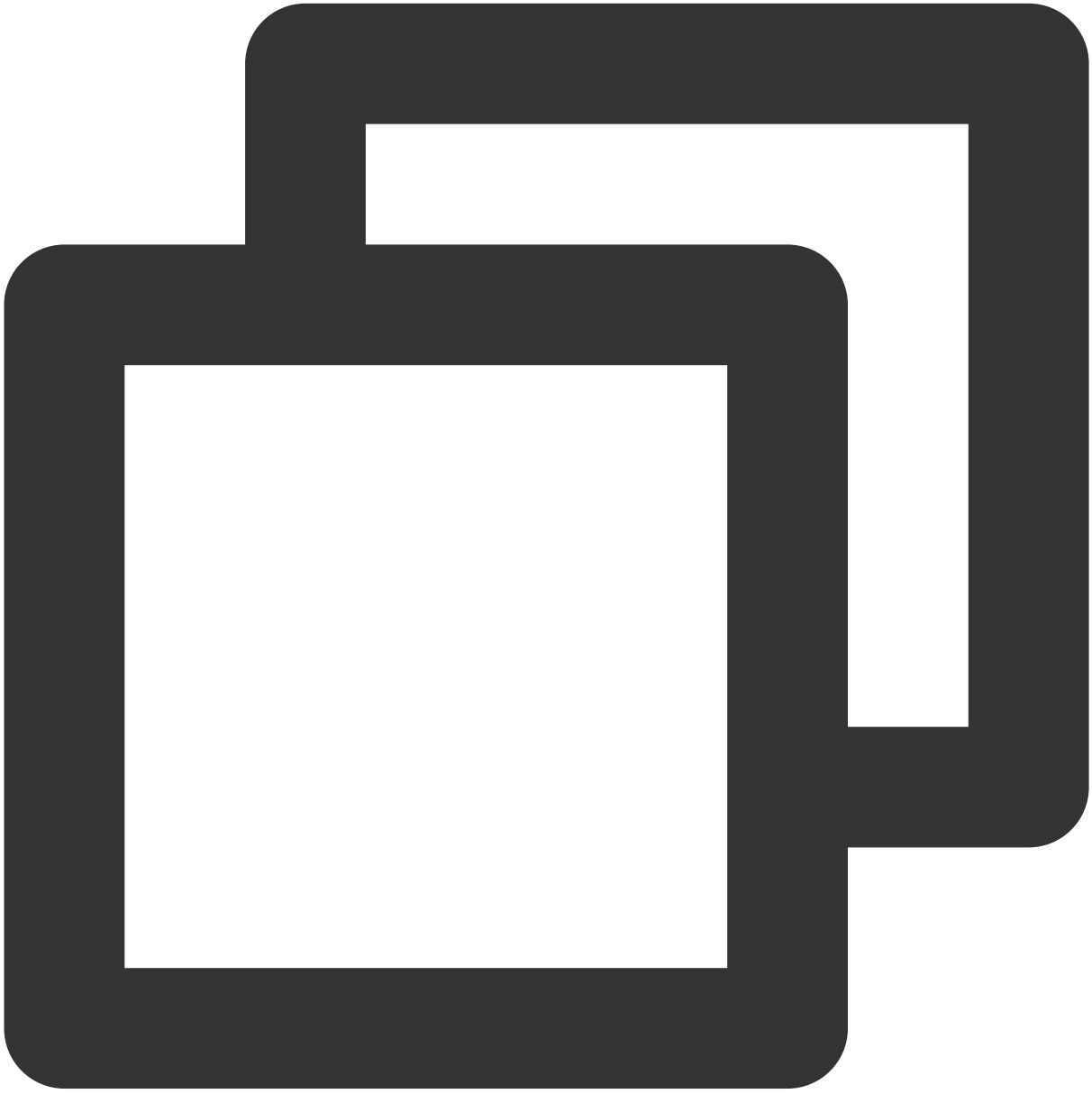
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

版权所有：腾讯云计算（北京）有限责任公司

上传时使用图片处理

下面示例展示了如何在上传图片时自动实现图片处理。
图片上传完成后，COS 会存储原始图片和已处理过的图片。后续用户可以通过普通的下载请求获取处理结果。

示例代码



```
List<PicOperationRule> rules = new LinkedList<>();
// 添加一条将图片转化为 png 格式的 rule，处理后的图片在存储桶中的位置标识符为
// examplepngobject
```

版权所有：腾讯云计算（北京）有限责任公司

```
rules.add(new PicOperationRule("examplepngobject", "imageView2/format/png"));
PicOperations picOperations = new PicOperations(true, rules);

PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, srcPath);
putObjectRequest.setPicOperations(picOperations);

// 上传成功后，您将会得到 2 张图片，分别是原始图片和处理后图片
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(putObjectRequest, upload
```

说明

更多完整示例，请前往 GitHub 查看。

基础图片处理

最近更新时间：2024-01-04 15:52:02

简介

腾讯云对象存储（Cloud Object Storage，COS）集成了 数据万象（Cloud Infinite，CI）专业的一体化多媒体解决方案，涵盖以下图片处理功能，详情可见 图片处理概述。

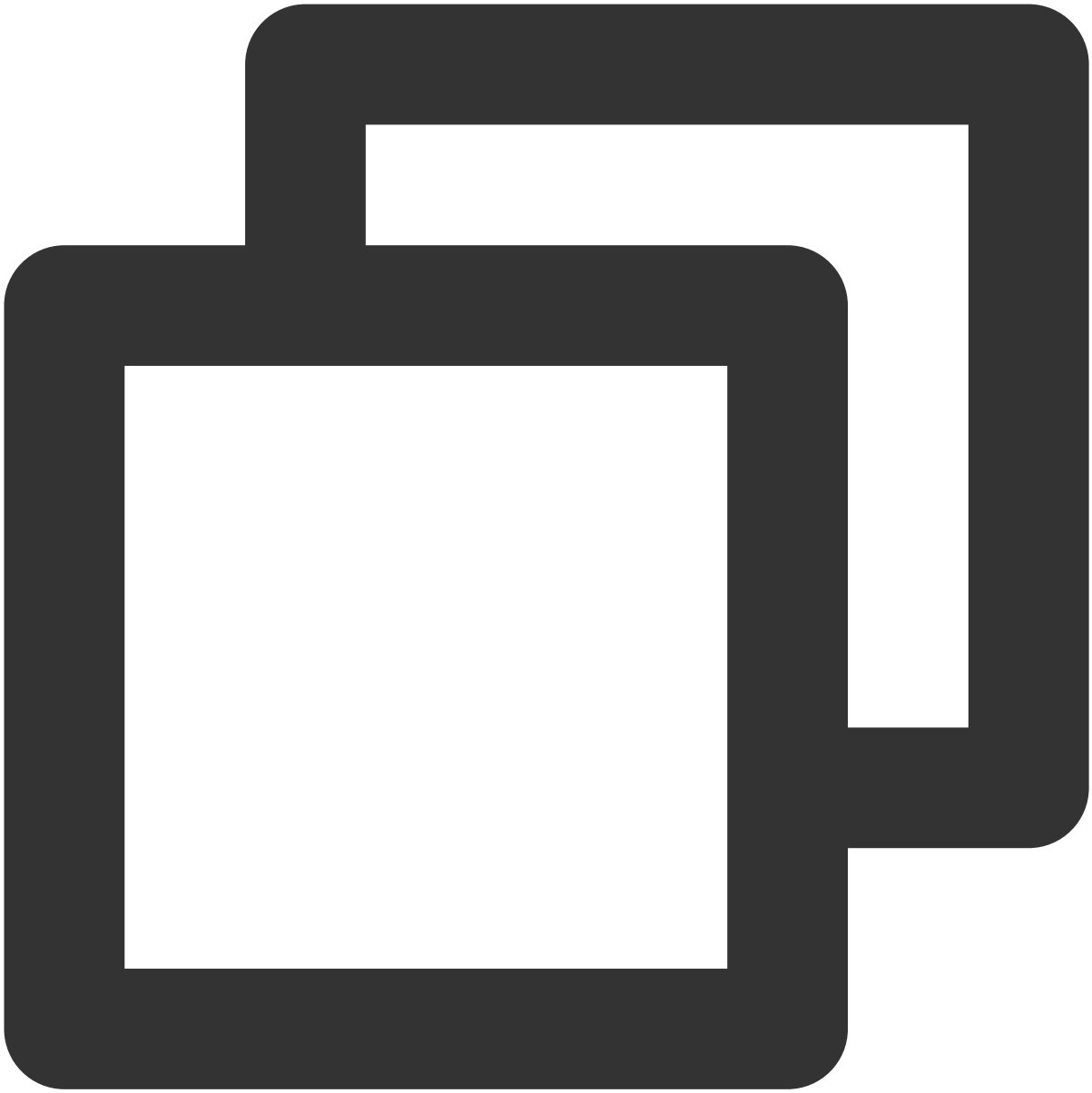
服务	功能	说明
基础图片处理服务	缩放	等比缩放、设定目标宽高缩放等多种方式
	裁剪	普通裁剪、缩放裁剪、内切圆、人脸智能裁剪
	旋转	自适应旋转、普通旋转
	格式转换	格式转换、GIF 格式优化、渐进显示
	质量变换	针对 JPG 和 WEBP 图片进行质量变换
	高斯模糊	对图片进行模糊处理
	锐化	对图片进行锐化处理
	添加水印	图片水印、文字水印
	获取图片信息	基本信息、EXIF 信息、主色调
	去除元信息	包括 EXIF 信息
	快速缩略模板	快速实现图片格式转换、缩略、剪裁等功能，生成缩略图

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

缩放

示例代码



```
String bucket = "examplebucket-1250000000"; //存储桶名称，格式：BucketName-APPID
String cosPath = "exampleobject"; //对象位于存储桶中的位置标识符，即对象键
String savePath = context.getExternalCacheDir().toString(); //本地路径

GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, cosPath,
        savePath);
getObjectRequest.addQuery("imageMogr2/thumbnail/!50p", null);

cosXmlService.getObjectAsync(getObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest,
```

```
                            CosXmlResult cosXmlResult) {
        GetObjectResult getObjectResult = (GetObjectResult) cosXmlResult;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

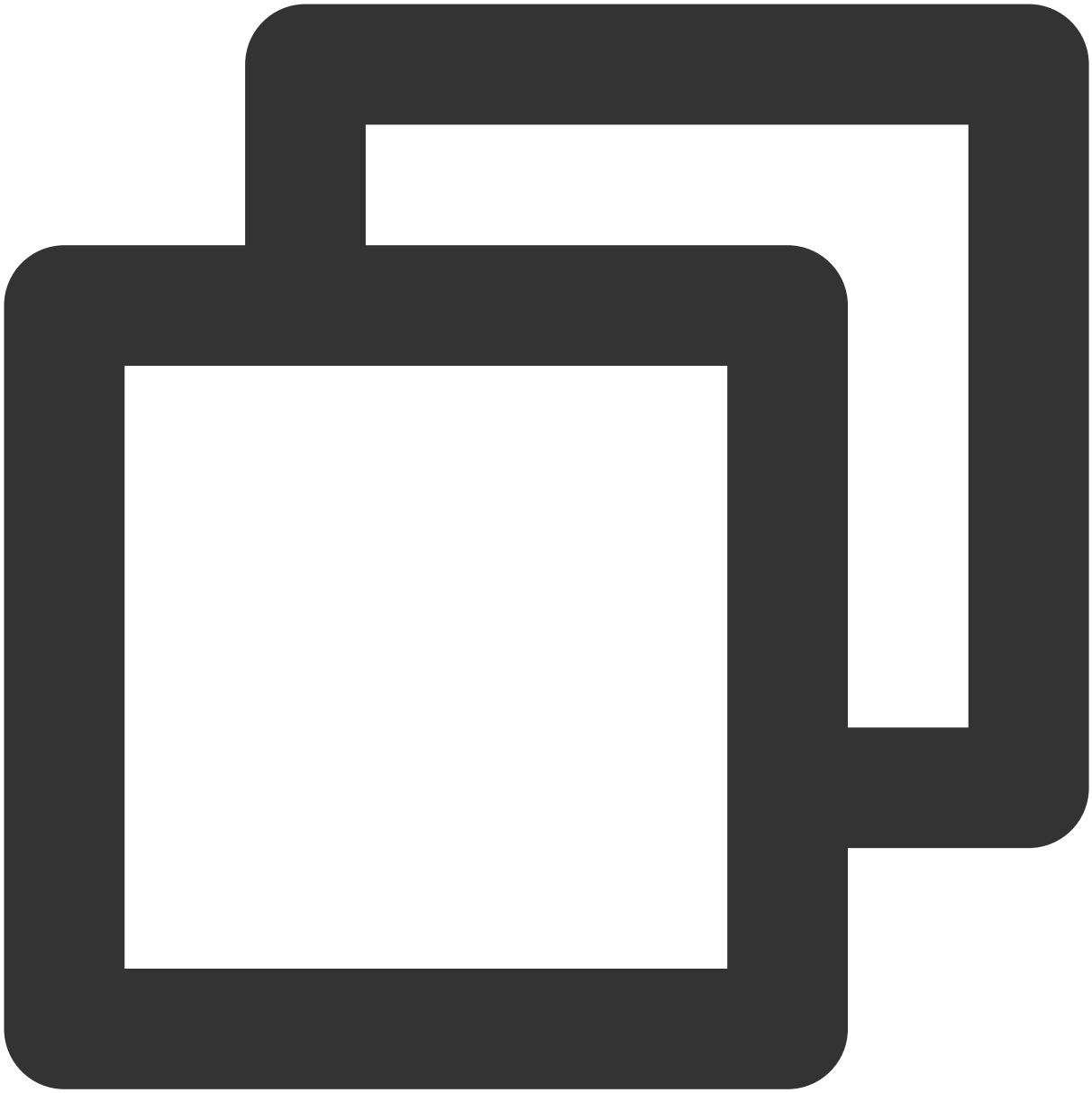
说明

更多完整示例，请前往 GitHub 查看。

裁剪

下面示例展示了如何在对已存储在 COS 的图片进行相应处理操作，并将结果存入到 COS。

示例代码



```
String bucket = "examplebucket-1250000000"; //存储桶名称，格式：BucketName-APPID
String cosPath = "exampleobject"; //对象位于存储桶中的位置标识符，即对象键
String savePath = context.getExternalCacheDir().toString(); //本地路径

GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, cosPath,
        savePath);
getObjectRequest.addQuery("imageMogr2/iradius/150", null);

cosXmlService.getObjectAsync(getObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest,
```

```
                            CosXmlResult cosXmlResult) {
        GetObjectResult getObjectResult = (GetObjectResult) cosXmlResult;
    }

    // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
    // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
    @Override
    public void onFail(CosXmlRequest cosXmlRequest,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

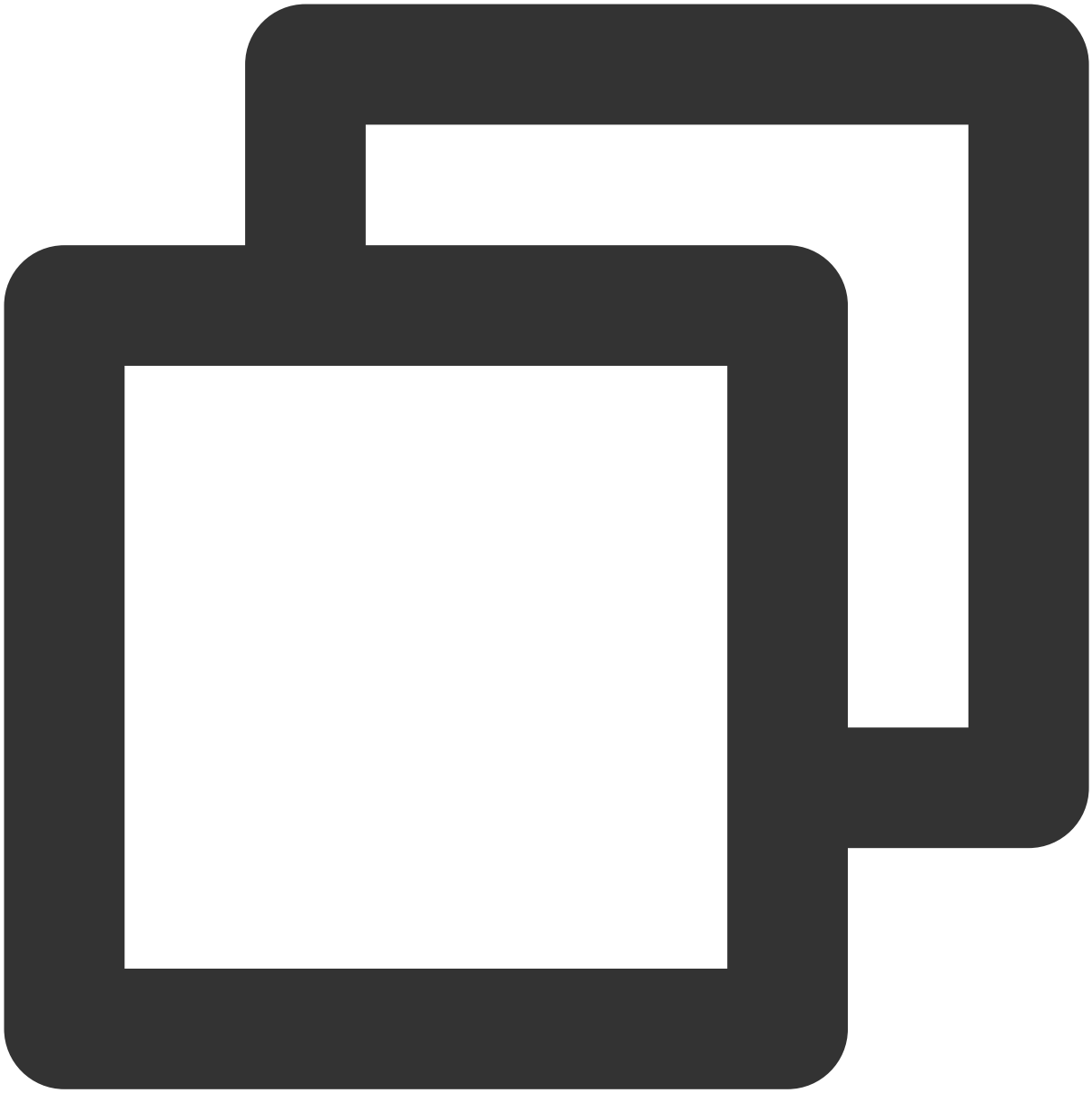
说明

更多完整示例，请前往 GitHub 查看。

旋转

下面示例展示了如何在对已存储在 COS 的图片在下载时进行处理操作。

示例代码



```
String bucket = "examplebucket-1250000000"; //存储桶名称，格式：BucketName-APPID
String cosPath = "exampleobject"; //对象位于存储桶中的位置标识符，即对象键
String savePath = context.getExternalCacheDir().toString(); //本地路径

GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, cosPath,
        savePath);
getObjectRequest.addQuery("imageMogr2/rotate/90", null);

cosXmlService.getObjectAsync(getObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest,
```

```
                            CosXmlResult cosXmlResult) {
            GetObjectResult getObjectResult = (GetObjectResult) cosXmlResult;
        }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest cosXmlRequest,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

图片高级压缩

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于图片高级压缩的 API 概览以及 SDK 示例代码。

API	操作描述
图片高级压缩	对指定存储桶下的图片进行压缩

SDK API 参考

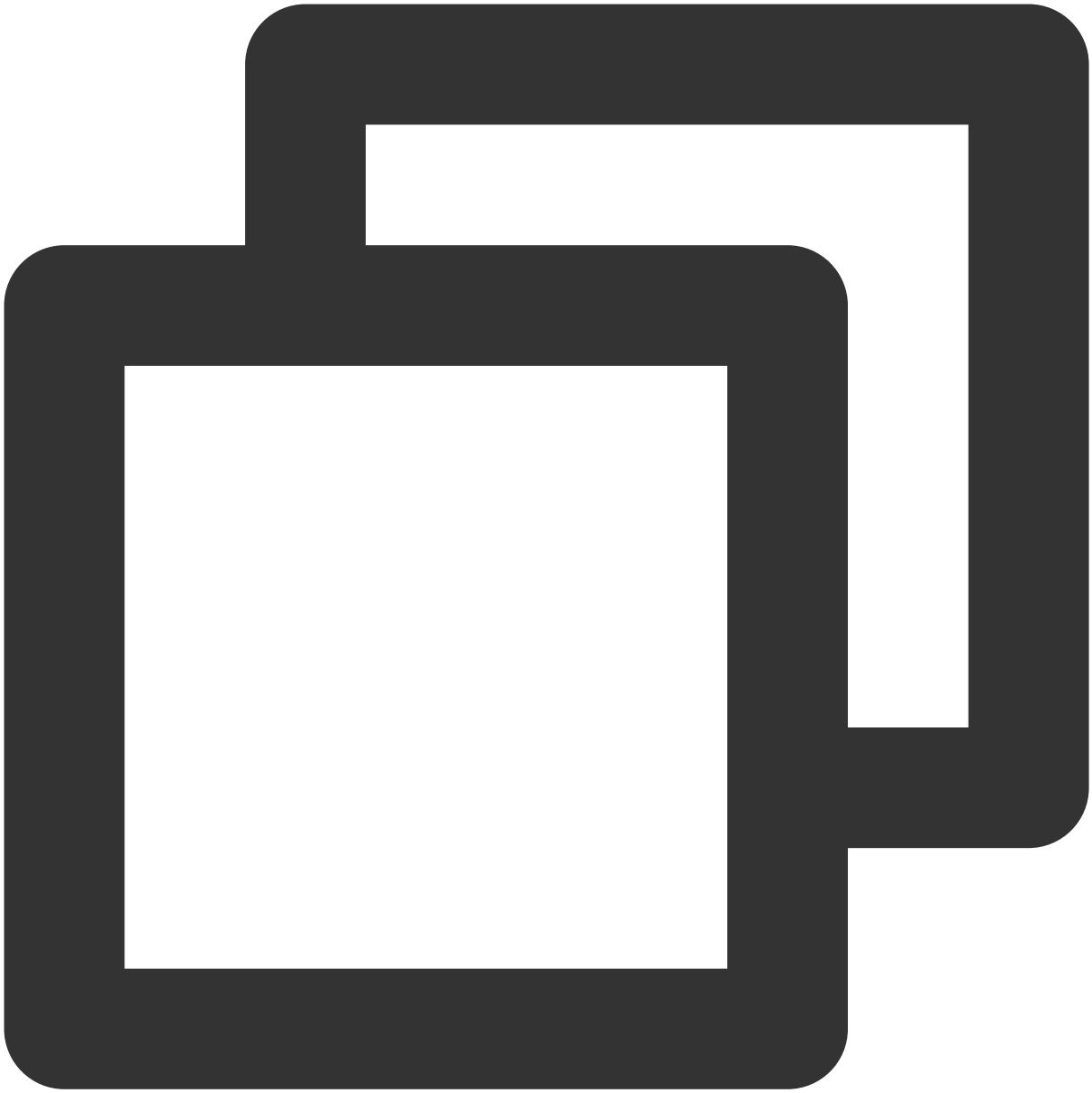
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

图片高级压缩

功能说明

图片高级压缩可以更加高效地将图片转换为 TPG 或 HEIF 等高压缩比格式，有效降低图片传输链路及加载耗时，降低带宽及流量成本。

示例代码：下载时进行高级压缩



```
String bucket = "examplebucket-1250000000"; //存储桶名称，格式：BucketName-APPID
String cosPath = "exampleobject"; //对象位于存储桶中的位置标识符，即对象键
String savePath = context.getExternalCacheDir().toString(); //本地路径

GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, cosPath,
        savePath);
getObjectRequest.addQuery("imageMogr2/format/tpg", null);

cosXmlService.getObjectAsync(getObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest,
```

```
                            CosXmlResult cosXmlResult) {
            GetObjectResult getObjectResult = (GetObjectResult) cosXmlResult;
        }

        // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFail 方法，
        // clientException 的类型为 CosXmlClientException?, serviceException 的类型为 CosX
        @Override
        public void onFail(CosXmlRequest cosXmlRequest,
                           @Nullable CosXmlClientException clientException,
                           @Nullable CosXmlServiceException serviceException) {
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

盲水印

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于盲水印的 API 概览以及 SDK 示例代码。

API	操作描述
盲水印	对本地图片添加或提取盲水印并上传至存储桶

SDK API 参考

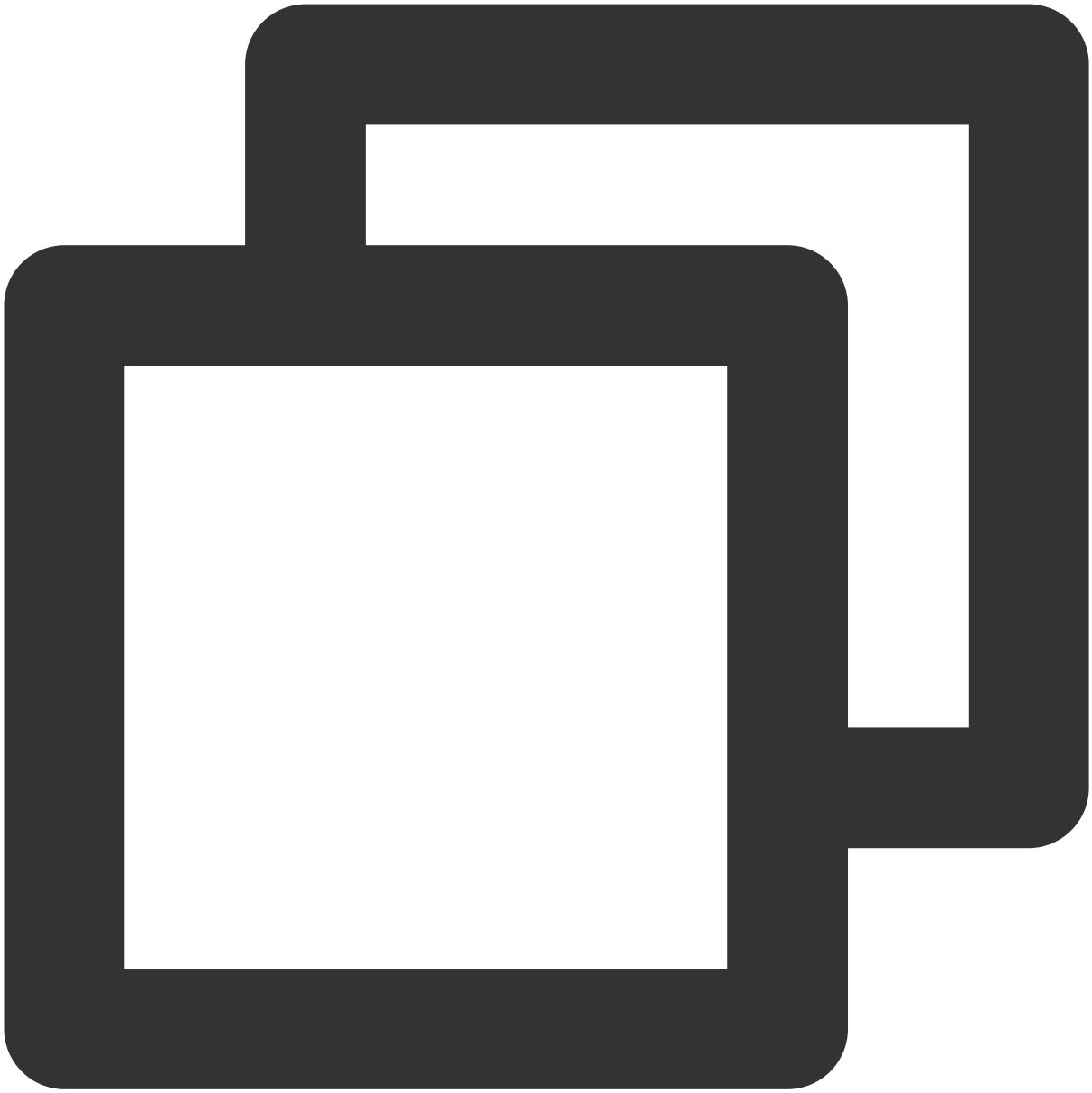
SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

添加盲水印

功能说明

盲水印支持在上传时添加以及下载时添加。

示例代码一：上传时添加盲水印



```
List<PicOperationRule> rules = new LinkedList<>();
// 添加一条将盲水印 rule，处理后的图片在存储桶中的位置标识符为
// examplewatermarkobject
rules.add(new PicOperationRule("examplewatermarkobject",
        "watermark/3/type/1/image/aHR0cDovL2V4YW1wbGVzLTEyNTEwMDAw"));
PicOperations picOperations = new PicOperations(true, rules);

PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, srcPath);
putObjectRequest.setPicOperations(picOperations);

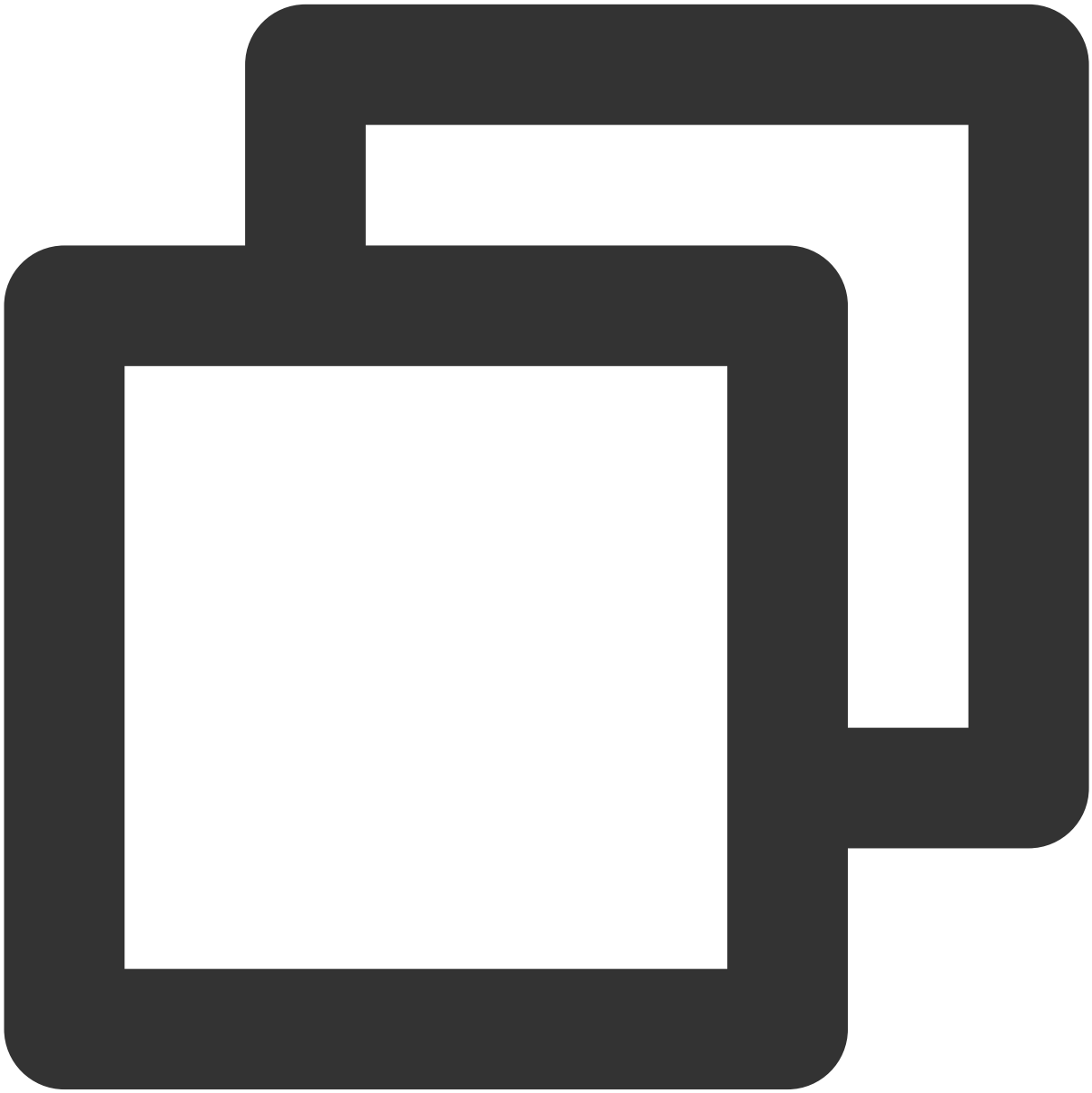
// 上传成功后，您将会得到 2 张图片，分别是原始图片和处理后图片
```

```
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(putObjectRequest, upload
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码二：下载时添加盲水印



```
GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, cosPath, savePathD
// 添加文字盲水印
getObjectRequest.addQuery("watermark/3/type/3/text/dGVuY2VudCBjbG91ZA==", null);
```

```
COSXMLDownloadTask cosxmlDownloadTask =
        transferManager.download(applicationContext, getObjectRequest);
```

说明

更多完整示例，请前往 GitHub 查看。

设置自定义头部

最近更新时间：2024-01-04 15:52:02

简介

本文档主要介绍 SDK 如何在请求时携带自定义头部。

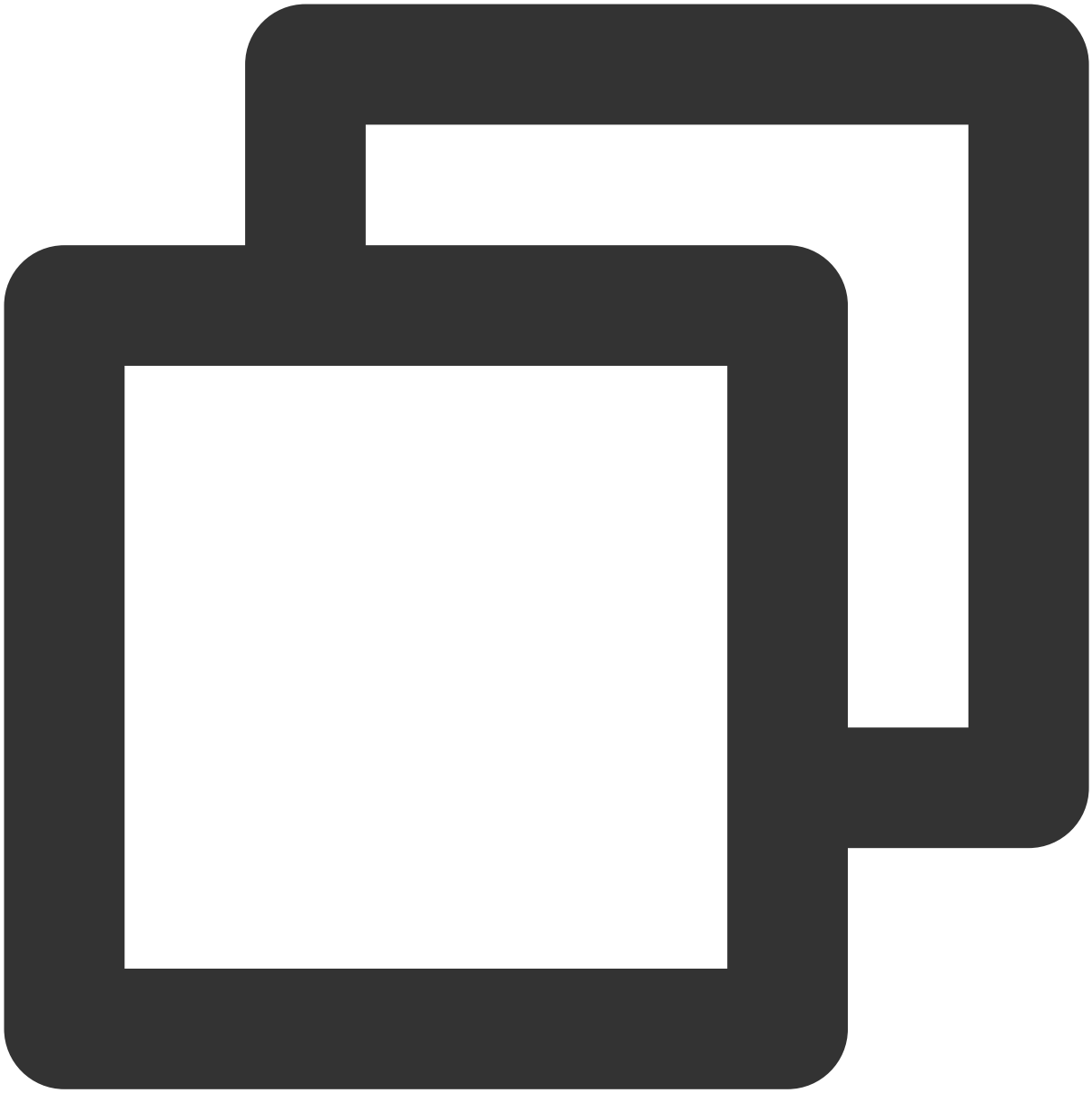
SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

功能说明

COS 在上传对象时可以携带以 `x-cos-meta-` 开头的自定义头部，包括用户自定义元数据头部后缀和用户自定义元数据信息，这些头部将作为对象元数据保存。
如果您开通了万象服务，可以携带 `Pic-Operations` 头部，实现后台自动图片处理，详细的 API 说明请参考 数据万象持久化。

示例代码



```
// 存储桶region可以在COS控制台指定存储桶的概览页查看 https://console.intl.cloud.tencent.cc
String region = "ap-beijing"; // 您的存储桶地域
String commonHeaderKey = "commonexamplekey"; // 自定义公共 Header 的键
String commonHeaderValue = "commonexamplevalue"; // 自定义公共 Header 的值
String requestHeaderKey = "requestexamplekey"; // 自定义请求 Header 的键
String requestHeaderValue = "requestexamplevalue"; // 自定义请求 Header 的值

CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig.Builder()
        .isHttps(true)
        .setRegion(region)
        .setDebuggable(false)
```

```
        // 给所有的请求添加公共的自定义 Header
        .addHeader(commonHeaderKey, commonHeaderValue)
        .builder();

CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig,
        credentialProvider);

// 给单个请求添加自定义 Header，优先级比公共 Header 更高
HeadObjectRequest headObjectRequest = new HeadObjectRequest(bucket, cosPath);
try {
    headObjectRequest.setRequestHeaders(requestHeaderKey, requestHeaderValue, false
} catch (CosXmlClientException e) {
    e.printStackTrace();
}

// 发起请求
cosXmlService.headObjectAsync(headObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        HeadObjectResult headObjectResult = (HeadObjectResult) result;
    }

    @Override
    public void onFail(CosXmlRequest request, CosXmlClientException clientException
                       CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

设置访问域名（CDN/全球加速）

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于如何使用非默认域名请求 COS 服务。

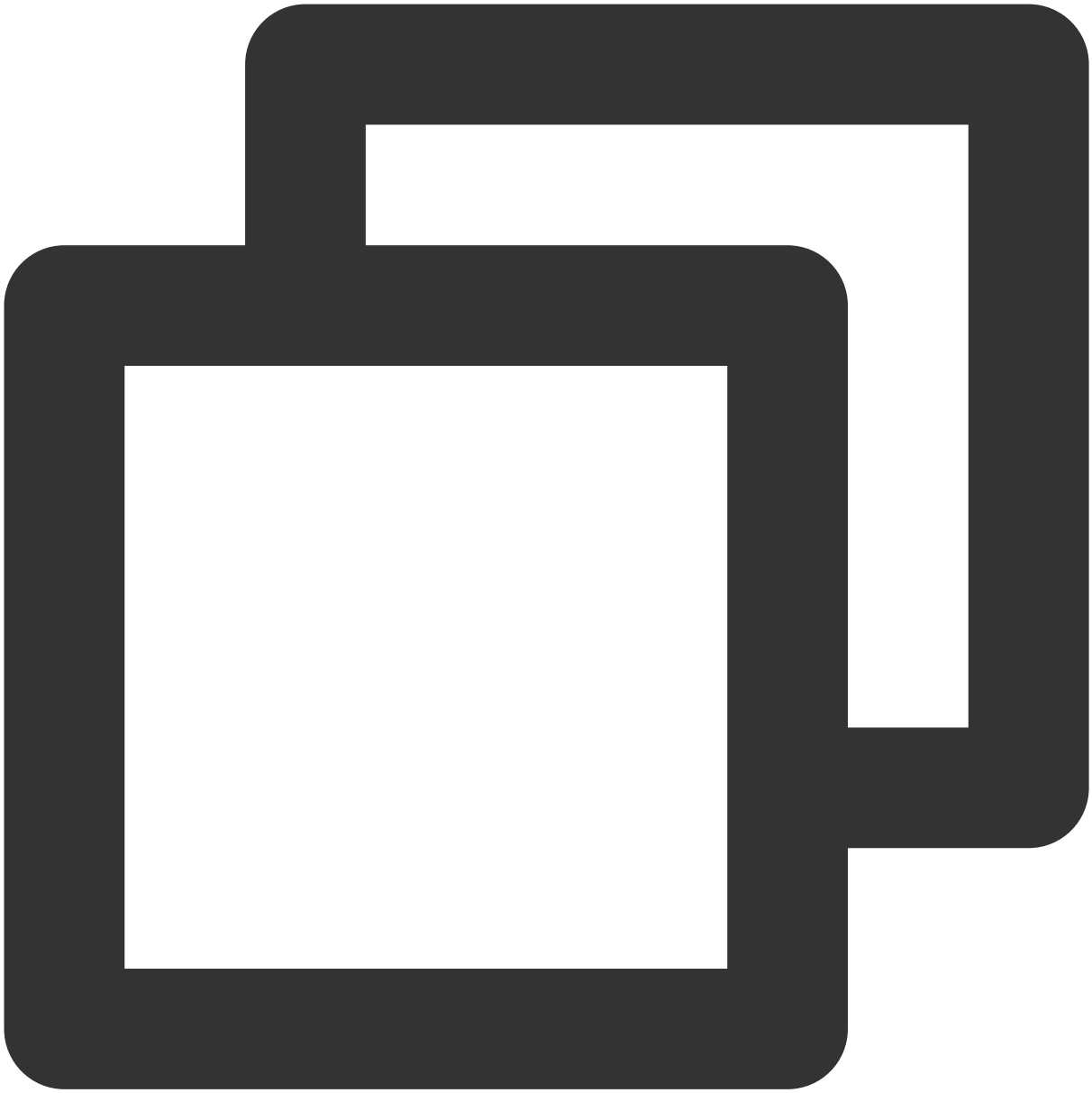
SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

CDN 默认加速域名

关于如何开启默认加速域名请参考 开启默认 CDN 加速域名。
以下代码展示了如何使用默认加速域名访问 COS 服务。

示例代码



```
// 存储桶region可以在COS控制台指定存储桶的概览页查看 https://console.intl.cloud.tencent.cc
String region = "ap-beijing"; // 您的存储桶地域
// 存储桶的默认加速域名
String cdnDomain = "examplebucket-1250000000.file.myqcloud.com";

CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig.Builder()
        .isHttps(true)
        .setRegion(region)
        .setDebuggable(false)
        .setHostFormat(cdnDomain) // 修改请求的域名
        .addHeader("Host", cdnDomain) // 修改 header 中的 host 字段
```

```
            .builder();

    // 不提供 credentialProvider 类，下载时可以通过给 url 添加 params 参数，来
    // 支持 CDN 权限校验
    CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig);
```

说明

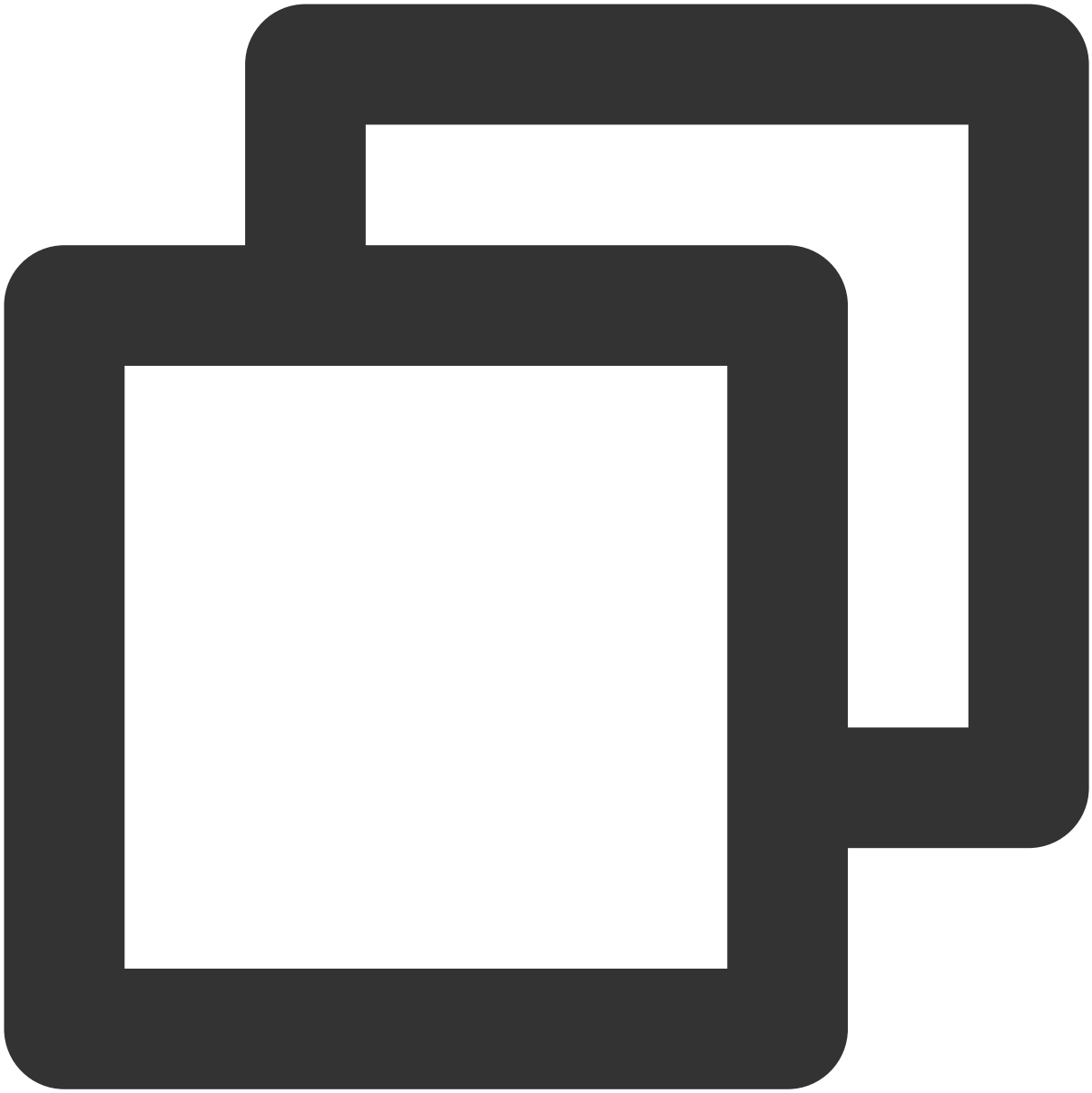
更多完整示例，请前往 GitHub 查看。

CDN 自定义加速域名

关于如何开启 CDN 自定义加速域名请参考 开启自定义 CDN 加速域名。

以下代码展示了如何使用自定义加速域名访问 COS 服务。

示例代码



```
// 存储桶region可以在COS控制台指定存储桶的概览页查看 https://console.intl.cloud.tencent.cc
String region = "ap-beijing"; // 您的存储桶地域
String cdnCustomDomain = "exampledomain.com"; // 自定义加速域名

CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig.Builder()
        .isHttps(true)
        .setRegion(region)
        .setDebuggable(false)
        .setHostFormat(cdnCustomDomain) // 修改请求的域名
        .addHeader("Host", cdnCustomDomain) // 修改 header 中的 host 字段
        .builder();
```

```
// 不提供 credentialProvider 类，下载时可以通过给 url 添加 params 参数，来
// 支持 CDN 权限校验
CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig);
```

说明

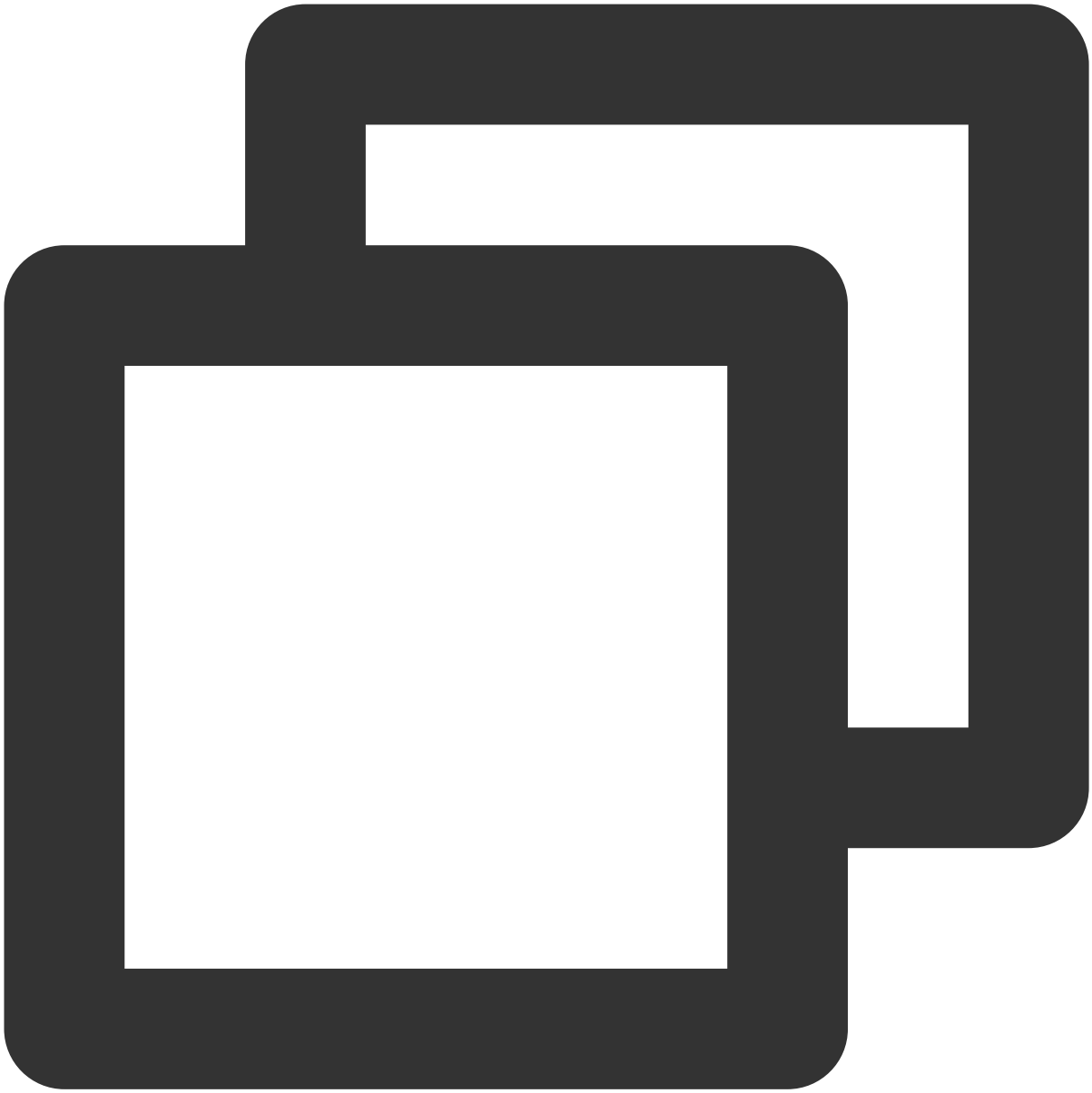
更多完整示例，请前往 GitHub 查看。

自定义源站域名

关于如何设置自定义源站域名请参考 自定义源站域名。

以下代码展示了如何使用自定义源站域名访问 COS 服务。

示例代码



```
// 存储桶region可以在COS控制台指定存储桶的概览页查看 https://console.intl.cloud.tencent.cc
String region = "ap-beijing"; // 您的存储桶地域
String customDomain = "exampledomain.com"; // 自定义域名

CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig.Builder()
        .isHttps(true)
        .setRegion(region)
        .setDebuggable(false)
        .setHostFormat(customDomain) // 修改请求的域名
        .builder();
```

```
CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig,
        credentialProvider);
```

说明

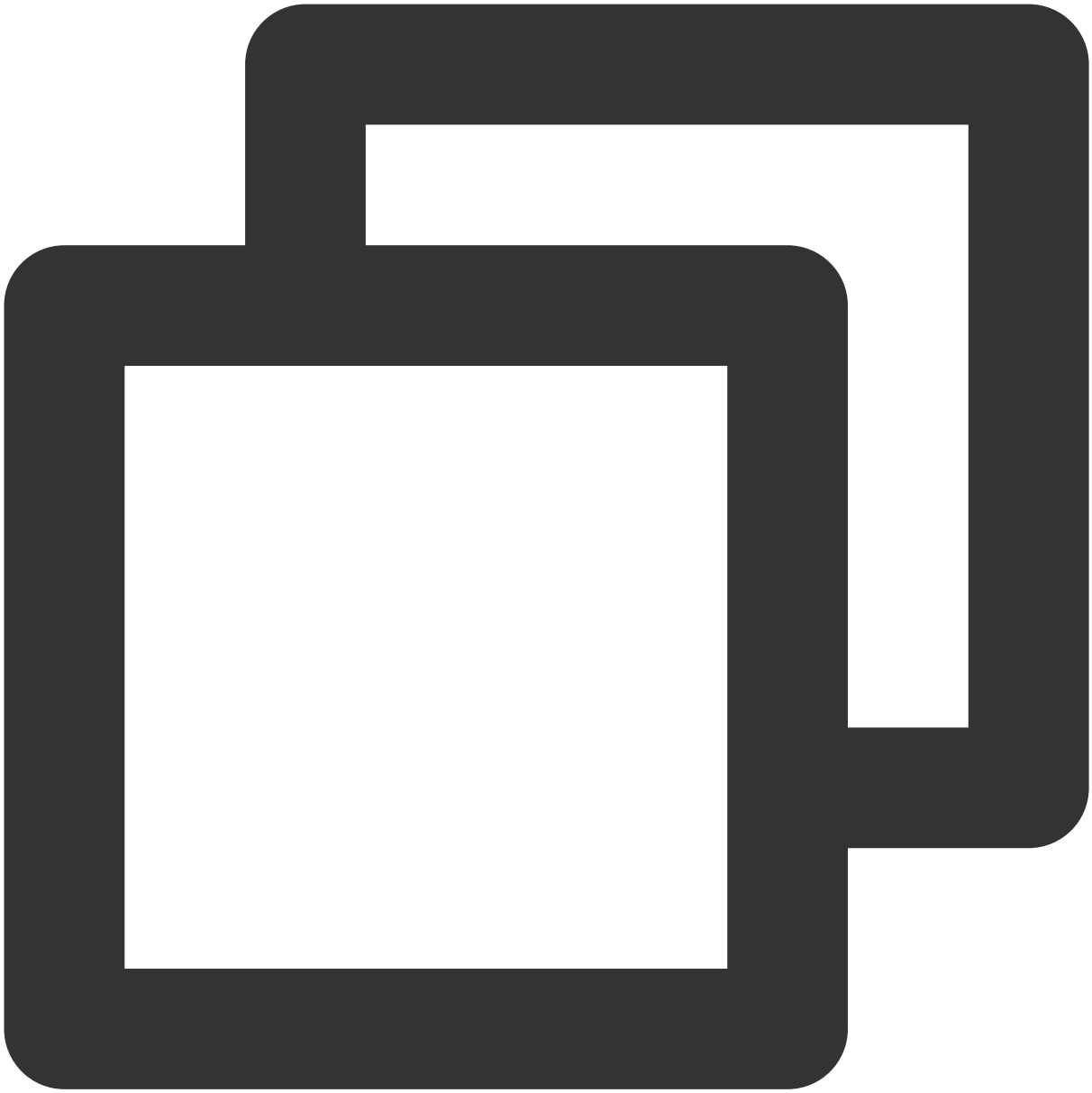
更多完整示例，请前往 GitHub 查看。

全球加速域名

关于全球加速功能请参考 全球加速功能概述。

以下代码展示了如何使用全球加速域名访问 COS 服务。

示例代码



```
// 存储桶region可以在COS控制台指定存储桶的概览页查看 https://console.intl.cloud.tencent.cc
String region = "ap-beijing"; // 您的存储桶地域

CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig.Builder()
        .isHttps(true)
        .setRegion(region)
        .setDebuggable(false)
        .setAccelerate(true) // 使能全球加速域名
        .builder();

CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig,
```

```
            credentialProvider);
```

说明

更多完整示例，请前往 GitHub 查看。

异常处理

最近更新时间：2024-01-04 15:52:02

简介

调用 SDK 接口请求 COS 服务失败时，系统将抛出 CosXmlClientException（客户端异常）或者 CosXmlServiceException （服务端异常）。
CosXmlClientException 是由于客户端无法和 COS 服务端正常进行交互所引起。如客户端无法连接到服务端，无法解析服务端返回的数据，读取本地文件发生 IO 异常等。
CosXmlServiceException 是客户端和 COS 服务端交互正常，但操作 COS 资源失败。如客户端访问一个不存在 Bucket ，删除一个不存在的文件，没有权限进行某个操作等。

客户端异常

CosClientException 集成自 Exception，使用方法同 Exception，同时添加一个额外的成员 errorCode，如下：

成员	描述	类型
errorCode	客户端错误码，如10000表示参数检验失败，更多详情请参见 SDK 错误码	int

服务端异常

CosXmlServiceException 例如客户端访问一个不存在 Bucket， 删除一个不存在的文件，没有权限进行某个操作，服务端故障异常等。CosXmlServiceException 包含了服务端返回的状态码， requestid， 出错明细等。捕获异常后，建议对整个异常进行打印， 异常包含了必须的排查因素。以下是异常成员变量的描述：

成员	描述	类型
requestId	请求 ID，用于表示一个请求，对于排查问题十分重要	string
statusCode	response 的 status 状态码，4xx 是指请求因客户端而失败，5xx 是服务端异常导致的失败，更多详情请参见 COS 错误信息	string
errorCode	请求失败时 body 返回的 Error Code，更多详情请参见 COS 错误信息	string
errorMessage	请求失败时 body 返回的 Error Message，更多详情请参见 COS 错误信息	string

版权所有：腾讯云计算（北京）有限责任公司

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**。
4. 稍侯片刻，便能看到相应的智能诊断结果。

版权所有：腾讯云计算（北京）有限责任公司

C SDK
快速入门

最近更新时间：2024-01-04 17:06:08

下载与安装

相关资源

对象存储的 XML C SDK 源码下载地址：XML C SDK。

演示示例 Demo 下载地址：XML C SDK Demo。

SDK 更新日志请参见 ChangeLog。

SDK 常见问题请参见：C SDK 常见问题。

说明

如果您在使用 SDK 时遇到函数或方法不存在等错误，请先将 SDK 升级到最新版再重试。

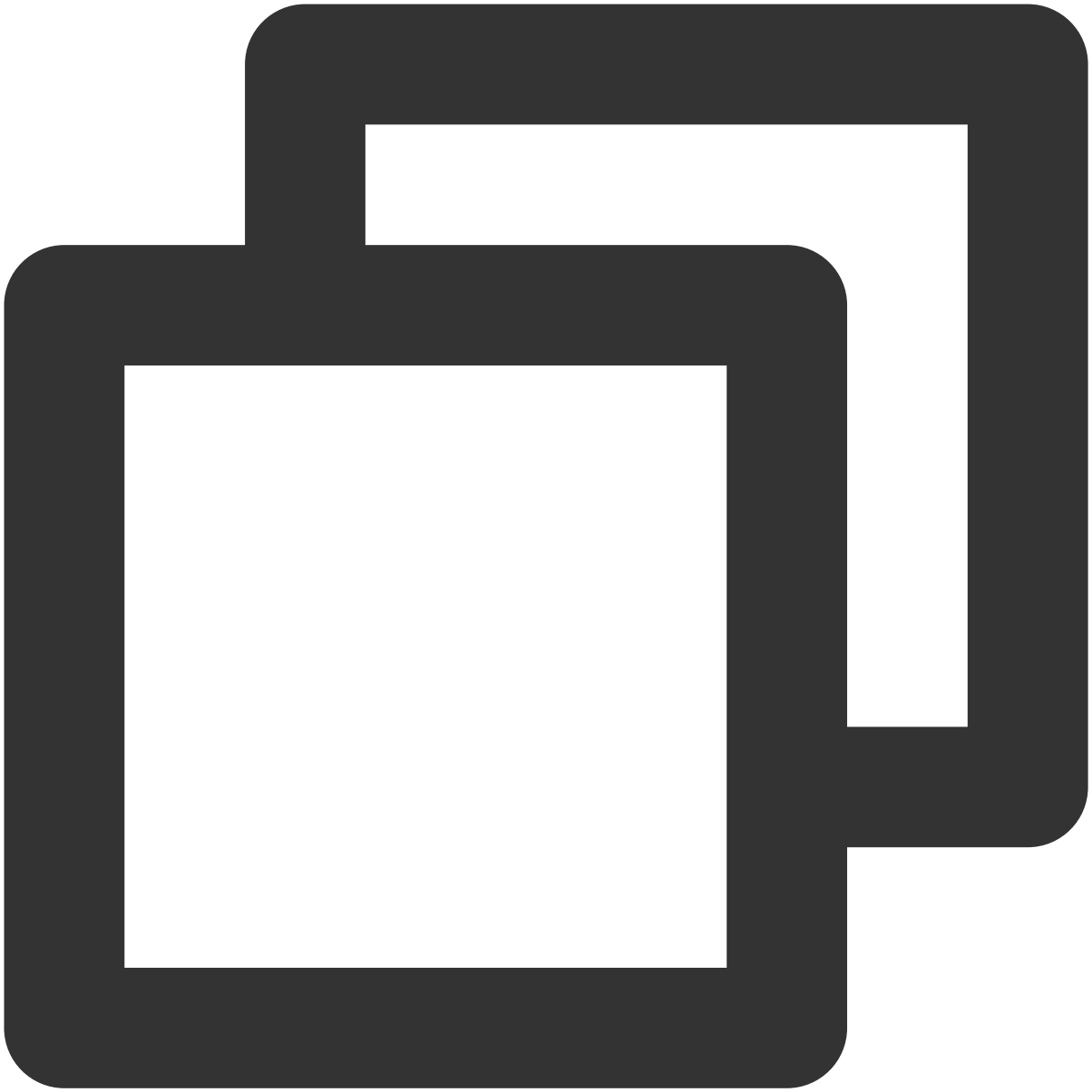
环境依赖

依赖库：libcurl apr apr-util minixml。

安装 SDK

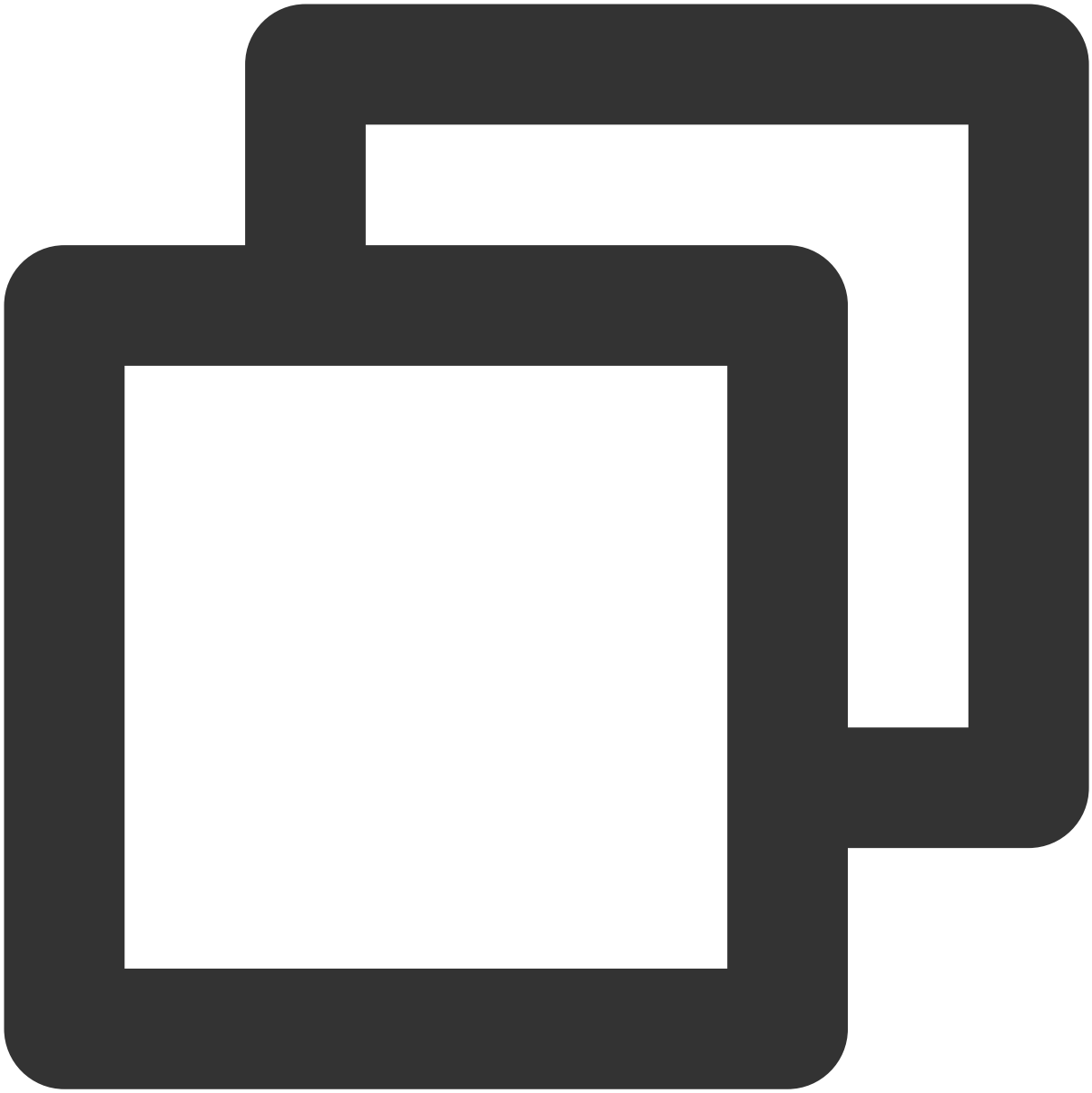
1. 安装 CMake 工具（建议 2.6.0 及以上版本），单击 这里 下载，安装方式如下：

版权所有：腾讯云计算（北京）有限责任公司



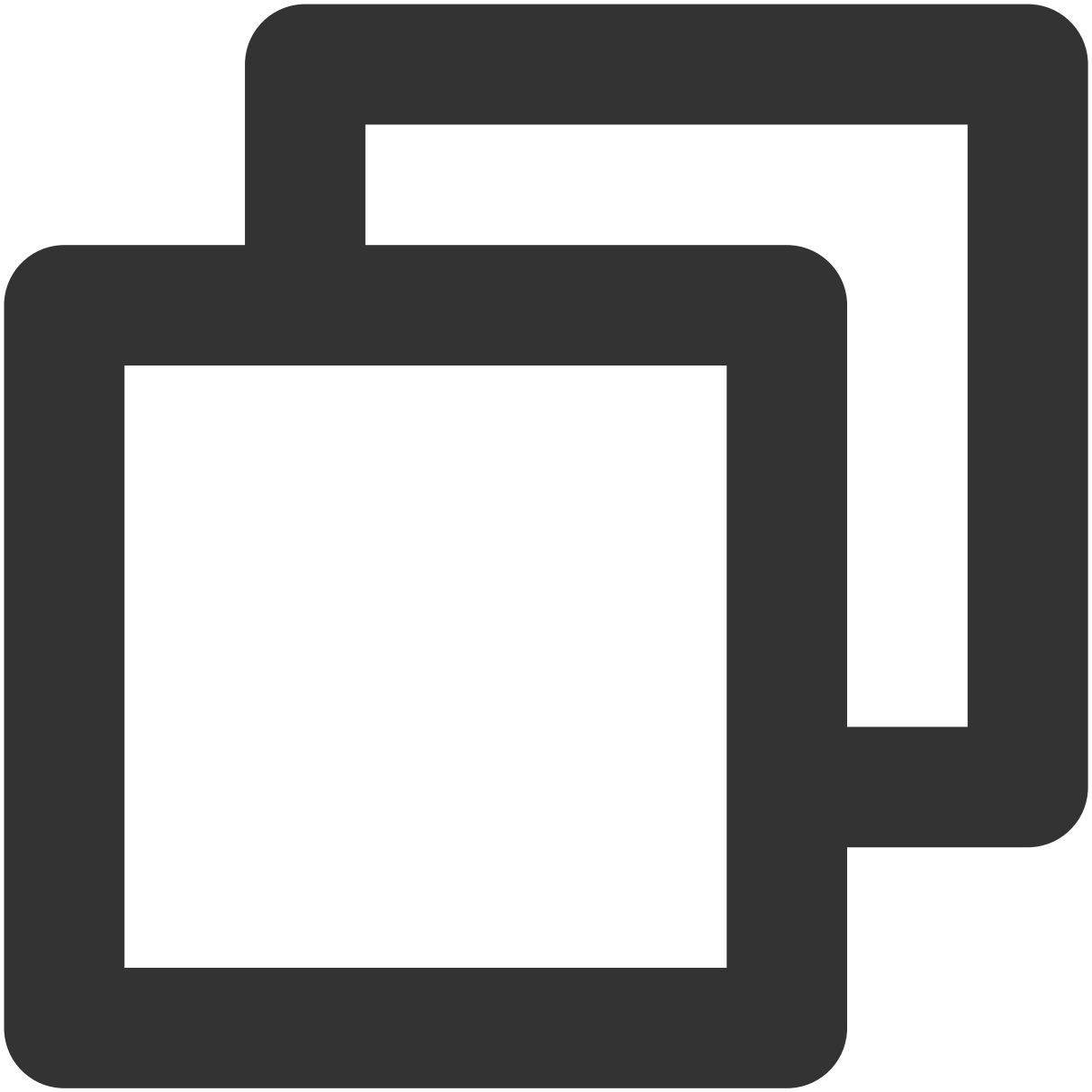
```
./configure
make
make install
```

2. 安装 libcurl（建议 7.32.0 及以上版本），单击 这里 下载，安装方式如下：



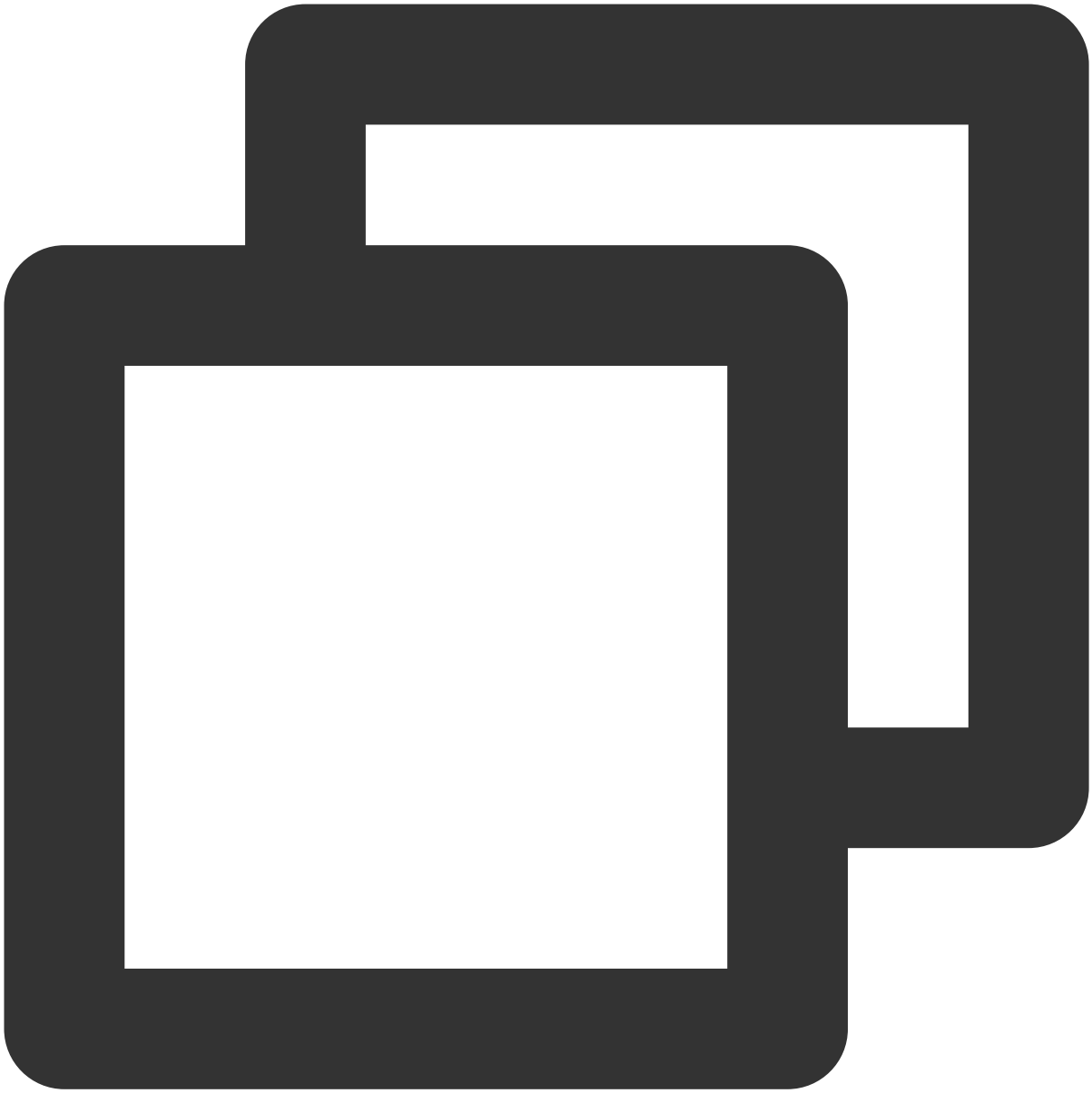
```
./configure
make
make install
```

3. 安装 apr（建议 1.5.2 及以上版本），单击 这里 下载，安装方式如下：



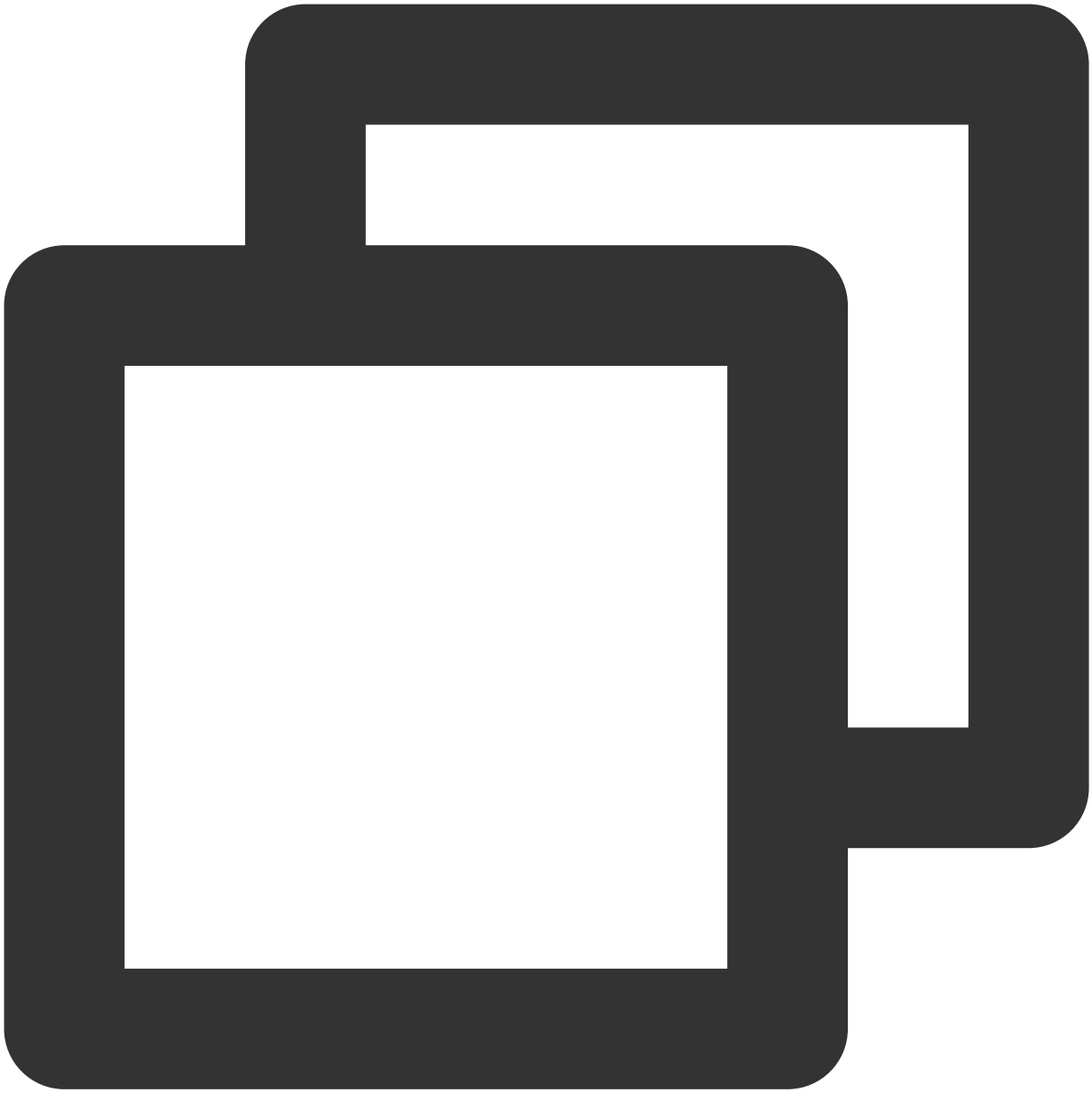
```
./configure
make
make install
```

4. 安装 apr-util（建议 1.5.4 及以上版本），单击 这里 下载，安装时需要指定 `--with-apr` 选项，安装方式如下：



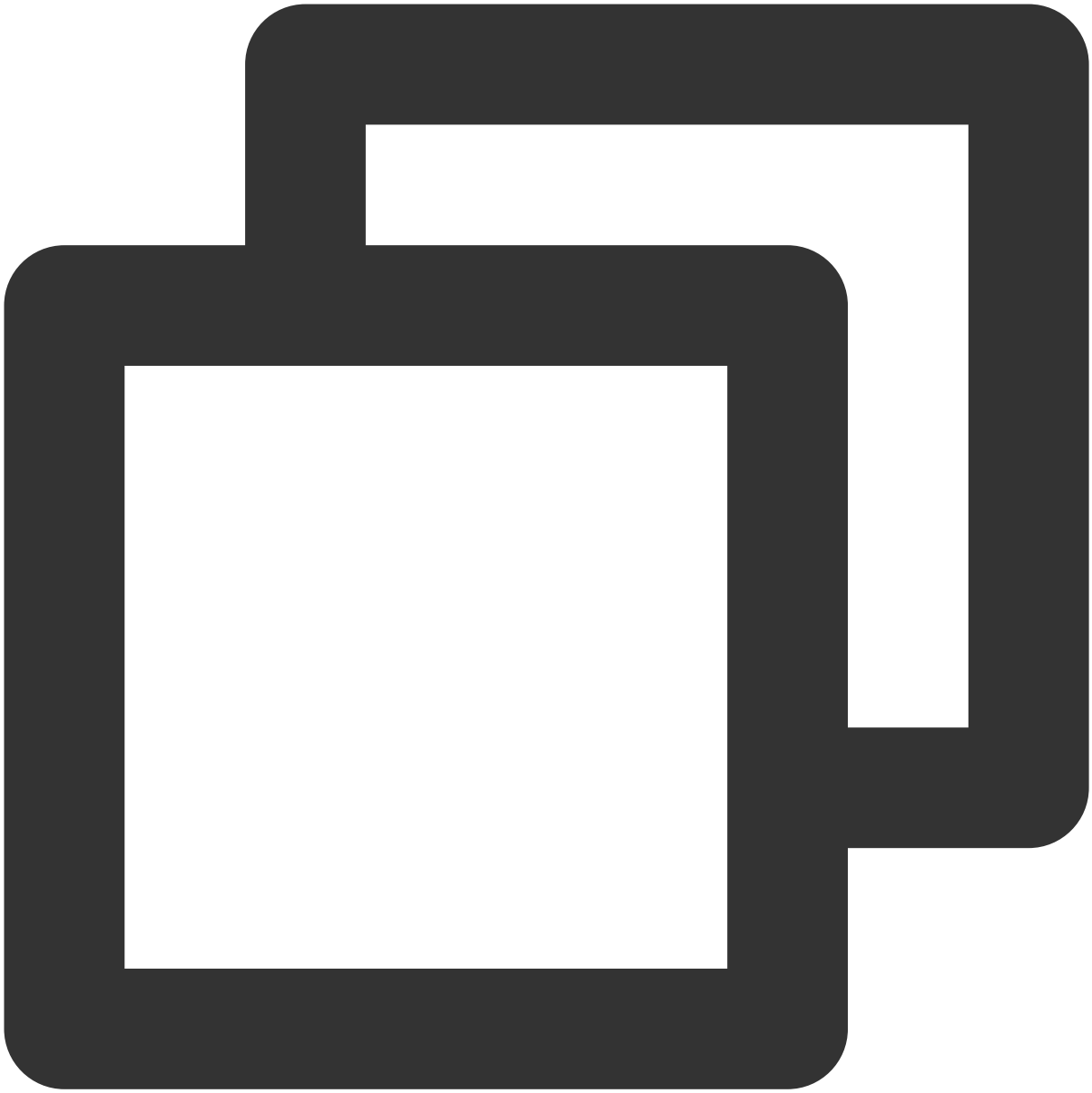
```
./configure --with-apr=/your/apr/install/path
make
make install
```

5. 安装 minixml（建议 2.8 - 2.12 版本），单击 这里 下载，安装方式如下：



```
./configure
make
make install
```

6. 编译 COS C SDK。下载 XML C SDK 源码，执行如下编译命令：



```
cmake .
make
make install
```

开始使用

下面为您介绍使用 XML C SDK 的一般流程。

1. 初始化 SDK。

2. 设置请求选项参数。关于 APPID、SecretId、SecretKey、Bucket 等名称的含义和获取方式请参见 COS 术语信息。

APPID 是申请腾讯云账号后，系统分配的账户标识之一。

access_key_id 与 access_key_secret 是账号 API 密钥。

endpoint 是 COS 访问域名信息，详情请参见 地域和访问域名 文档。例如，广州地域 endpoint 为 `cos.ap-guangzhou.myqcloud.com`，全球加速域名的 endpoint 为 `cos.accelerate.myqcloud.com`。endpoint可以加上http或者https，SDK 默认通过 http 访问 COS，如通过 https 访问广州地域的 endpoint 为 `https://cos.ap-guangzhou.myqcloud.com`。

is_cname 表示 endpoint 是否为自定义域名，如果is_cname设置成1，则 endpoint 表示自定义域名。

3. 设置 API 接口必需的参数。

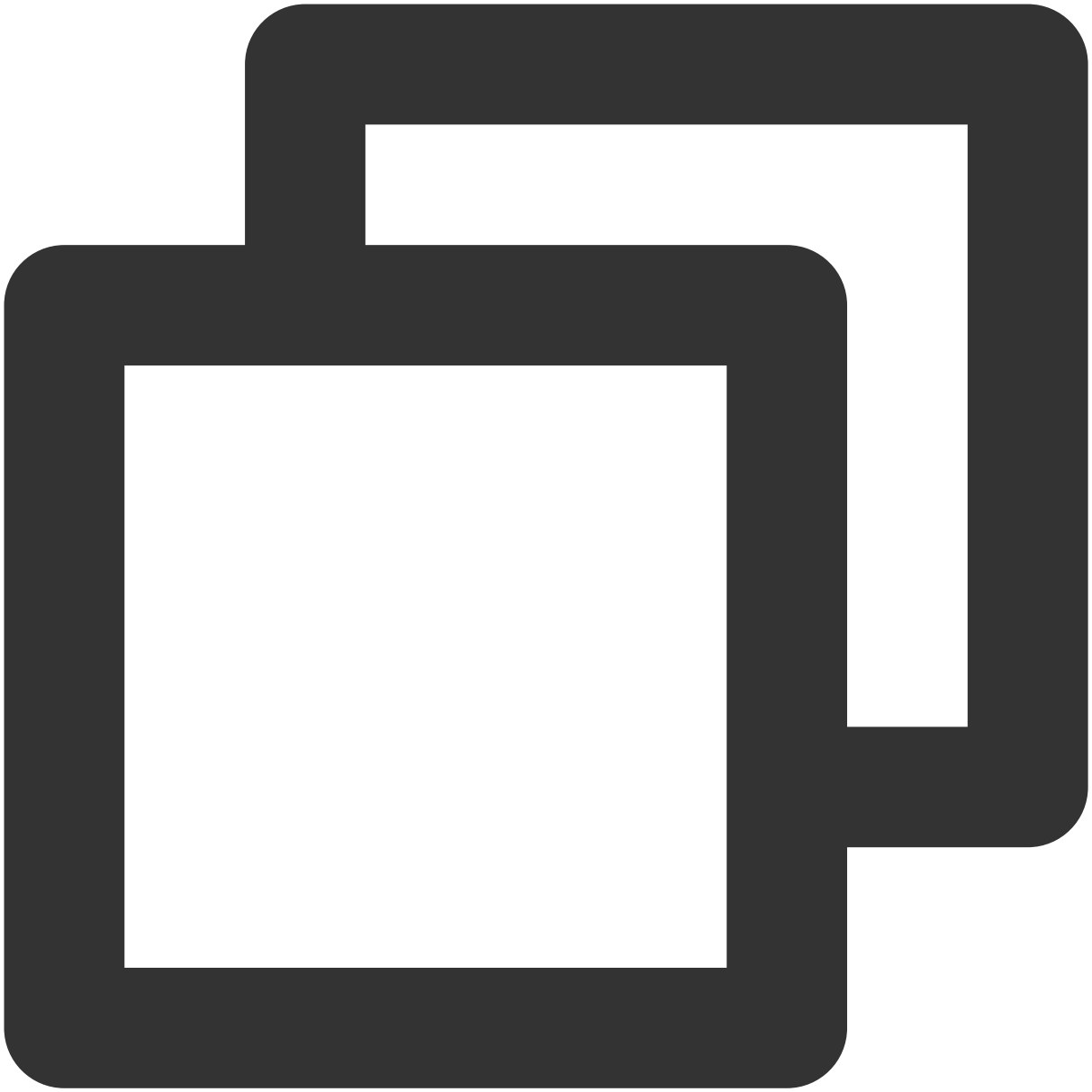
4. 调用 SDK API 发起请求并获得请求响应结果。

初始化

注意

建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。

如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。



```
int main(int argc, char *argv[])
{
    /* 程序入口处调用 cos_http_io_initialize 方法，这个方法内部会做一些全局资源的初始化，涉及
    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
        exit(1);
    }

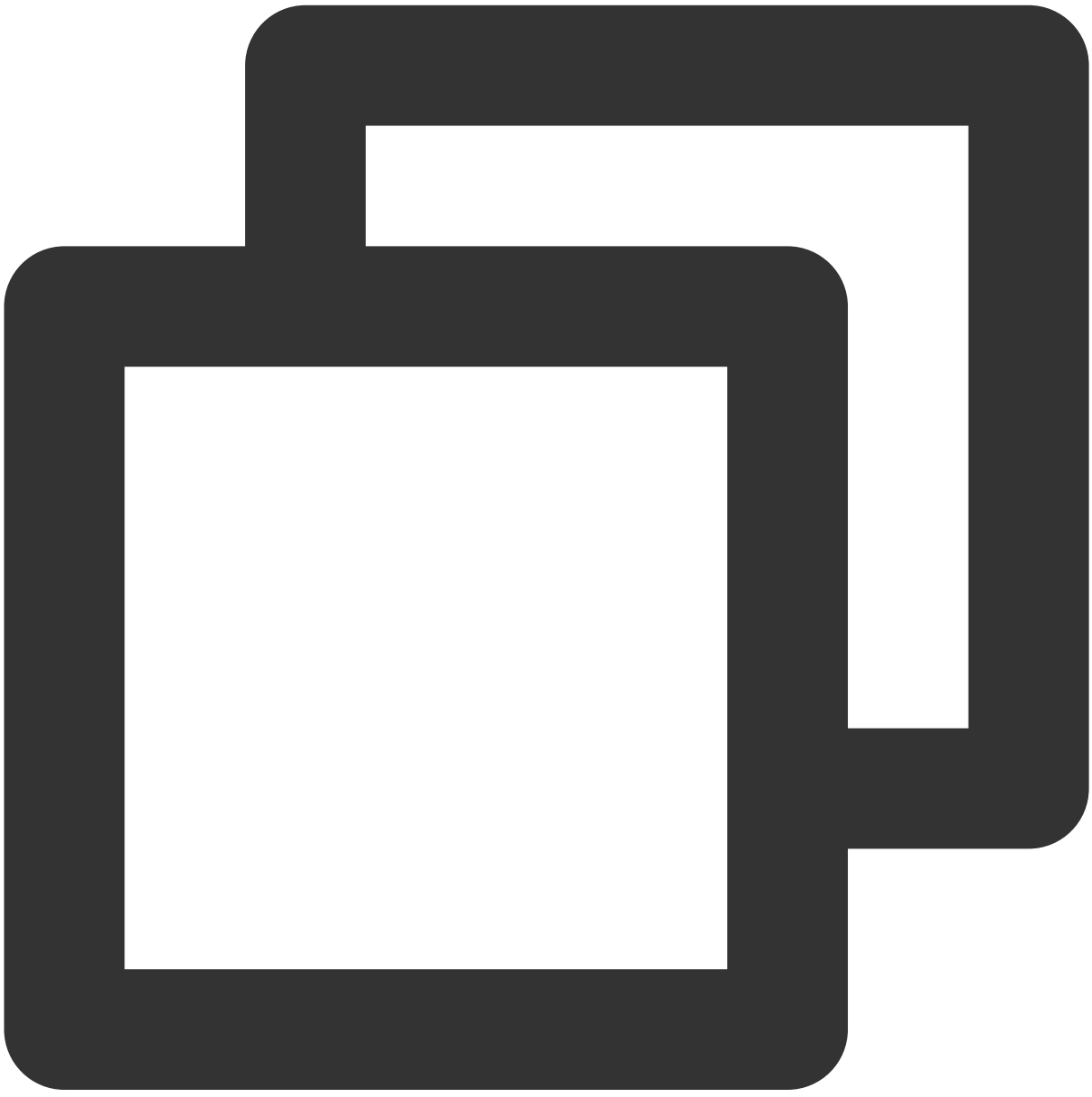
    /* 调用 COS SDK 的接口上传或下载文件 */
    /* ... 用户逻辑代码，这里省略 */

    /* 程序结束前，调用 cos_http_io_deinitialize 方法释放之前分配的全局资源 */
```

版权所有：腾讯云计算（北京）有限责任公司

```
    cos_http_io_deinitialize();
    return 0;
}
```

初始化请求选项



```
/* 等价于 apr_pool_t，用于内存管理的内存池，实现代码在 apr 库中 */
cos_pool_t *pool;
cos_request_options_t *options;
```

```
/* 重新创建一个新的内存池，第二个参数是 NULL，表示没有继承自其它内存池 */
cos_pool_create(&pool, NULL);

/* 创建并初始化 options，这个参数内部主要包括endpoint,access_key_id,acces_key_secret, is_
 * options的内存是由pool分配的，后续释放掉pool后，options的内存也相当于释放掉了，不再需要单独释
 */
options = cos_request_options_create(pool);
options->config = cos_config_create(options->pool);

/* cos_str_set 是用 char* 类型的字符串初始化 cos_string_t 类型*/
cos_str_set(&options->config->endpoint, "<用户的Endpoint>");                //Endpoint
cos_str_set(&options->config->access_key_id, "<用户的SecretId>");           //用户注册 (
cos_str_set(&options->config->access_key_secret, "<用户的SecretKey>");      //用户注册 (
cos_str_set(&options->config->appid, "<用户的AppId>");                      //用户注册 (

/* 可以通过设置 sts_token 来使用临时密钥，当使用临时密钥时，access_key_id 和access_key_secr
//cos_str_set(&options->config->sts_token, "MyTokenString");
/* 是否使用了 CNAME */
options->config->is_cname = 0;
/* 使用自定义域名访问 COS */
/*
options->config->is_cname = 1;
cos_str_set(&options->config->endpoint, "<自定义域名>");
*/

/* 用于设置网络相关参数，例如超时时间等*/
options->ctl = cos_http_controller_create(options->pool, 0);

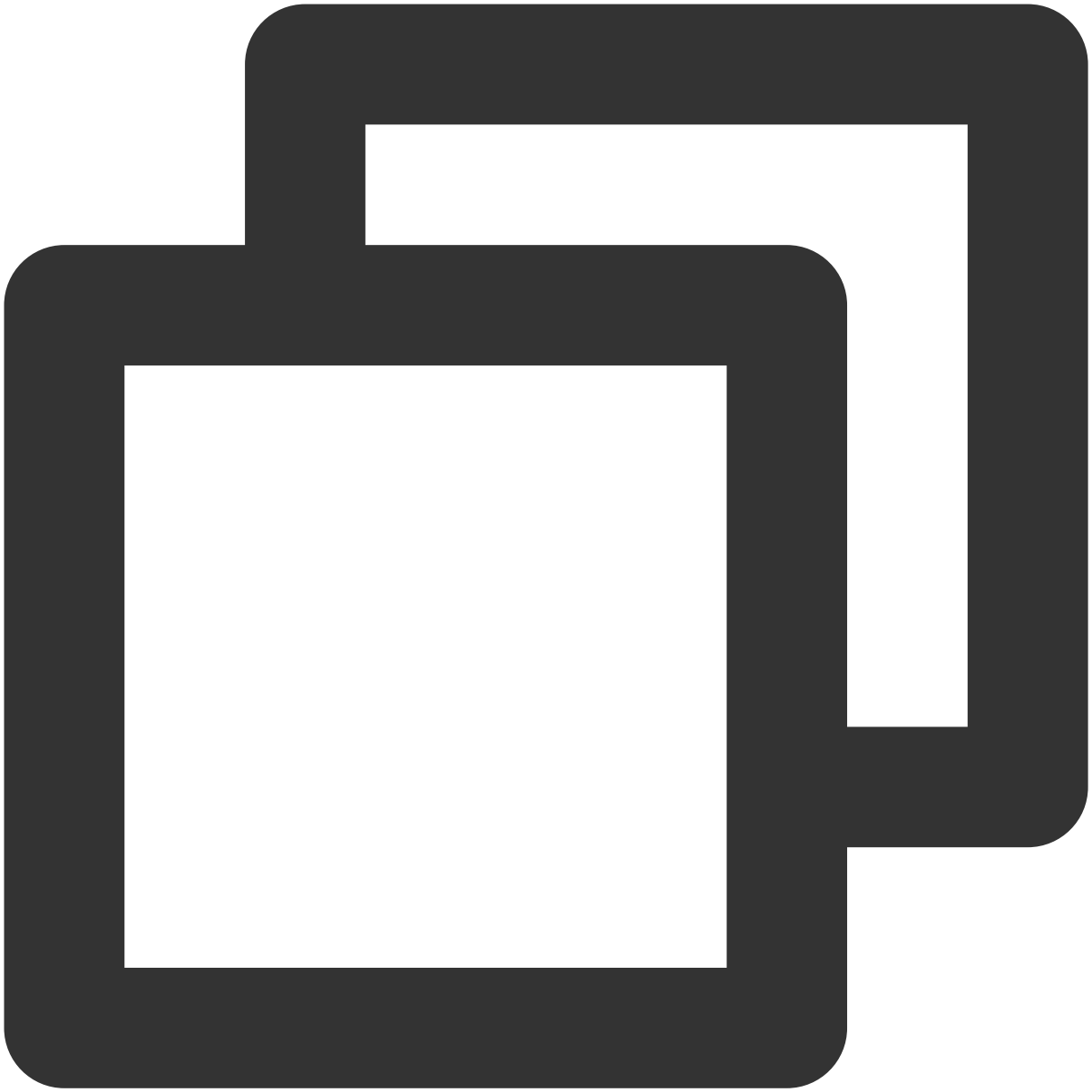
/* 用于设置上传请求是否自动添加 Content-MD5 头部，enable 为 COS_FALSE 时上传请求将不自动添加
cos_set_content_md5_enable(options->ctl, COS_FALSE);

/* 用于设置请求路由地址和端口，一般情况下无需设置此参数，请求将按域名解析结果路由 */
//cos_set_request_route(options->ctl, "192.168.12.34", 80);
```

说明

临时密钥生成和使用可参见 临时密钥生成及使用指引。

创建存储桶



```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_acl_e cos_acl = COS_ACL_PRIVATE;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

/* 重新创建一个新的内存池，第二个参数是 NULL，表示没有继承自其它内存池 */
cos_pool_create(&p, NULL);
```

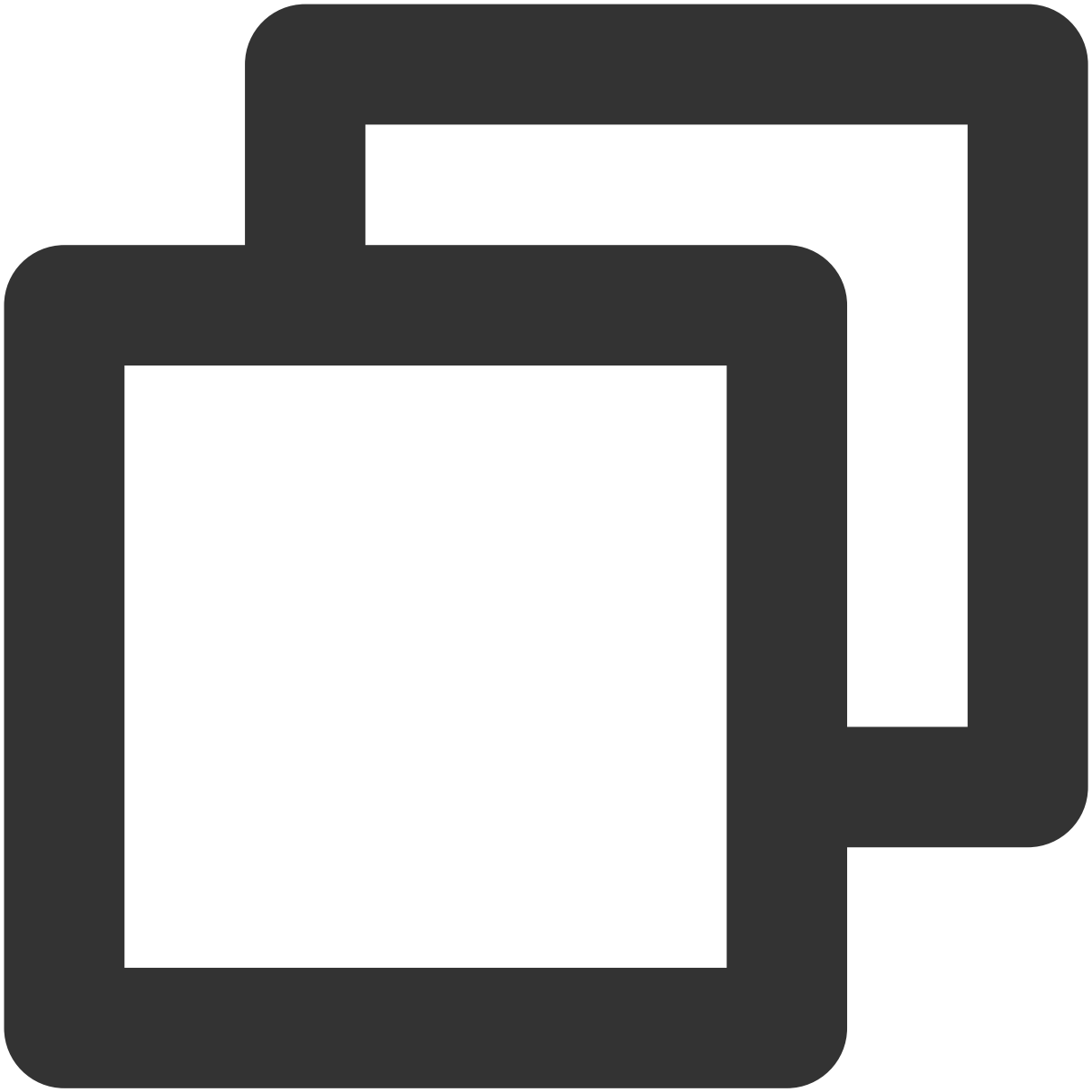
```
/* 创建并初始化 options，这个参数内部主要包括endpoint,access_key_id,acces_key_secret, is_
 * options 的内存是由 pool 分配的，后续释放掉 pool 后，options的内存也相当于释放掉了，不再需要
 */
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);

/* 设置 appid, endpoint, access_key_id, acces_key_secret, is_cname, curl参数等配置信息
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
/* 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式 */
cos_str_set(&bucket, TEST_BUCKET_NAME);

/* 调用api创建存储桶 */
s = cos_create_bucket(options, &bucket, cos_acl, &resp_headers);
if (cos_status_is_ok(s)) {
        printf("create bucket succeeded\\n");
} else {
        printf("create bucket failed\\n");
}

//destroy memory pool
cos_pool_destroy(p);
```

查询对象列表



```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_list_object_params_t *list_params = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

/* 重新创建一个新的内存池，第二个参数是NULL，表示没有继承自其它内存池 */
cos_pool_create(&p, NULL);
```

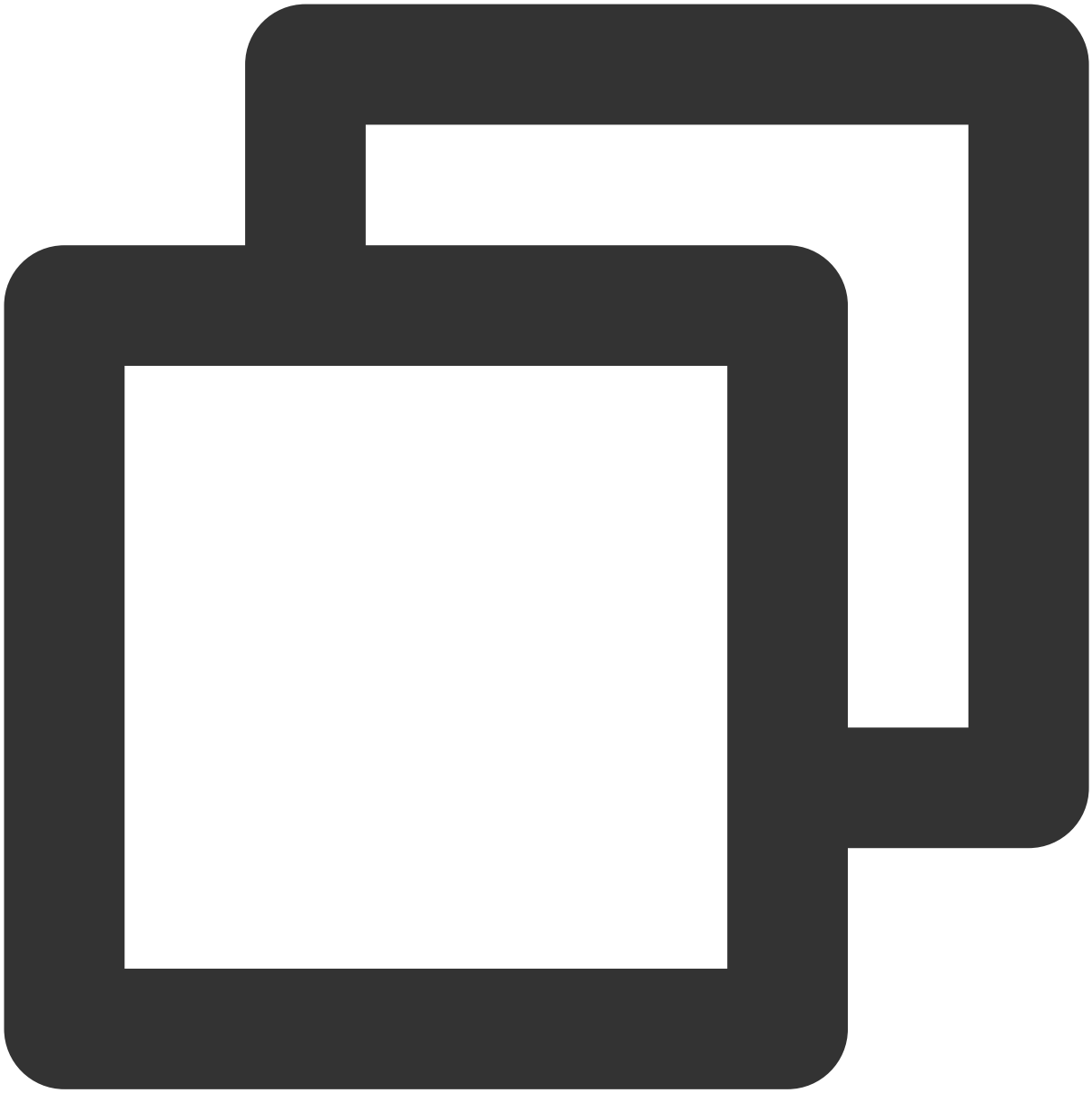
```
/* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,acces_key_secret，is_c
 * options的内存是由pool分配的，后续释放掉pool后，options的内存也相当于释放掉了，不再需要单独释
 */
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);

/* 设置 appid, endpoint, access_key_id, acces_key_secret, is_cname, curl参数等配置信息
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
/* 存储桶的命名格式为 BucketName-APPID, , 此处填写的存储桶名称必须为此格式 */
cos_str_set(&bucket, TEST_BUCKET_NAME);

/* 调用api查询对象列表 */
list_params = cos_create_list_object_params(p);
cos_str_set(&list_params->encoding_type, "url");
s = cos_list_object(options, &bucket, list_params, &resp_headers);
if (cos_status_is_ok(s)) {
        printf("list object succeeded\\n");
} else {
        printf("list object failed\\n");
}

//destroy memory pool
cos_pool_destroy(p);
```

上传对象



```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_string_t file;
cos_table_t *resp_headers = NULL;

/* 重新创建一个新的内存池，第二个参数是NULL，表示没有继承自其它内存池 */
cos_pool_create(&p, NULL);
```

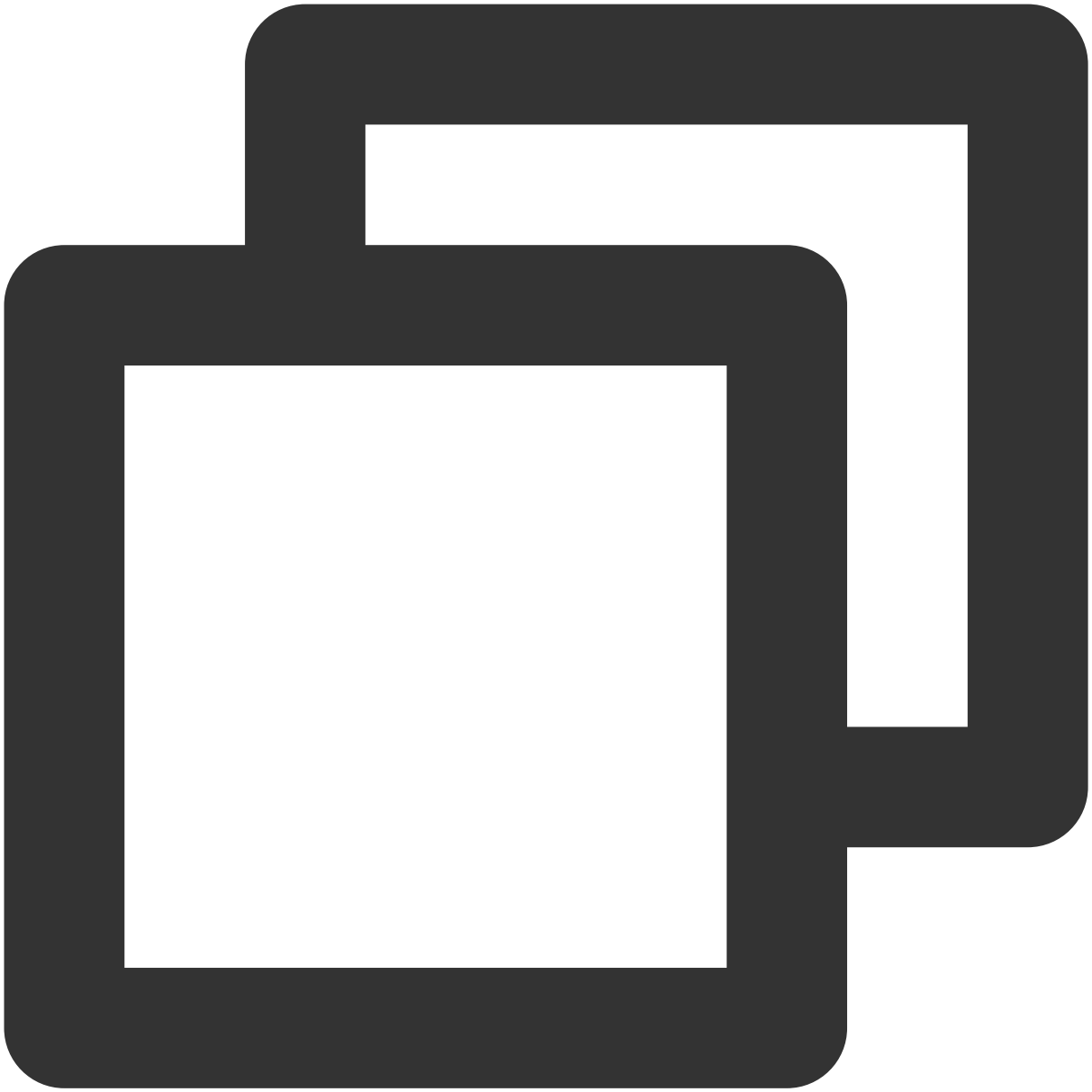
```
/* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,acces_key_secret, is_c
 * options的内存是由pool分配的，后续释放掉pool后，options的内存也相当于释放掉了，不再需要单独释
 */
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);

/* 设置appid, endpoint, access_key_id, acces_key_secret, is_cname, curl参数等配置信息 */
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
/* 存储桶的命名格式为 BucketName-APPID, , 此处填写的存储桶名称必须为此格式 */
cos_str_set(&bucket, TEST_BUCKET_NAME);

/* 调用api上传对象 */
cos_str_set(&file, TEST_DOWNLOAD_NAME);
cos_str_set(&object, TEST_OBJECT_NAME);
s = cos_put_object_from_file(options, &bucket, &object, &file, NULL, &resp_headers)
if (cos_status_is_ok(s)) {
        printf("put object succeeded\\n");
} else {
        printf("put object failed\\n");
}

//destroy memory pool
cos_pool_destroy(p);
```

下载对象



```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_string_t file;
cos_table_t *resp_headers = NULL;

/* 重新创建一个新的内存池，第二个参数是NULL，表示没有继承自其它内存池 */
cos_pool_create(&p, NULL);
```

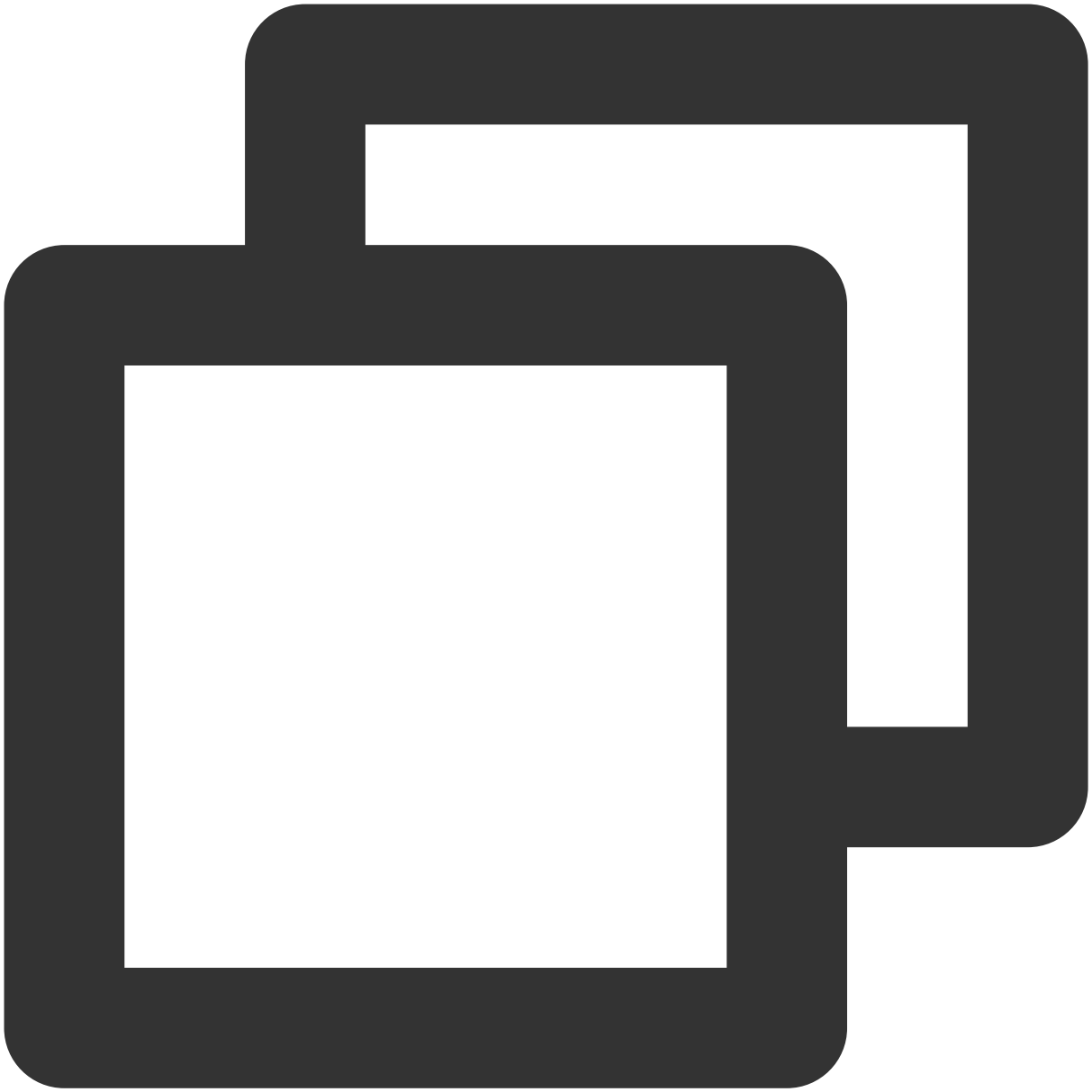
```
/* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,acces_key_secret, is_c
 * options的内存是由pool分配的，后续释放掉pool后，options的内存也相当于释放掉了，不再需要单独释
 */
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);

/* 设置appid, endpoint, access_key_id, acces_key_secret, is_cname, curl参数等配置信息 */
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
/* 存储桶的命名格式为 BucketName-APPID, , 此处填写的存储桶名称必须为此格式 */
cos_str_set(&bucket, TEST_BUCKET_NAME);

/* 调用api下载对象 */
cos_str_set(&file, TEST_DOWNLOAD_NAME);
cos_str_set(&object, TEST_OBJECT_NAME);
s = cos_get_object_to_file(options, &bucket, &object, NULL, NULL, &file, &resp_head
if (cos_status_is_ok(s)) {
        printf("get object succeeded\\n");
} else {
        printf("get object failed\\n");
}

//destroy memory pool
cos_pool_destroy(p);
```

删除对象



```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_table_t *resp_headers = NULL;

/* 重新创建一个新的内存池，第二个参数是 NULL，表示没有继承自其它内存池 */
cos_pool_create(&p, NULL);
```

```
/* 创建并初始化 options，这个参数内部主要包括 endpoint,access_key_id,acces_key_secret, is
 * options的内存是由pool分配的，后续释放掉 pool 后，options 的内存也相当于释放掉了，不再需要单
 */
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);

/* 设置 appid, endpoint, access_key_id, acces_key_secret, is_cname, curl参数等配置信息
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
/* 存储桶的命名格式为 BucketName-APPID, , 此处填写的存储桶名称必须为此格式 */
cos_str_set(&bucket, TEST_BUCKET_NAME);

/* 调用 api 删除对象 */
cos_str_set(&object, TEST_OBJECT_NAME);
s = cos_delete_object(options, &bucket, &object, &resp_headers);
if (cos_status_is_ok(s)) {
        printf("delete object succeeded\\n");
} else {
        printf("delete object failed\\n");
}

//destroy memory pool
cos_pool_destroy(p);
```

C SDK

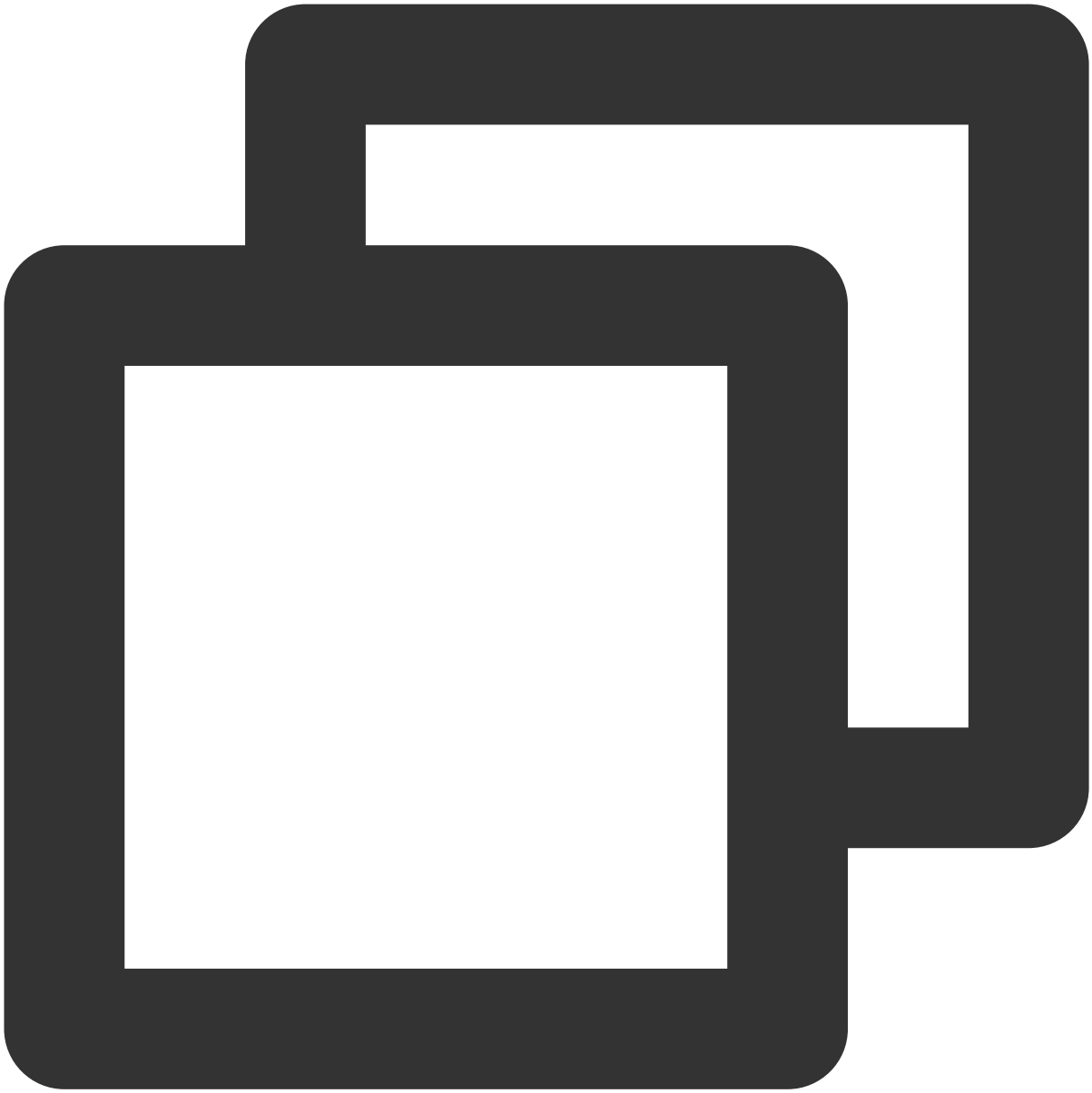
最近更新时间：2024-01-04 17:06:08

使用 C SDK 如何实现断点续传？

可以使用 C SDK 高级上传 接口实现断点续传功能。使用断点续传时，需设置上传控制参数为 **COS_TRUE**，例如：`clt_params = cos_create_resumable_clt_params_content(p, 0, 1, COS_TRUE, NULL)` 。

使用 C SDK 出现 HttpIOError 错误？

在 SDK 使用过程中发现任何一个接口都没法正常使用，且没法返回 requestid，抓包分析发现 HTTP 请求都没有发送出去。结合如下的日志情况：



```
transport failure curl code:1 error:Unsupported protocol
status->code: -996
status->error_code: HttpIoError
status->error_msg: Unsupported protocol
status->req_id:
```

发现出现这种情况的原因为：使用了 HTTPS 协议，但是 libcurl 库却不支持 HTTPS 协议。从而导致 libcurl 编译时没有使用 openssl 库或版本不匹配。

解决方法：检查运行环境，重新安装 libcurl 库（源码编译安装需要使能 SSL）或更新 openssl 库。

存储桶操作

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于存储桶的基本操作和访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

注意

建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。

如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

基本操作

API	操作名	操作描述
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

检索存储桶及其权限

API	操作名	操作描述
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问

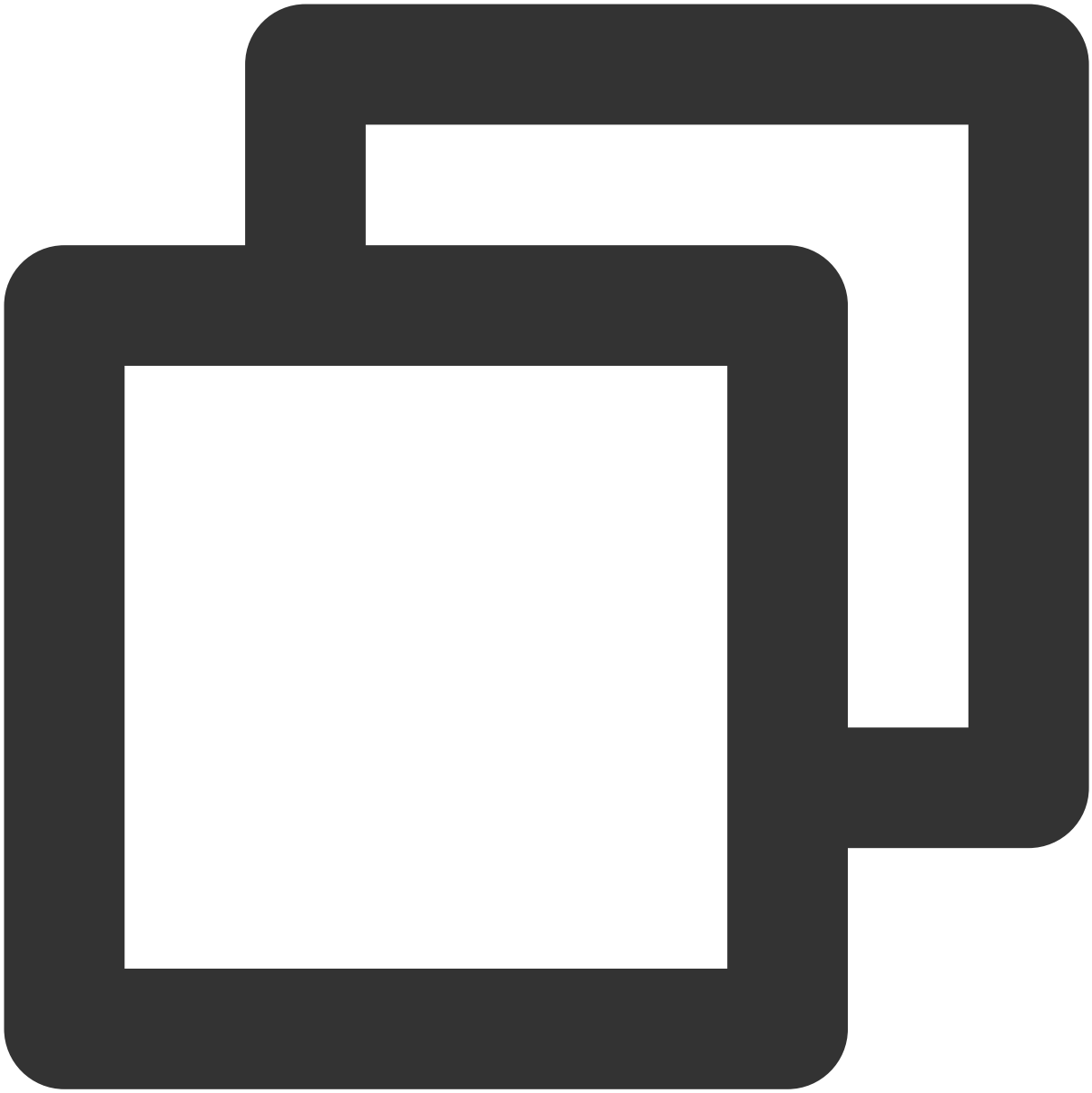
基本操作

创建存储桶

功能说明

在指定账号下创建一个存储桶。

方法原型



```
cos_status_t *cos_create_bucket(const cos_request_options_t *options,
                                const cos_string_t *bucket,
                                cos_acl_e cos_acl,
                                cos_table_t **resp_headers);
```

参数说明

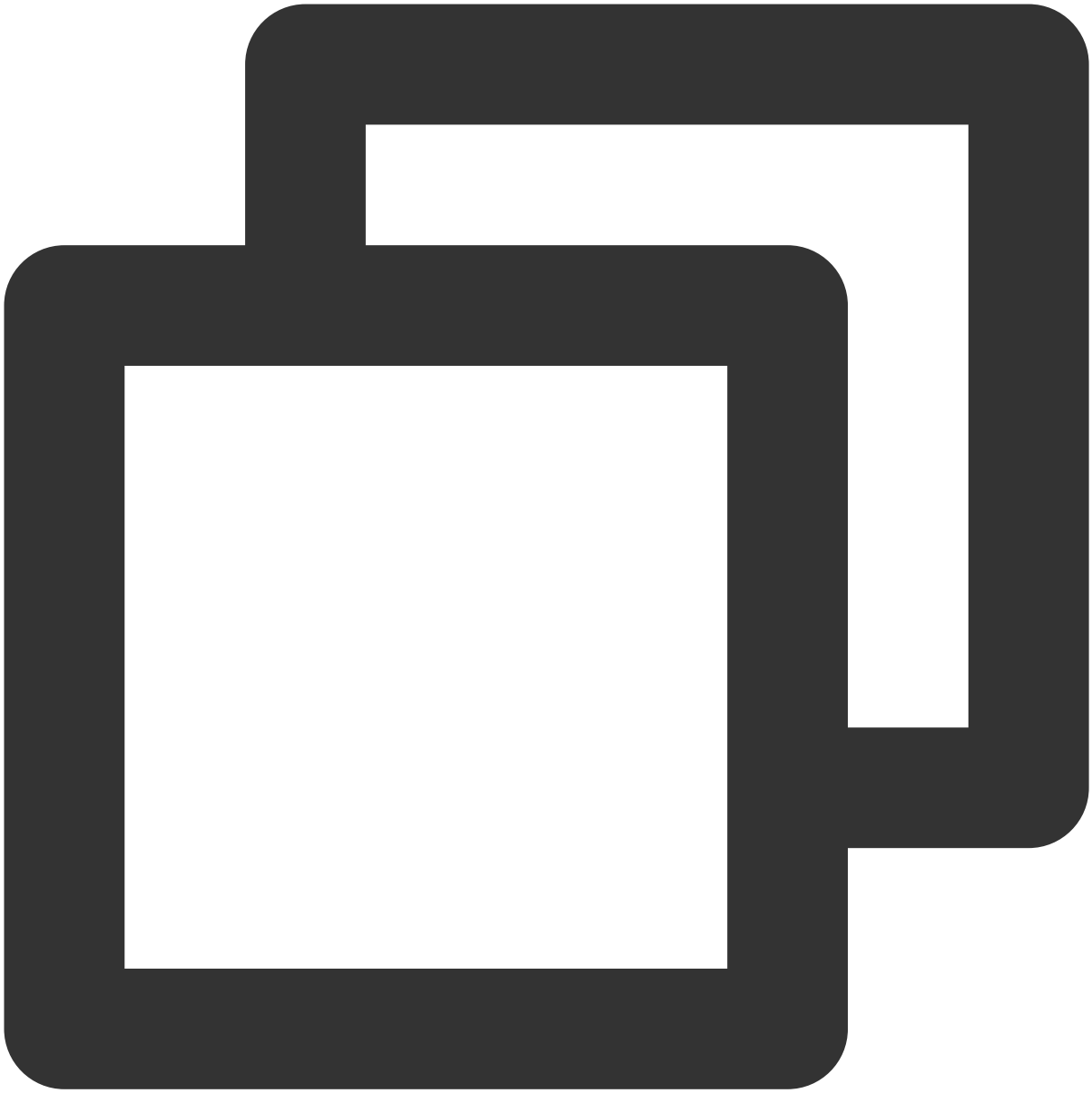
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
cos_acl	允许用户自定义权限。 有效值：COS_ACL_PRIVATE(0)，COS_ACL_PUBLIC_READ(1)，COS_ACL_PUBLIC_READ_WRITE(2) 默认值：COS_ACL_PRIVATE(0)	Enum
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_create_bucket()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_acl_e cos_acl = COS_ACL_PRIVATE;
    cos_string_t bucket;
    cos_table_t *resp_headers;

    //初始化请求选项
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //创建存储桶
    s = cos_create_bucket(options, &bucket, cos_acl, &resp_headers);
    if (cos_status_is_ok(s)) {
```

```
            printf("create bucket succeeded\\n");
    } else {
            printf("create bucket failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_create_bucket();

    cos_http_io_deinitialize();

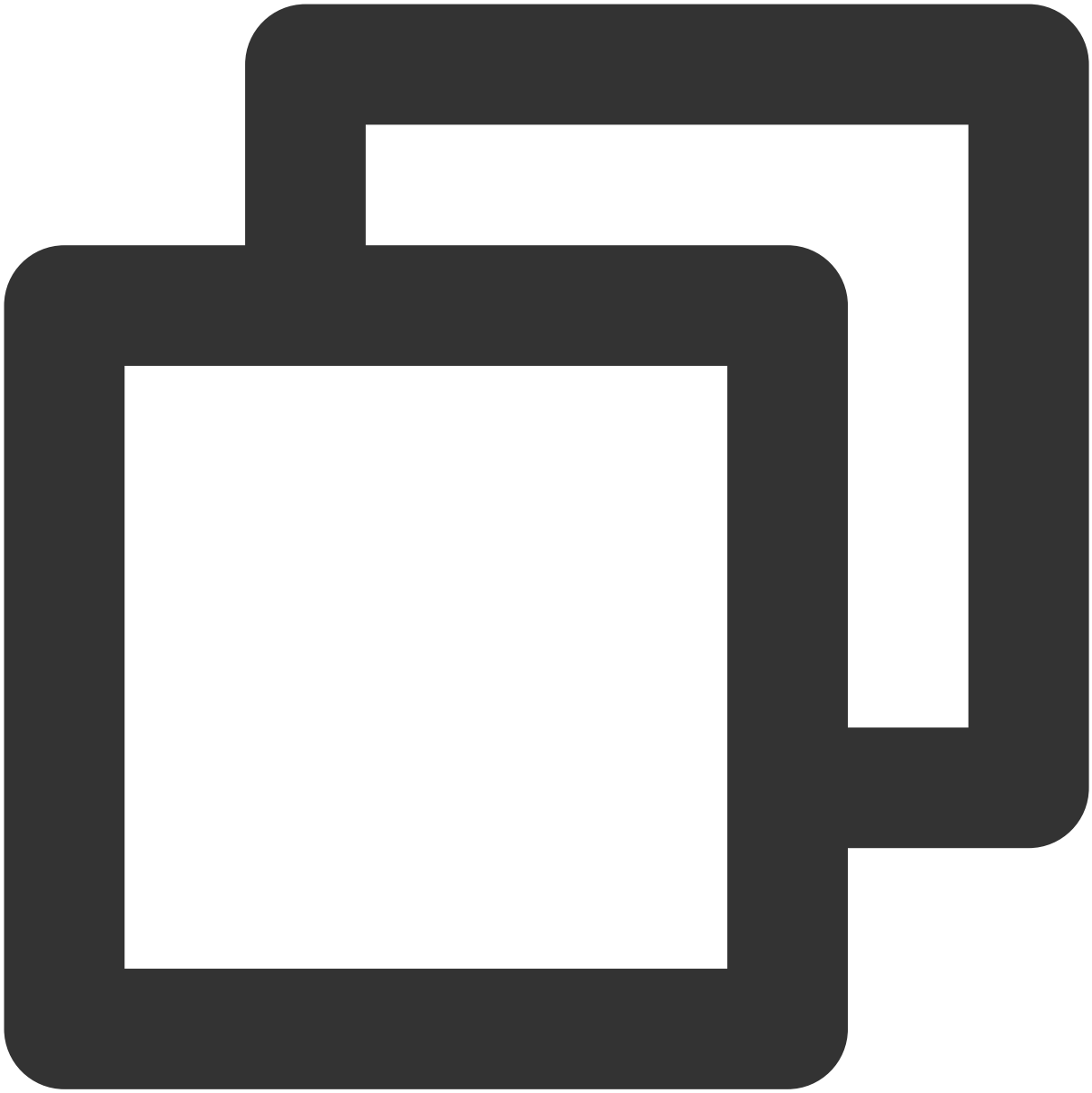
    return 0;
}
```

删除存储桶

功能说明

删除指定账号下的空存储桶。

方法原型



```
cos_status_t *cos_delete_bucket(const cos_request_options_t *options,
                                const cos_string_t *bucket,
                                cos_table_t **resp_headers);
```

参数说明

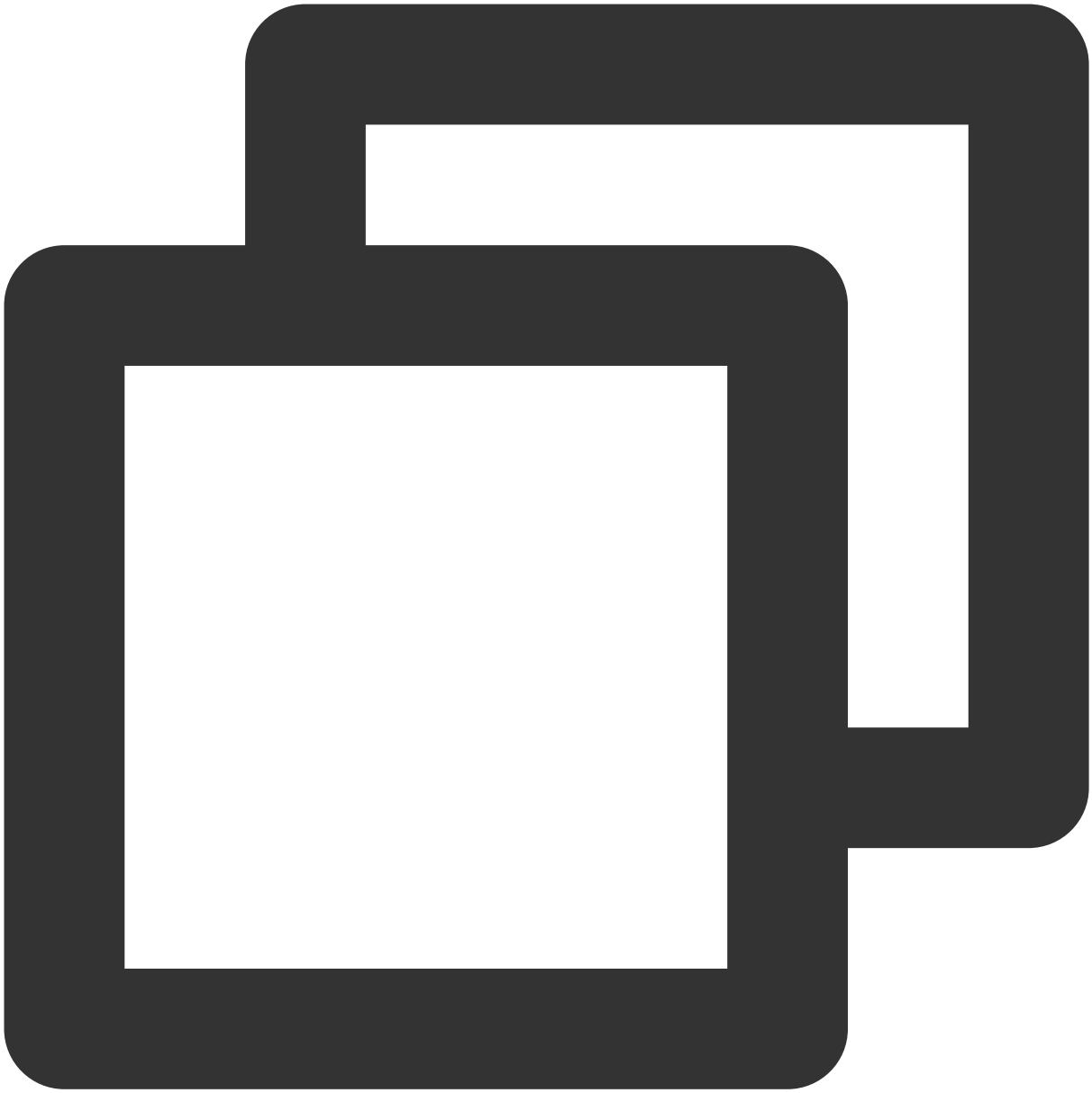
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_bucket()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers;

    //初始化请求选项
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //删除存储桶
    s = cos_delete_bucket(options, &bucket, &resp_headers);
    if (cos_status_is_ok(s)) {
            printf("create bucket succeeded\\n");
```

```
    } else {
            printf("create bucket failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_delete_bucket();

    cos_http_io_deinitialize();

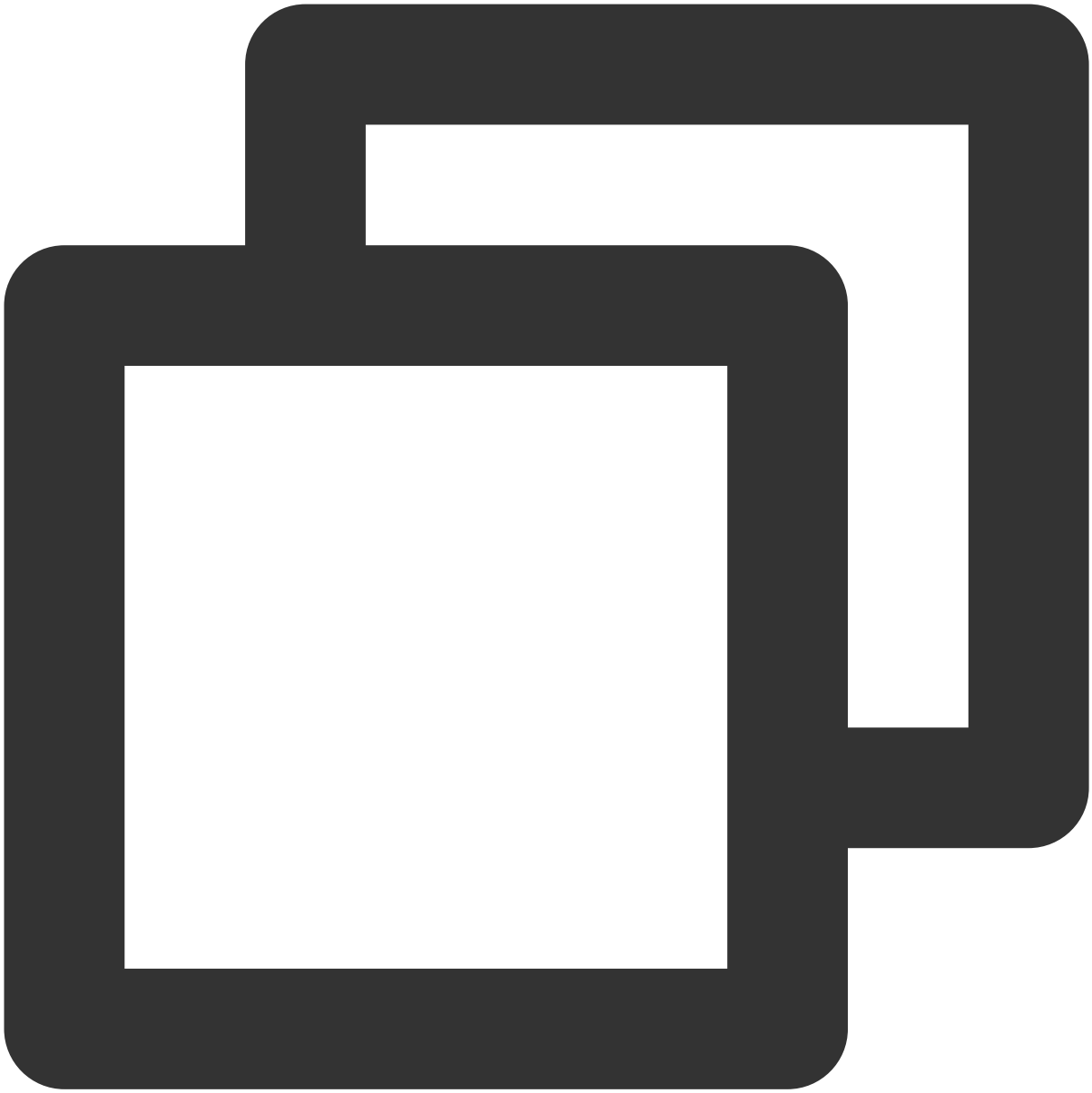
    return 0;
}
```

判断存储桶是否存在

功能说明

检查存储桶是否存在，此接口实际是调用了HEAD Bucket API来检查对象是否存在的。

方法原型



```
cos_status_t *cos_check_bucket_exist(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     cos_bucket_exist_status_e *bucket_exist,
                                     cos_table_t **resp_headers)
```

参数说明

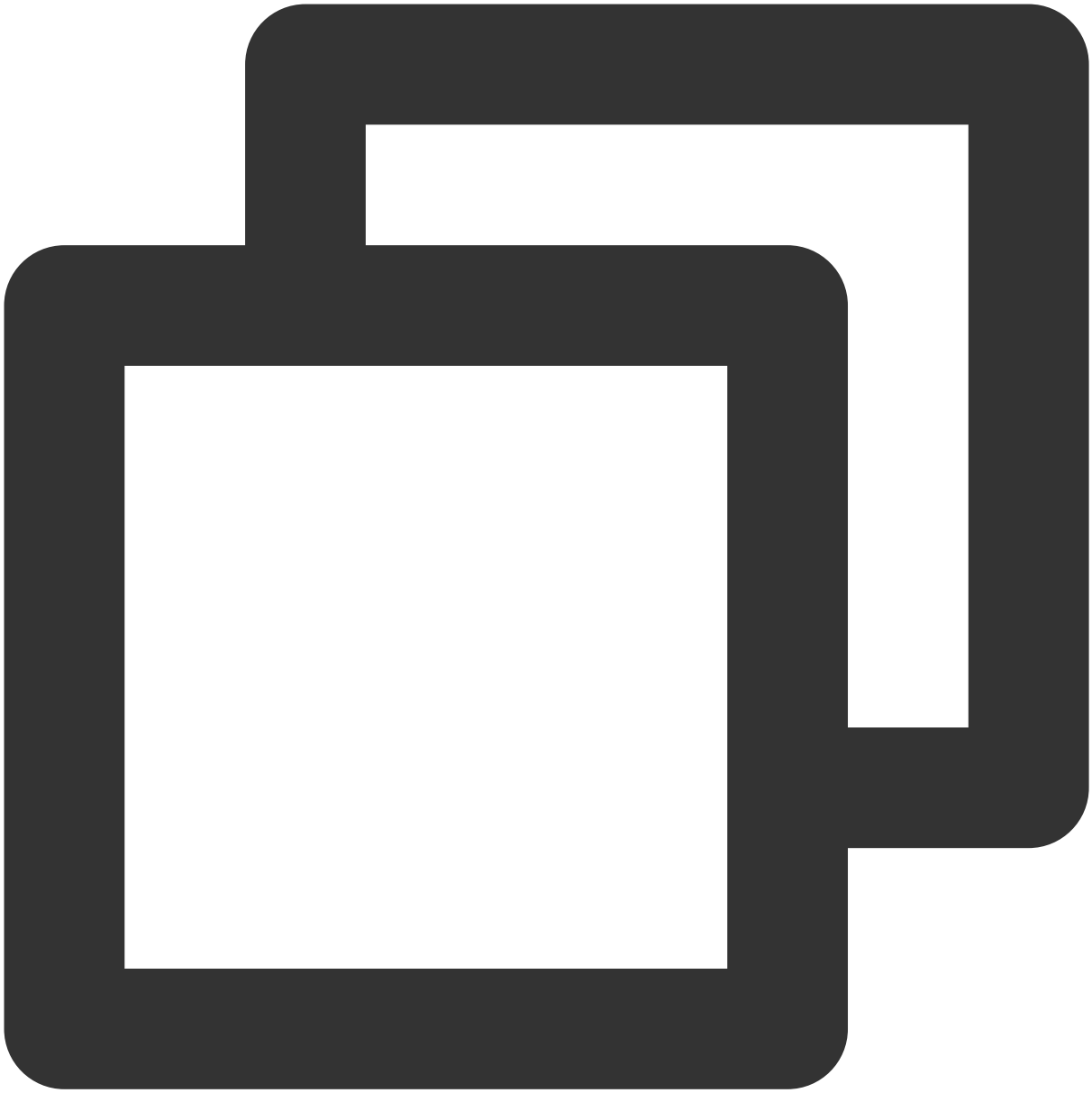
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
bucket_exist	桶是否存在的枚举值，分为存在、不存在、未知状态(没有head到明确的状态)	Enum
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_check_bucket_exist()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;
    cos_bucket_exist_status_e bucket_exist;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);

    cos_str_set(&bucket, TEST_BUCKET_NAME);
```

```
    // 检查桶是否存在
    status = cos_check_bucket_exist(options, &bucket, &bucket_exist, &resp_headers)
    if (bucket_exist == COS_BUCKET_NON_EXIST) {
        printf("bucket: %.*s non exist.\\n", bucket.len, bucket.data);
    } else if (bucket_exist == COS_BUCKET_EXIST) {
        printf("bucket: %.*s exist.\\n", bucket.len, bucket.data);
    } else {
        printf("bucket: %.*s unknown status.\\n", bucket.len, bucket.data);
        log_status(status);
    }

    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_check_bucket_exist();

    cos_http_io_deinitialize();

    return 0;
}
```

对象操作
上传对象

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于对象的上传操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object	简单上传对象（创建文件夹）	上传一个对象至存储桶
APPEND Object	追加上传对象	使用分块追加的方式上传对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传任务
Upload Part	上传分块	分块上传文件
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

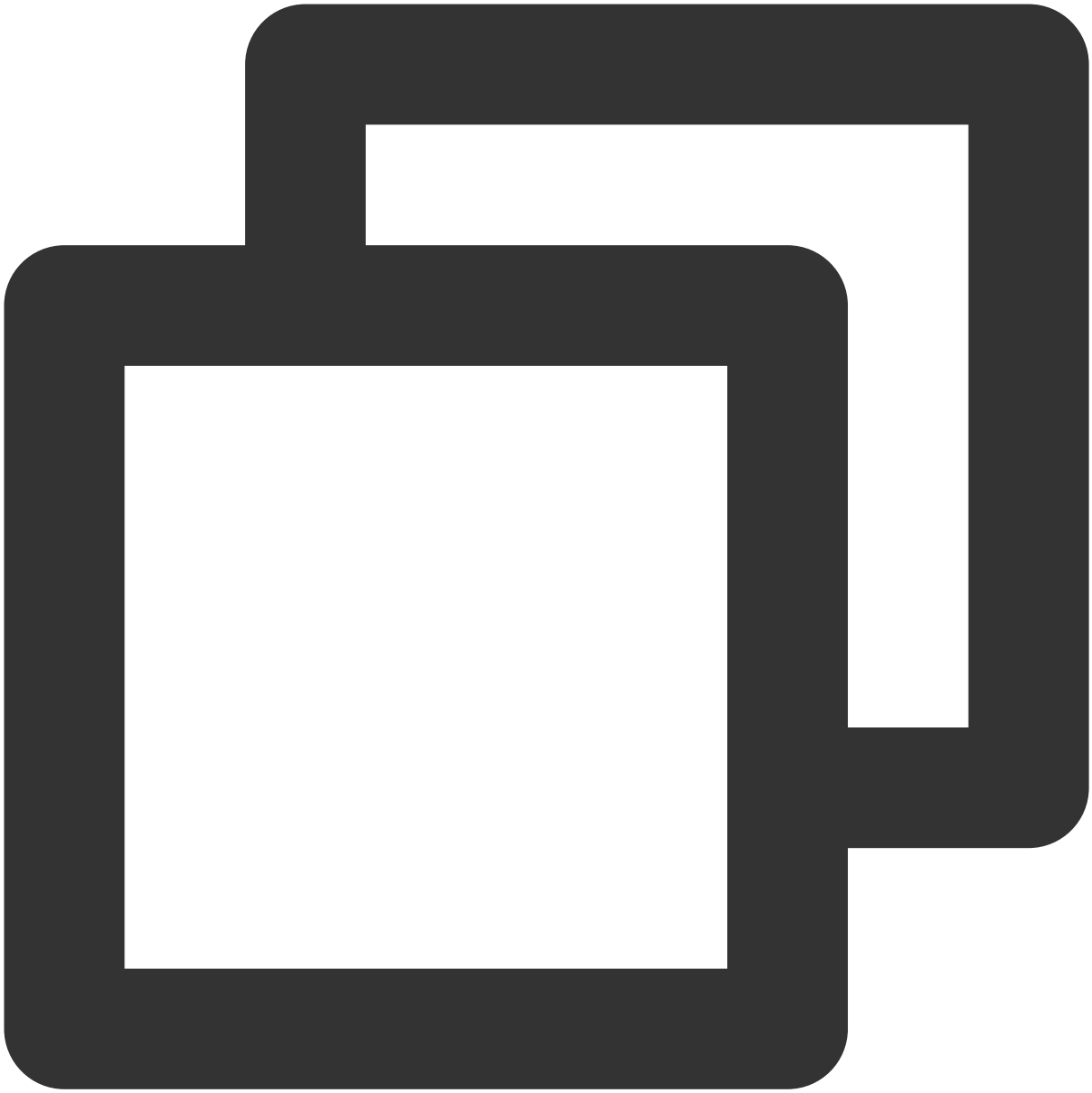
高级接口（推荐）

上传对象（断点续传）

功能说明

上传接口根据用户文件的长度，自动切分数据， 降低用户的使用门槛，用户无需关心分块上传的每个步骤， 且可以对分块上传未完成的文件会进行断点续传，分块大小默认 1048576（1MB），可通过 part_size 参数调整。

方法原型



```
cos_status_t *cos_resumable_upload_file(cos_request_options_t *options,
                                        cos_string_t *bucket,
                                        cos_string_t *object,
                                        cos_string_t *filepath,
                                        cos_table_t *headers,
                                        cos_table_t *params,
                                        cos_resumable_clt_params_t *clt_params,
                                        cos_progress_callback progress_callback,
                                        cos_table_t **resp_headers,
                                        cos_list_t *resp_body)
```

参数说明

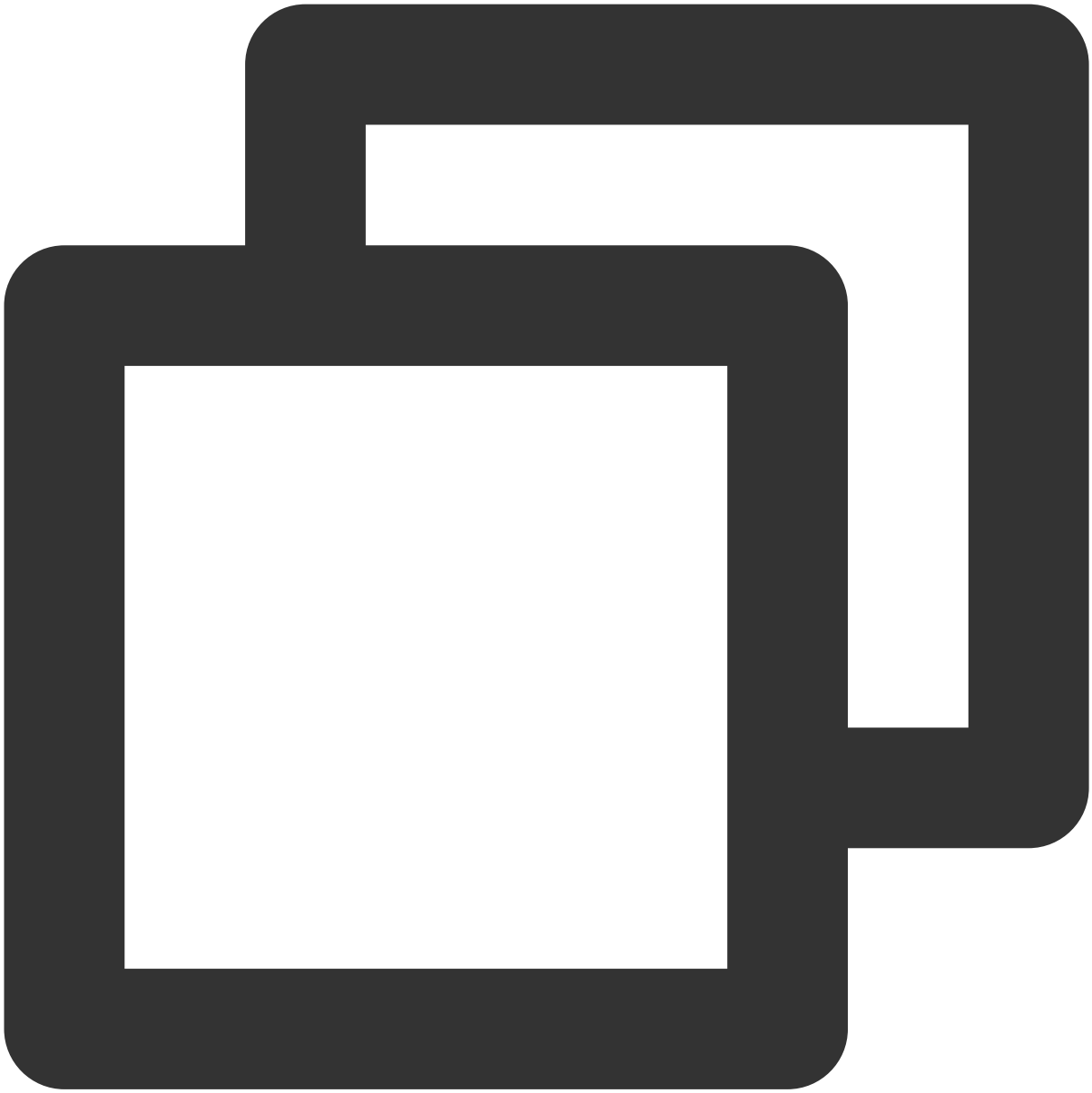
参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
filepath	Object 本地文件名称	String
headers	COS 请求附加头域	Struct
params	COS 请求操作参数	Struct
clt_params	上传对象控制参数	Struct
part_size	块大小，单位为 bytes，如果用户指定的 part_size 小于1048576（1MB），由 C SDK 自动切分, 分块大小默认1048576（1MB），如果分块数超过10000，则根据文件大小调整	Int
thread_num	线程池大小，默认为1	Int
enable_checkpoint	是否使能断点续传	Int
checkpoint_path	当使能断点续传时，表示保存上传进度的文件路径，默认路径为 `<filepath>.cp` ，其中 filepath 为 Object 本地文件名称	String
progress_callback	上传进度回调函数	Function
resp_headers	返回 HTTP 响应消息的头域	Struct
resp_body	保存完成分块上传请求时返回的数据	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

版权所有：腾讯云计算（北京）有限责任公司

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
```

```
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_FILE[] = "test.zip";
static char TEST_MULTIPART_OBJECT4[] = "multipart4.dat";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_resumable_upload_with_multi_threads()
{
    cos_pool_t *p = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t filename;
    cos_status_t *s = NULL;
    int is_cname = 0;
    cos_table_t *headers = NULL;
    cos_table_t *resp_headers = NULL;
    cos_request_options_t *options = NULL;
    cos_resumable_clt_params_t *clt_params;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
```

```
    headers = cos_table_make(p, 0);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_MULTIPART_OBJECT4);
    cos_str_set(&filename, TEST_MULTIPART_FILE);

    // upload
    clt_params = cos_create_resumable_clt_params_content(p, 1024 * 1024, 8, COS_FAL
    s = cos_resumable_upload_file(options, &bucket, &object, &filename, headers, NU
        clt_params, NULL, &resp_headers, NULL);

    if (cos_status_is_ok(s)) {
        printf("upload succeeded\\n");
    } else {
        printf("upload failed\\n");
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_resumable_upload_with_multi_threads();

    cos_http_io_deinitialize();

    return 0;
}
```

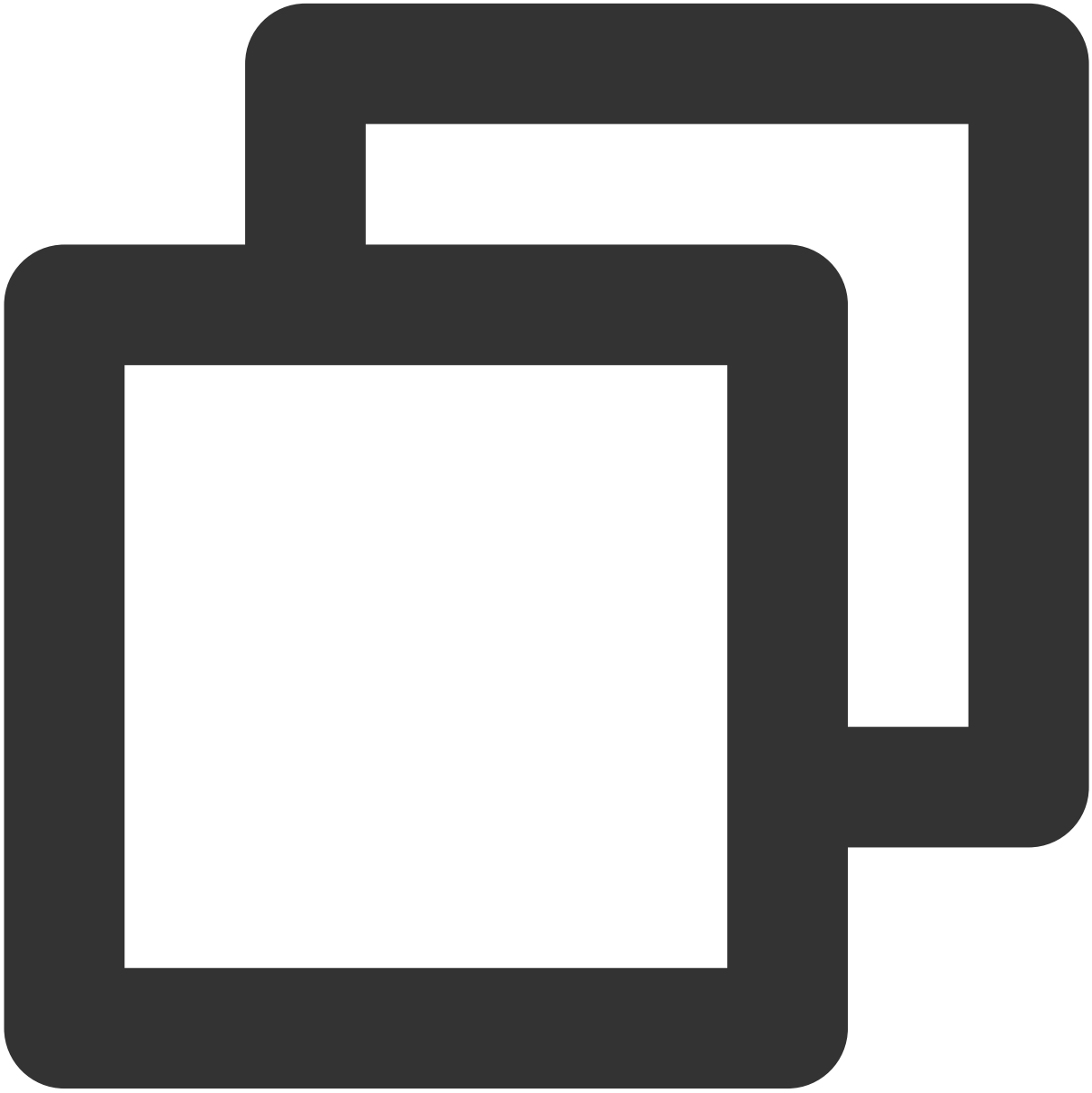
简单操作

简单上传对象（创建文件夹）

功能说明

上传一个对象至存储桶，最大支持上传不超过5GB的对象，5GB以上对象请使用 分块上传 或 高级接口 上传。

方法原型



```
cos_status_t *cos_put_object_from_file(const cos_request_options_t *options,
                                       const cos_string_t *bucket,
                                       const cos_string_t *object,
                                       const cos_string_t *filename,
```

```
                                        cos_table_t *headers,
                                        cos_table_t **resp_headers);
```

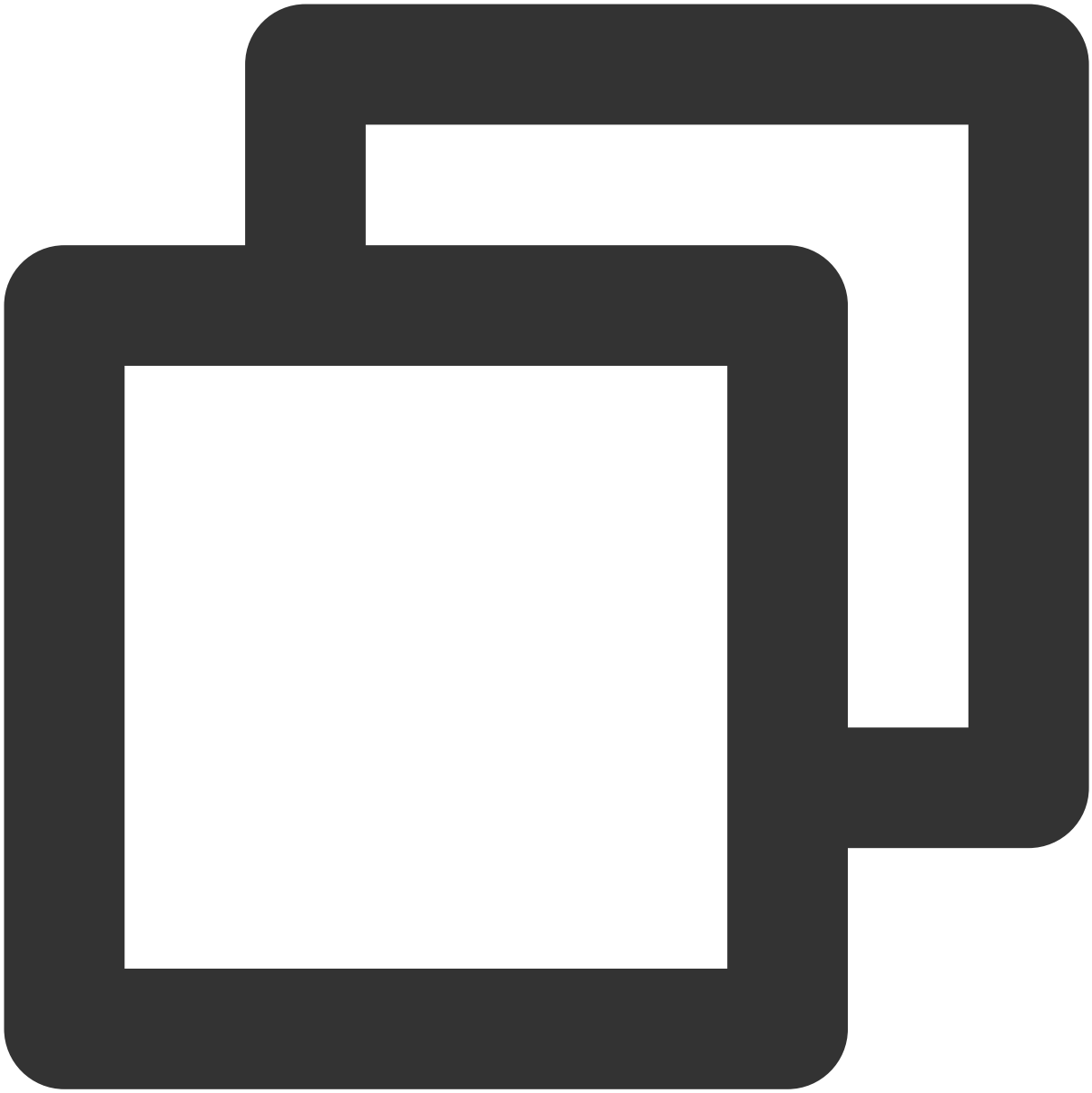
参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
filename	Object 本地保存文件名称	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例1：上传对象



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_object_from_file()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_table_t *resp_headers;
    cos_string_t file;
    int traffic_limit = 0;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_table_t *headers = NULL;
    if (traffic_limit) {
        // 限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，如果超出该范围将返回40
```

```
        headers = cos_table_make(p, 1);
        cos_table_add_int(headers, "x-cos-traffic-limit", 819200);
    }
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&file, TEST_OBJECT_NAME1);
    cos_str_set(&object, TEST_OBJECT_NAME1);
    s = cos_put_object_from_file(options, &bucket, &object, &file, headers, &resp_h
    log_status(s);

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

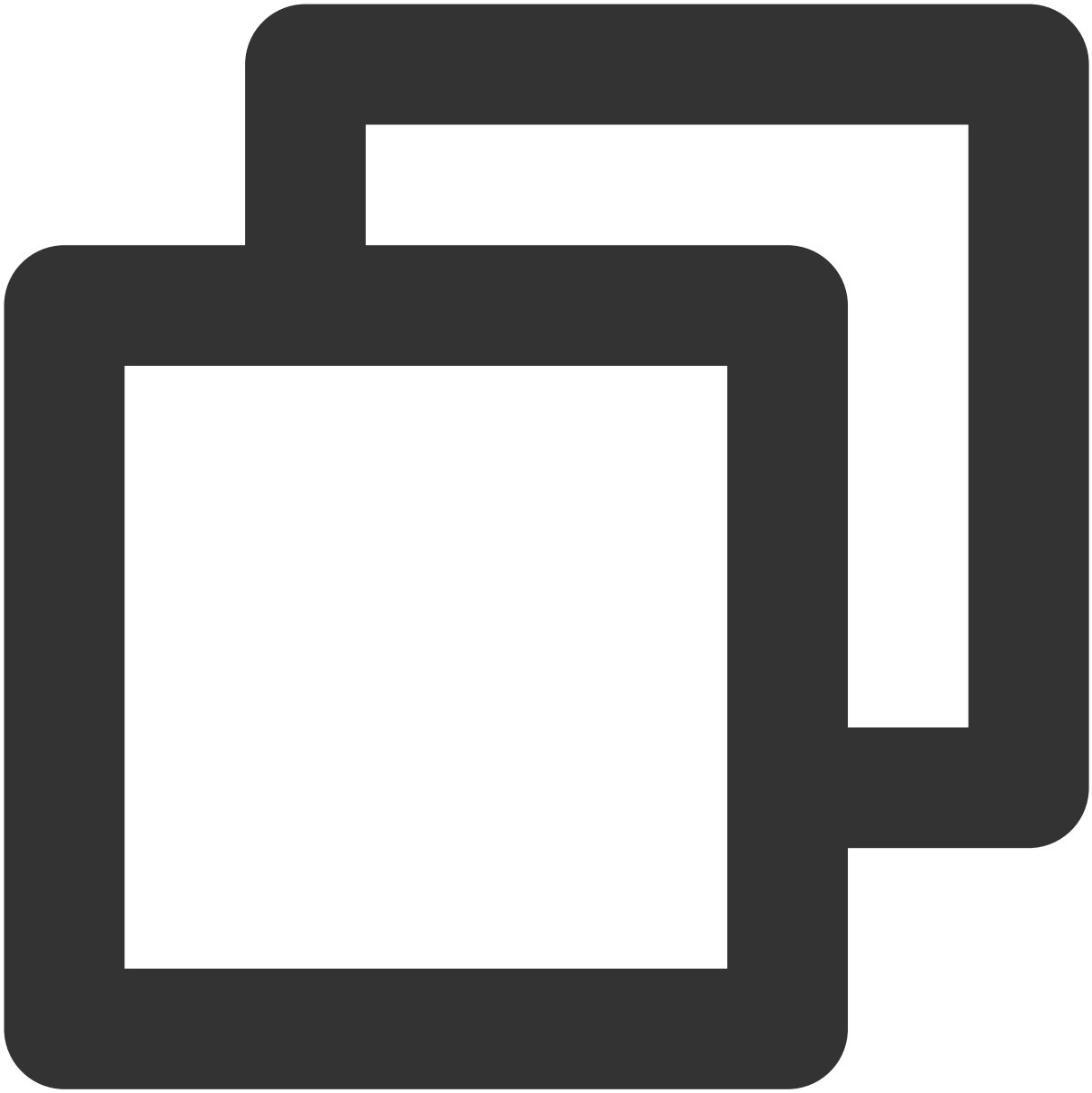
    test_put_object_from_file();

    cos_http_io_deinitialize();

    return 0;
}
```

示例2：创建文件夹

COS 上可以将以 `/` 分隔的对象路径看做一个虚拟文件夹，根据此特性，可以上传一个空的流，并且命名以 `/` 结尾，可实现在 COS 上创建一个空文件夹。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_create_dir()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_table_t *resp_headers;
    cos_table_t *headers = NULL;
    cos_list_t buffer;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, "folder/");

    //上传文件夹
    cos_list_init(&buffer);
```

```
    s = cos_put_object_from_buffer(options, &bucket, &object,
            &buffer, headers, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("put object succeeded\\n");
    } else {
        printf("put object failed\\n");
    }
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

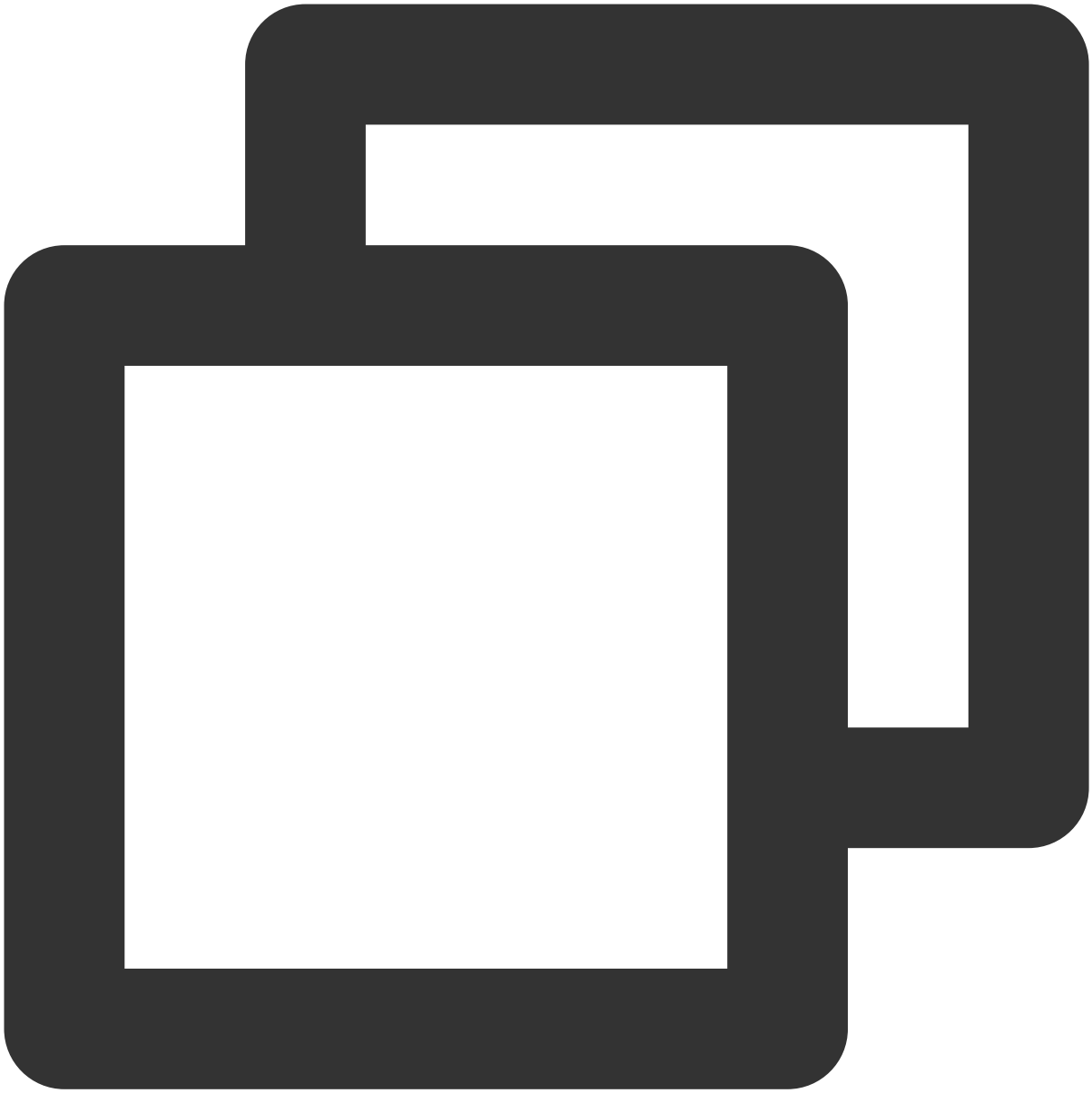
    test_create_dir();

    cos_http_io_deinitialize();

    return 0;
}
```

示例3：上传文件到虚拟目录

上传由 `/` 分隔的对象名，可自动创建包含文件的文件夹。想要在此文件夹中添加新文件时，只需要在上传文件至 COS 时，将 Key 填写为此目录前缀即可。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_upload_file_to_dir()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t file;
    cos_table_t *resp_headers;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&file, "examplefile");
    cos_str_set(&object, "folder/exampleobject");
    s = cos_put_object_from_file(options, &bucket, &object, &file, NULL, &resp_head
    if (cos_status_is_ok(s)) {
        printf("put object succeeded\\n");
```

```
    } else {
        printf("put object failed\\n");
    }
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_upload_file_to_dir();

    cos_http_io_deinitialize();

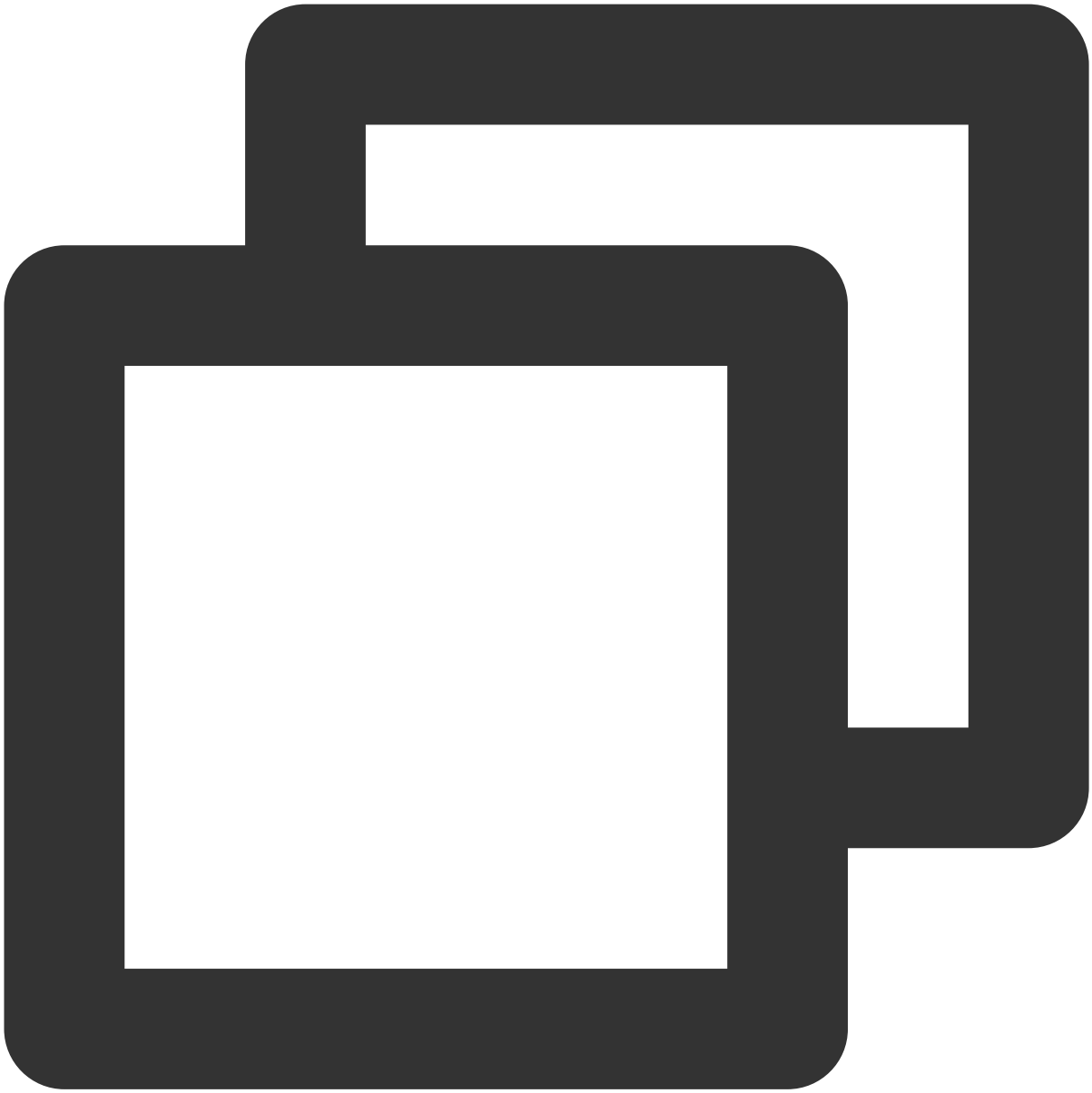
    return 0;
}
```

追加上传对象

功能说明

使用分块追加的方式上传对象，追加上传的对象，每个分块大小默认最大为5GB，无最小限制，同时通过追加方式产生的对象大小不得超过5GB。如果 Position 的值和当前对象的长度不致，COS 将返回409错误。如果追加一个 normal 属性的文件，COS 将返回409 ObjectNotAppendable。

方法原型



```
cos_status_t *cos_append_object_from_file(const cos_request_options_t *options,
                                          const cos_string_t *bucket,
                                          const cos_string_t *object,
                                          int64_t position,
                                          const cos_string_t *append_file,
                                          cos_table_t *headers,
                                          cos_table_t **resp_headers);
```

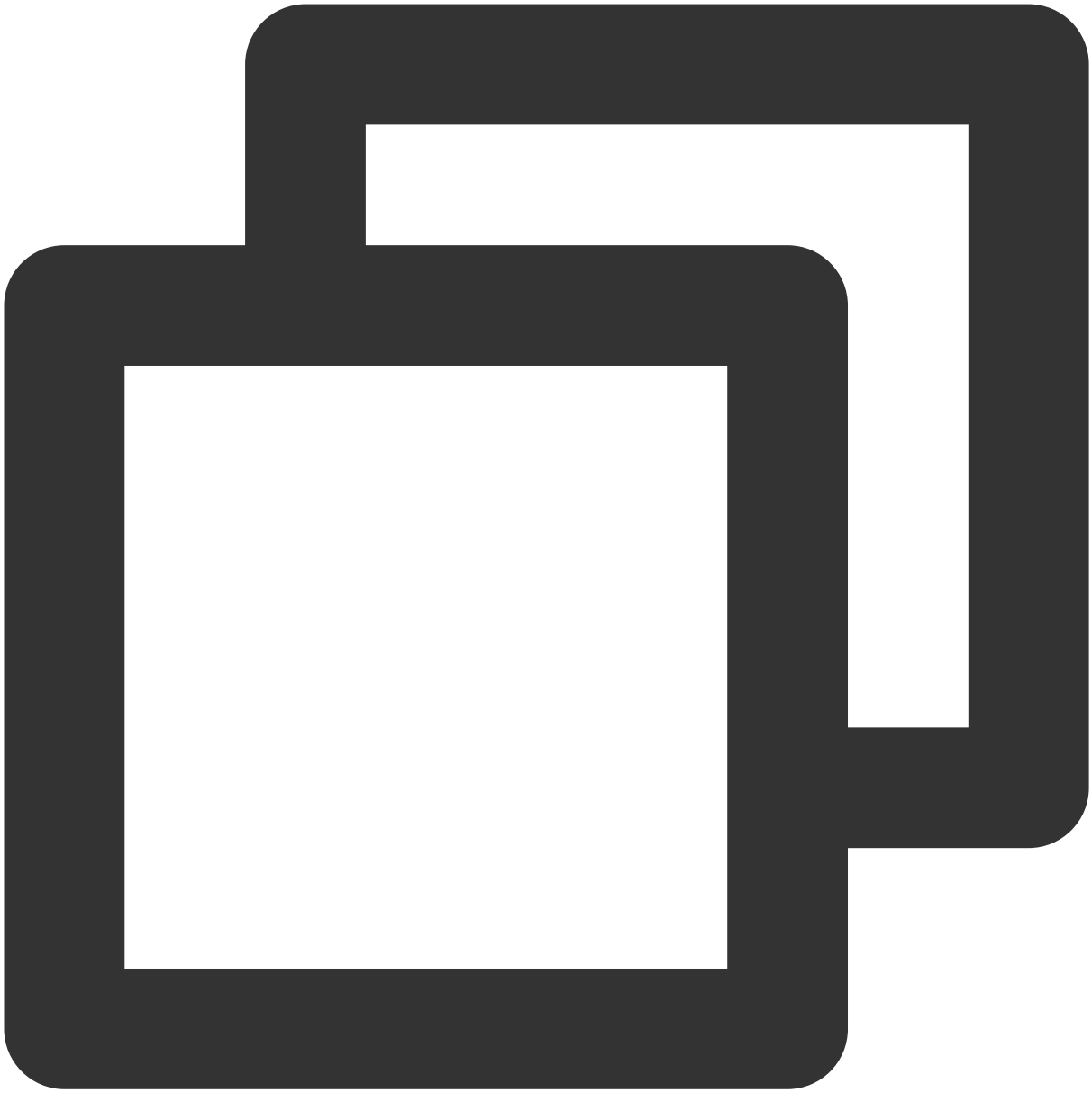
参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
position	追加操作的起始点，单位为字节。首次追加则设置 Position=0，后续追加则设置 Position 为当前 Object 的 content-length	String
append_file	Object 本地保存文件名称	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_OBJECT_NAME3[] = "test3.dat";
static char *TEST_APPEND_NAMES[] = {"test.7z.001", "test.7z.002"};

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_append_object()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t file;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
```

```
    //追加上传对象
    cos_str_set(&object, TEST_OBJECT_NAME3);
    int32_t count = sizeof(TEST_APPEND_NAMES)/sizeof(char*);
    int32_t index = 0;
    int64_t position = 0;
    s = cos_head_object(options, &bucket, &object, NULL, &resp_headers);
    if(s->code == 200) {
        char *content_length_str = (char*)apr_table_get(resp_headers, COS_CONTENT_L
        if (content_length_str != NULL) {
            position = atol(content_length_str);
        }
    }
    for (; index < count; index++)
    {
        cos_str_set(&file, TEST_APPEND_NAMES[index]);
        s = cos_append_object_from_file(options, &bucket, &object,
                                        position, &file, NULL, &resp_headers);
        log_status(s);

        s = cos_head_object(options, &bucket, &object, NULL, &resp_headers);
        if(s->code == 200) {
            char *content_length_str = (char*)apr_table_get(resp_headers, COS_CONTE
            if (content_length_str != NULL) {
                position = atol(content_length_str);
            }
        }
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
```

```
    cos_log_set_output(NULL);

    test_append_object();

    cos_http_io_deinitialize();

    return 0;
}
```

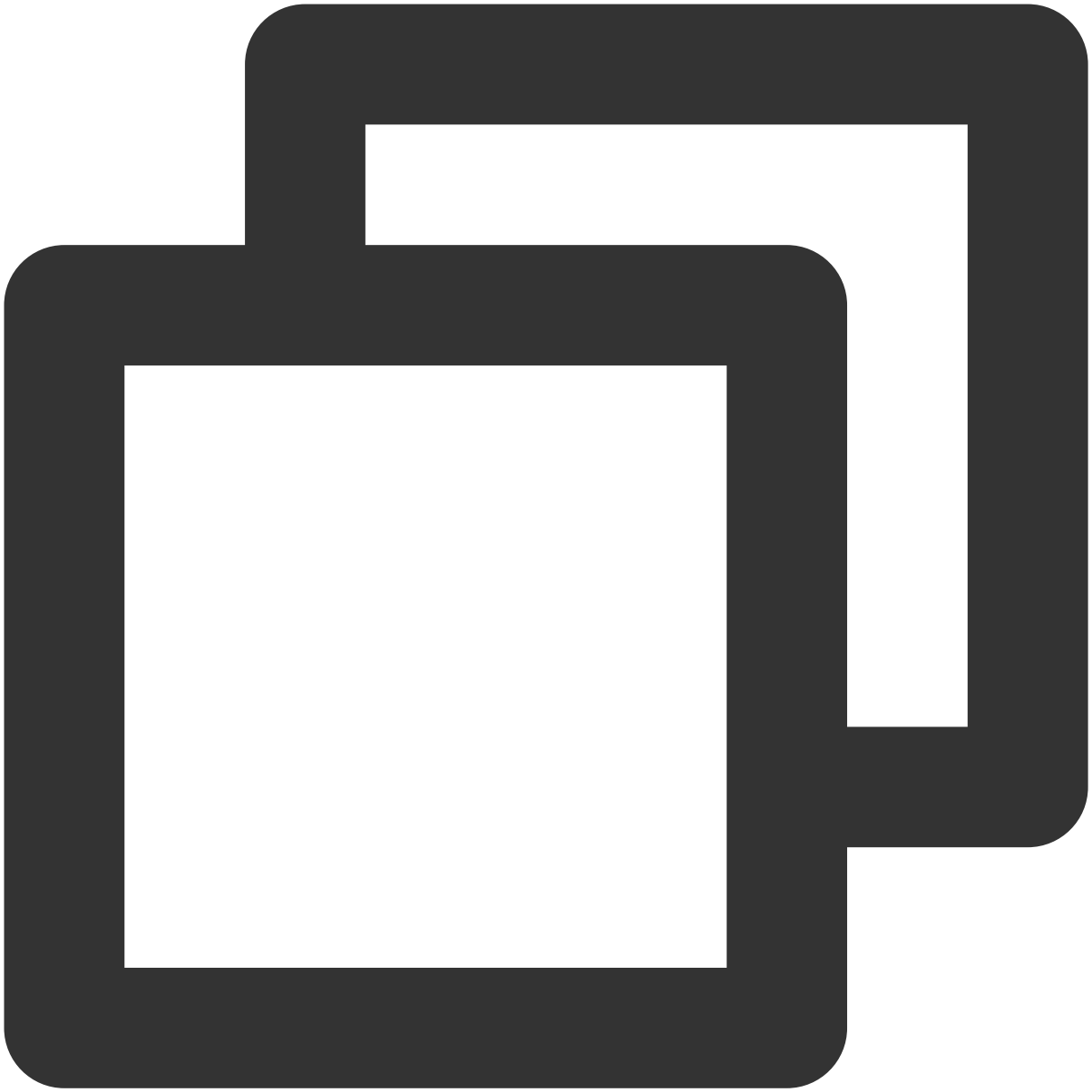
分块操作

查询分块上传

功能说明

查询正在进行中的分块上传信息。单次最多列出1000个正在进行中的分块上传。

方法原型



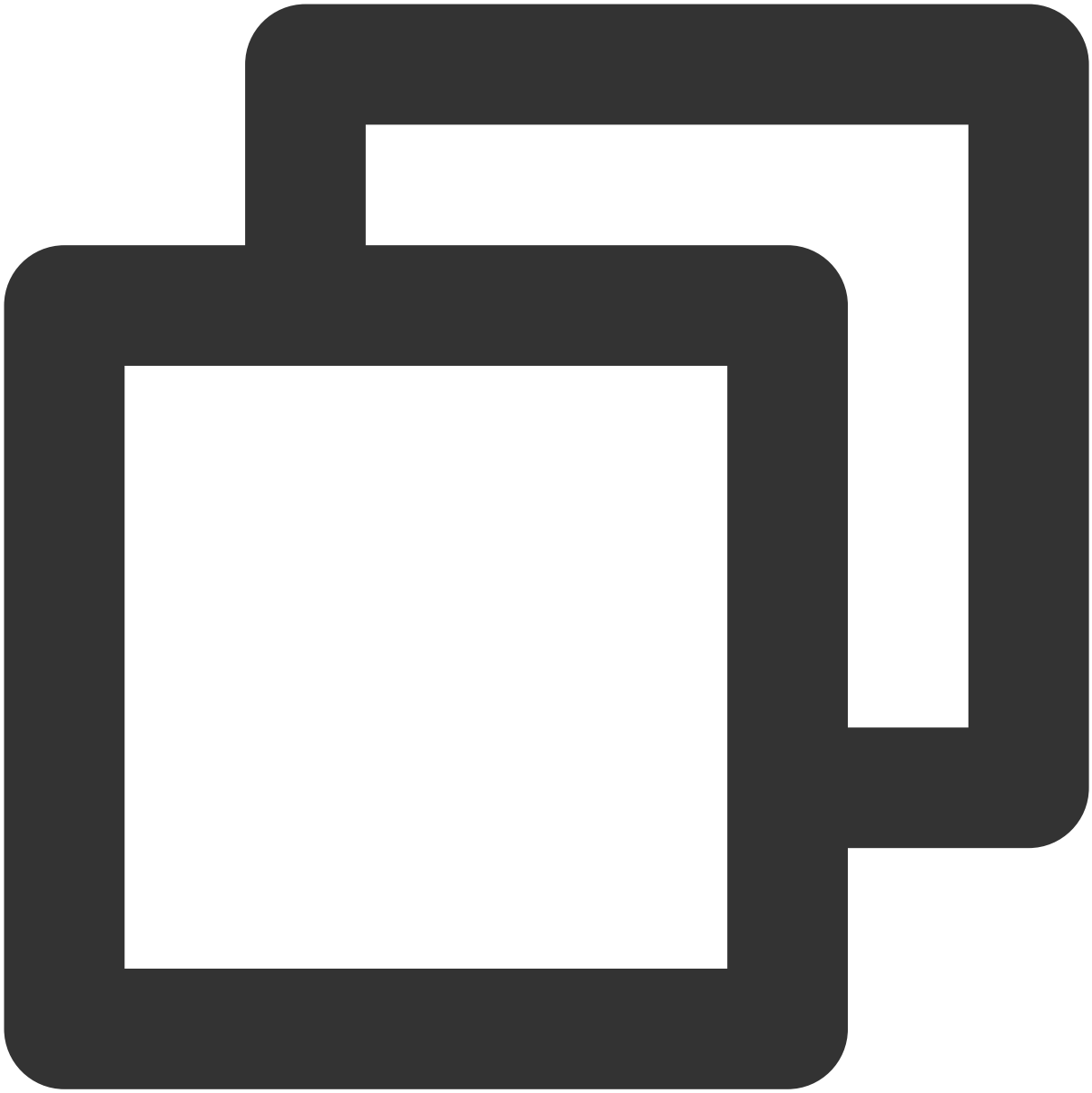
```
cos_status_t *cos_list_multipart_upload(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        cos_list_multipart_upload_params_t *params,
                                        cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
params	List Multipart Uploads 操作参数	Struct
encoding_type	规定返回值的编码方式	String
prefix	前缀匹配，用来规定返回的文件前缀地址	String
delimiter	界符为一个符号： 如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix 如果没有 Prefix，则从路径起点开始	String
max_ret	单次返回最大的条目数量，默认1000	String
key_marker	与 upload-id-marker 一起使用： 当 upload-id-marker 未被指定时，ObjectName 字母顺序大于 key-marker 的条目将被列出 当 upload-id-marker 被指定时，ObjectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出	String
upload_id_marker	与 key-marker 一起使用： 当 key-marker 未被指定时，upload-id-marker 将被忽略 当 key-marker 被指定时，ObjectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出	String
truncated	返回条目是否被截断，'true' 或者 'false'	Boolean
next_key_marker	假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点	String
next_upload_id_marker	假如返回条目被截断，则返回 UploadId 就是下一个条目的起点	String
upload_list	分块上传的信息	Struct
key	Object 的名称	String
upload_id	标示本次分块上传的 ID	String
initiated	标示本次分块上传任务的启动时间	String
resp_headers	返回 HTTP 响应消息的头域	Struct

版权所有：腾讯云计算（北京）有限责任公司



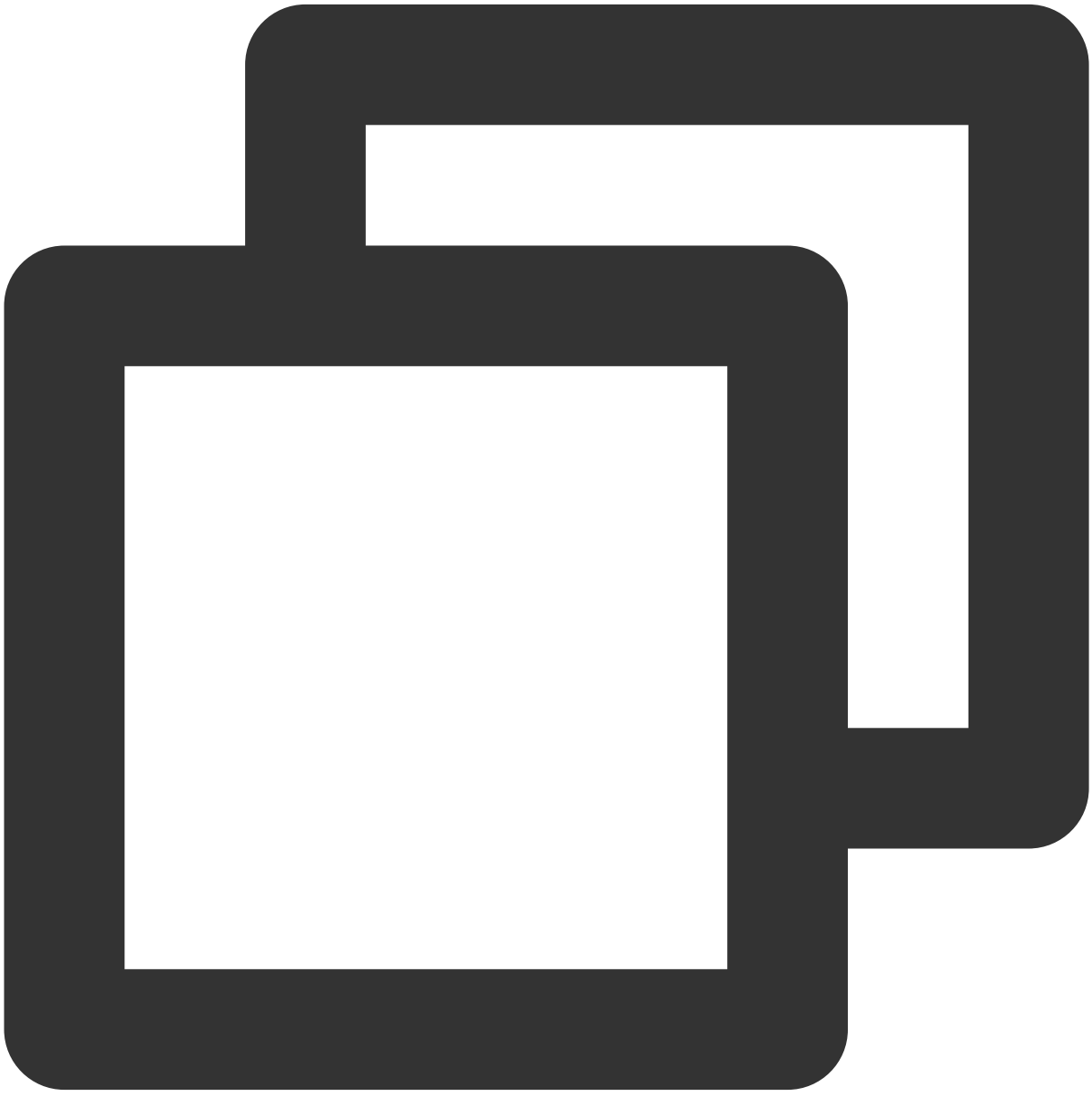
```
typedef struct {
    cos_list_t node;
    cos_string_t key;
    cos_string_t upload_id;
    cos_string_t initiated;
} cos_list_multipart_upload_content_t;
```

返回结果说明

返回结果	描述	类型

code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
```

```
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_OBJECT[] = "multipart.dat";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void list_multipart()
{
    cos_pool_t *p = NULL;
    cos_string_t bucket;
    cos_string_t object;
    int is_cname = 0;
    cos_table_t *resp_headers = NULL;
    cos_request_options_t *options = NULL;
    cos_status_t *s = NULL;
```

```
    cos_list_multipart_upload_params_t *list_multipart_params = NULL;
    cos_list_upload_part_params_t *list_upload_param = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    list_multipart_params = cos_create_list_multipart_upload_params(p);
    list_multipart_params->max_ret = 999;
    s = cos_list_multipart_upload(options, &bucket, list_multipart_params, &resp_he
    log_status(s);

    list_upload_param = cos_create_list_upload_part_params(p);
    list_upload_param->max_ret = 1000;
    cos_string_t upload_id;
    cos_str_set(&upload_id,"149373379126aee264fecbf5fe8ddb8b9cd23b76c73ab1af0bcfd50
    cos_str_set(&object, TEST_MULTIPART_OBJECT);
    s = cos_list_upload_part(options, &bucket, &object, &upload_id,
                             list_upload_param, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("List upload part succeeded, upload_id::%.*s\\n",
               upload_id.len, upload_id.data);
        cos_list_part_content_t *part_content = NULL;
        cos_list_for_each_entry(cos_list_part_content_t, part_content, &list_upload
            printf("part_number = %s, size = %s, last_modified = %s, etag = %s\\n",
                   part_content->part_number.data,
                   part_content->size.data,
                   part_content->last_modified.data,
                   part_content->etag.data);
        }
    } else {
        printf("List upload part failed\\n");
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }
```

```
    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    list_multipart();

    cos_http_io_deinitialize();

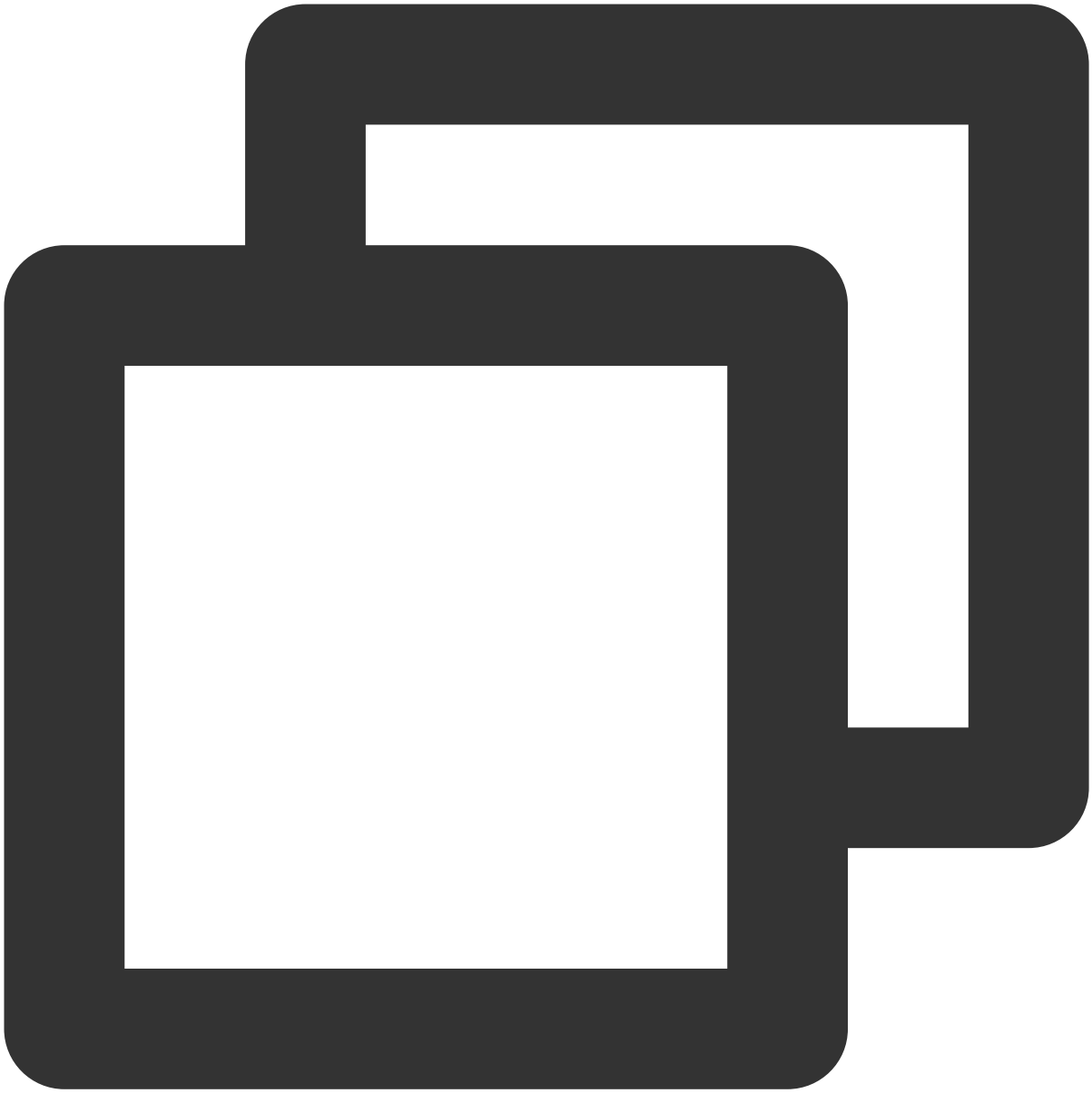
    return 0;
}
```

初始化分块上传

功能说明

Initiate Multipart Upload 请求实现初始化分片上传，成功执行此请求以后会返回 Upload ID 用于后续的 Upload Part 请求。

方法原型



```
cos_status_t *cos_init_multipart_upload(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        const cos_string_t *object,
                                        cos_string_t *upload_id,
                                        cos_table_t *headers,
                                        cos_table_t **resp_headers);
```

参数说明

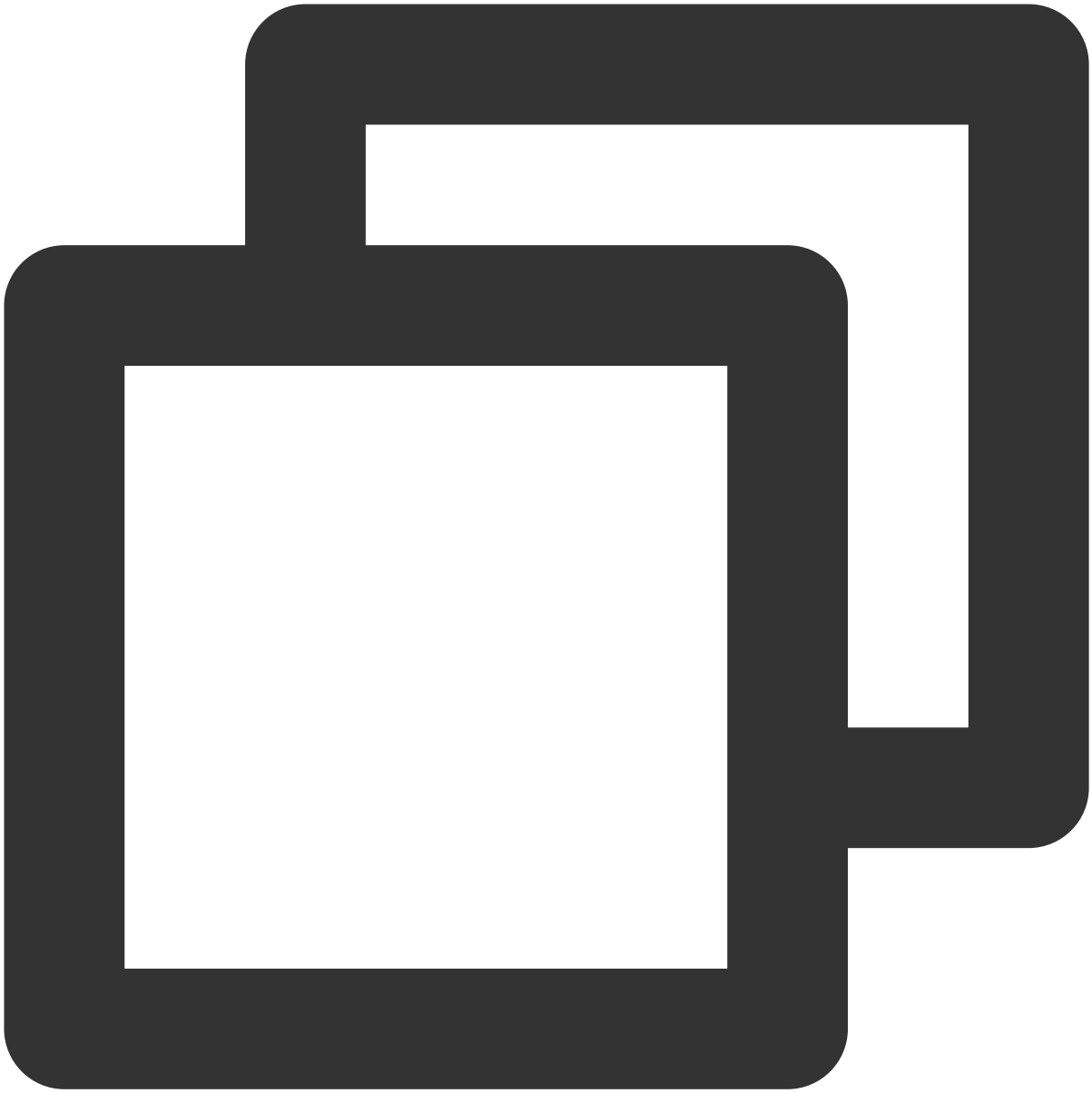
参数名称	参数描述	类型

options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
upload_id	操作返回的 Upload ID	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_OBJECT[] = "multipart.dat";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_init_multipart_upload()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t upload_id;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
```

```
    //初始化分块上传
    cos_str_set(&object, TEST_MULTIPART_OBJECT);
    s = cos_init_multipart_upload(options, &bucket, &object,
                                  &upload_id, NULL, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("init multipart upload succeeded\\n");
    } else {
        printf("init multipart upload failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_init_multipart_upload();

    cos_http_io_deinitialize();

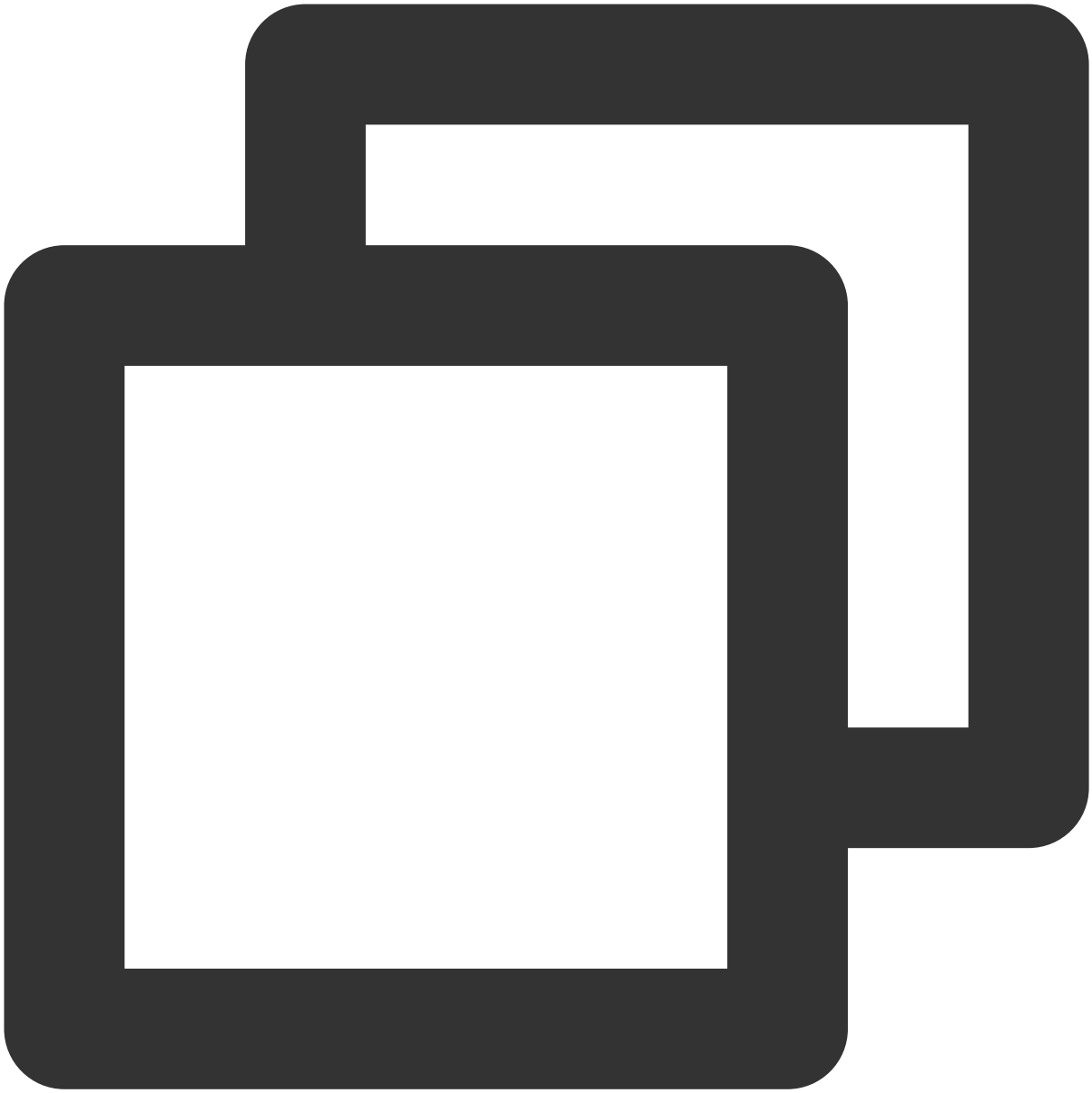
    return 0;
}
```

上传分块

功能说明

分块上传文件。Upload Part 请求实现在初始化以后的分块上传，支持的块的数量为1 - 10000，块的大小为1MB - 5GB。在每次请求 Upload Part 时，需要携带 partNumber 和 uploadID，partNumber 为块的编号，支持乱序上传。

方法原型



```
cos_status_t *cos_upload_part_from_file(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        const cos_string_t *object,
                                        const cos_string_t *upload_id,
                                        int part_num,
                                        cos_upload_file_t *upload_file,
                                        cos_table_t **resp_headers);
```

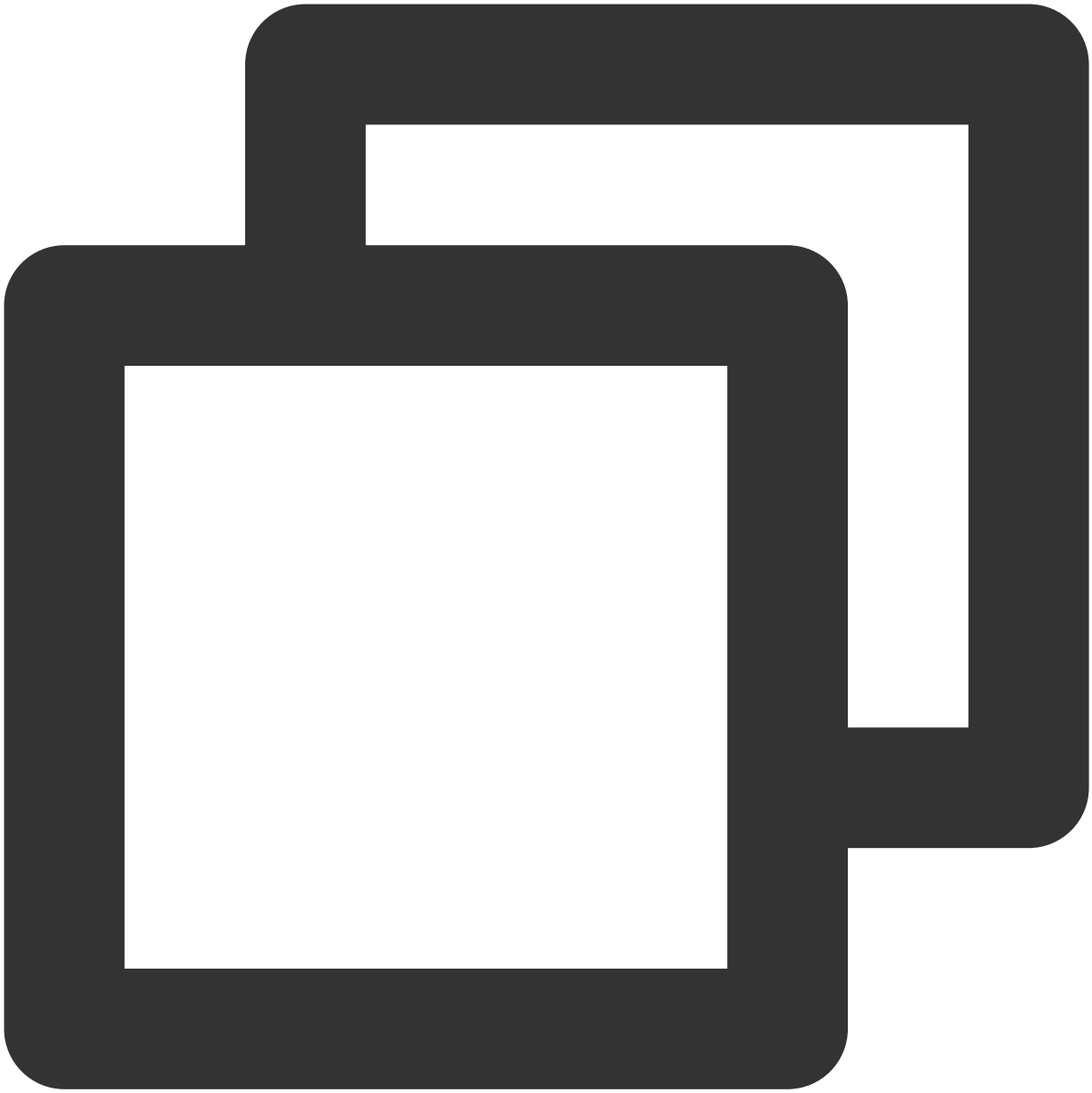
参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
upload_id	上传任务编号	String
part_num	分块编号	Int
upload_file	待上传本地文件信息	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_OBJECT[] = "multipart.dat";
static char TEST_MULTIPART_FILE[] = "test.zip";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_upload_part()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t upload_id = cos_string("xxxxxxxxxxxxxxxxxx");  // 替换您自己的uploa
    cos_table_t *resp_headers = NULL;
    int part_num = 1;
    int64_t pos = 0;
    int64_t file_length = 0;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
```

```
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_MULTIPART_OBJECT);

    //上传分块
    int res = COSE_OK;
    cos_upload_file_t *upload_file = NULL;
    cos_file_buf_t *fb = cos_create_file_buf(p);
    res = cos_open_file_for_all_read(p, TEST_MULTIPART_FILE, fb);
    if (res != COSE_OK) {
        cos_error_log("Open read file fail, filename:%s\\n", TEST_MULTIPART_FILE);
        return;
    }
    file_length = fb->file_last;
    apr_file_close(fb->file);
    while(pos < file_length) {
        upload_file = cos_create_upload_file(p);
        cos_str_set(&upload_file->filename, TEST_MULTIPART_FILE);
        upload_file->file_pos = pos;
        pos += 2 * 1024 * 1024;
        upload_file->file_last = pos < file_length ? pos : file_length; //2MB
        s = cos_upload_part_from_file(options, &bucket, &object, &upload_id,
                                      part_num++, upload_file, &resp_headers);

        if (cos_status_is_ok(s)) {
            printf("upload part succeeded\\n");
        } else {
            printf("upload part failed\\n");
        }
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
```

```
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_upload_part();

    cos_http_io_deinitialize();

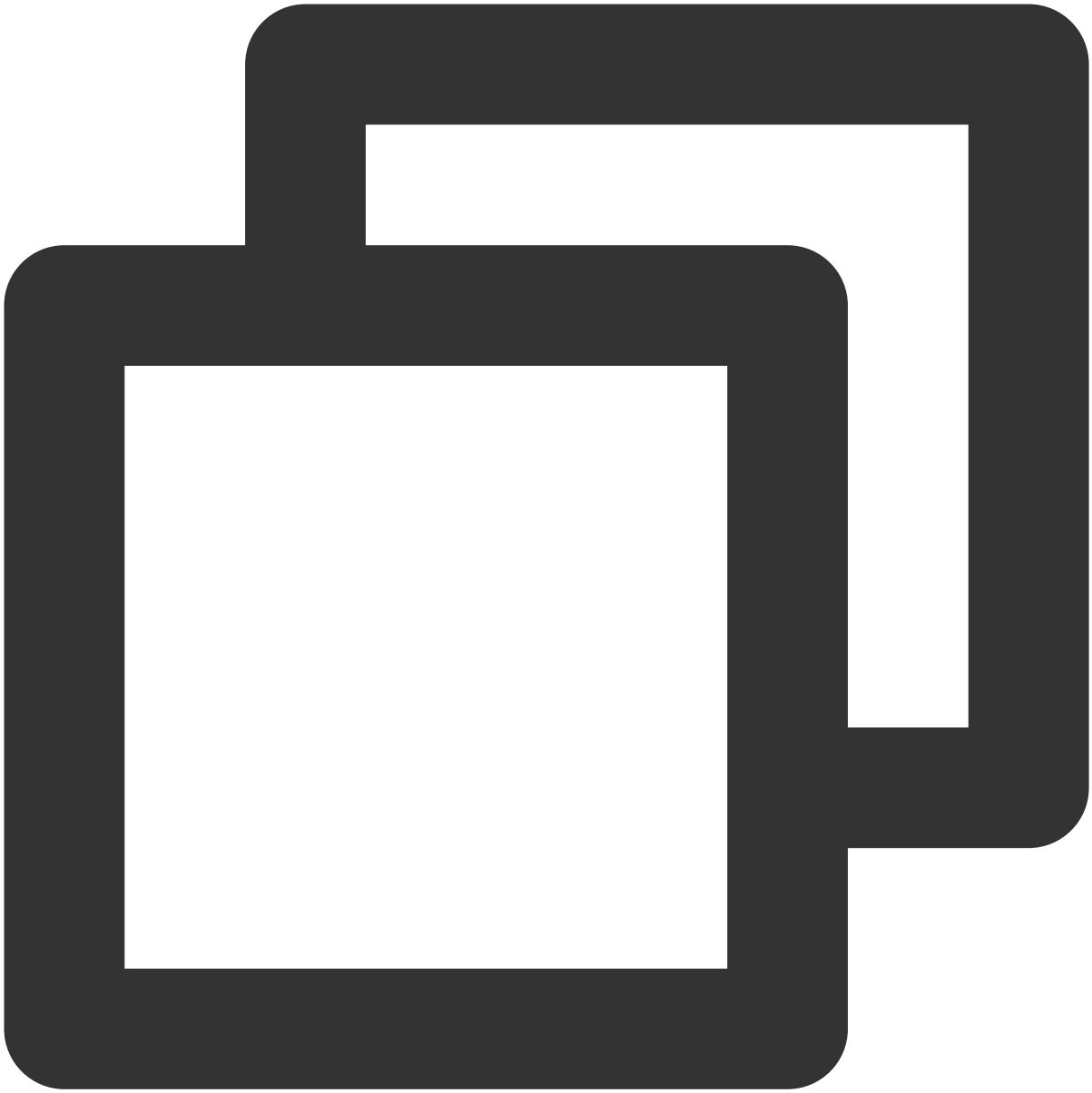
    return 0;
}
```

查询已上传块

功能说明

查询特定分块上传操作中的已上传的块。

方法原型



```
cos_status_t *cos_list_upload_part(const cos_request_options_t *options,
                                   const cos_string_t *bucket,
                                   const cos_string_t *object,
                                   const cos_string_t *upload_id,
                                   cos_list_upload_part_params_t *params,
                                   cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型

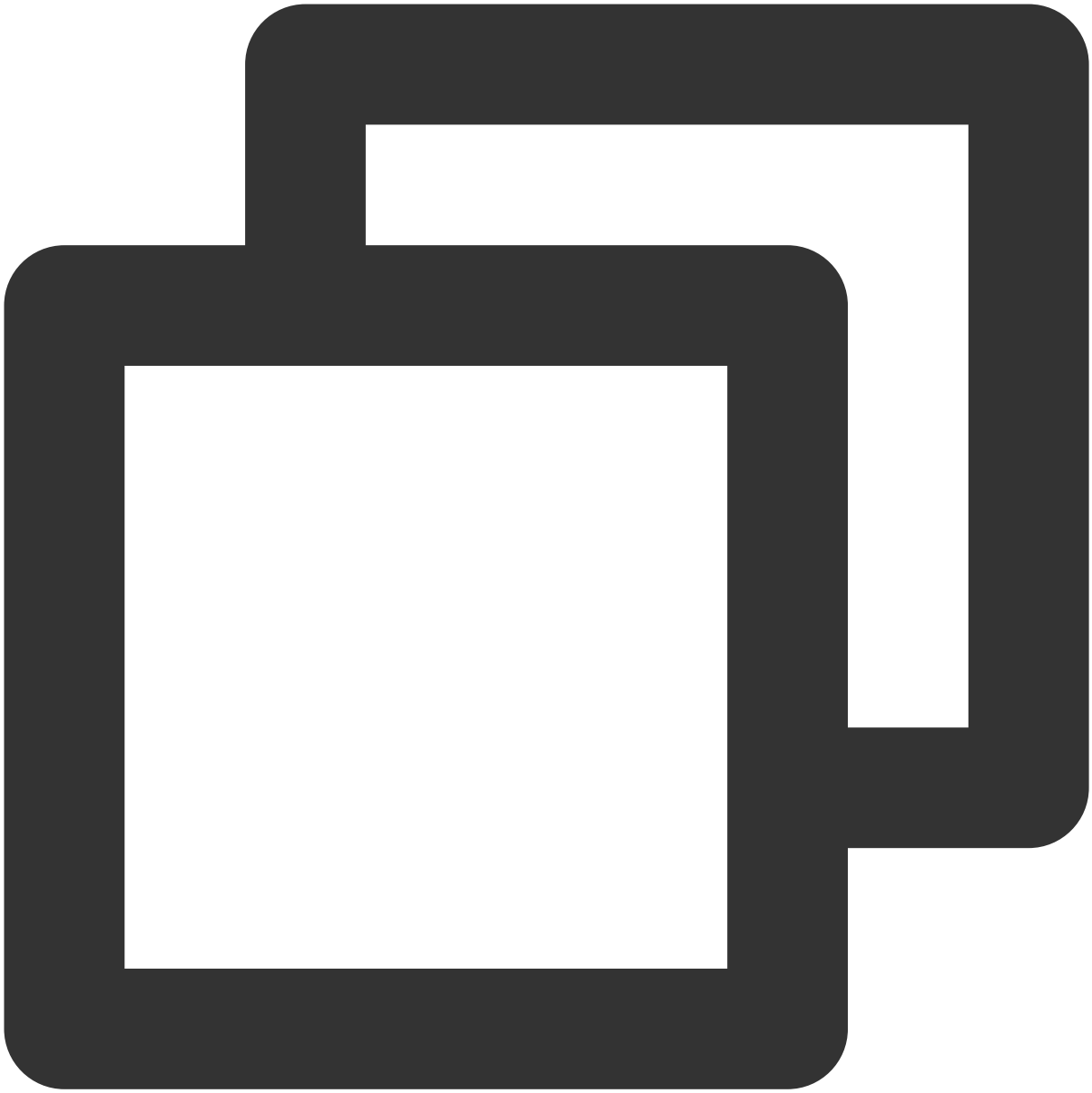
版权所有：腾讯云计算（北京）有限责任公司

options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
upload_id	上传任务编号	String
params	List Parts 操作参数	Struct
part_number_marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
encoding_type	规定返回值的编码方式	String
max_ret	单次返回最大的条目数量，默认1000	String
truncated	返回条目是否被截断，'true' 或者 'false'	Boolean
next_part_number_marker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
part_list	完成分块的信息	Struct
part_number	分块编号	String
size	分块大小，单位 Byte	String
etag	分块的 SHA-1 算法校验值	String
last_modified	分块最后修改时间	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_OBJECT[] = "multipart.dat";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_list_upload_part()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_list_upload_part_params_t *params = NULL;
    cos_list_t complete_part_list;
    cos_string_t upload_id = cos_string("xxxxxxxxxxxxxxxxxxx");  // 替换您自己的uploa
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
```

```
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_MULTIPART_OBJECT);

    //查询已上传块
    params = cos_create_list_upload_part_params(p);
    params->max_ret = 1000;
    cos_list_init(&complete_part_list);
    s = cos_list_upload_part(options, &bucket, &object, &upload_id,
                             params, &resp_headers);

    if (cos_status_is_ok(s)) {
        printf("List multipart succeeded\\n");
    } else {
        printf("List multipart failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_list_upload_part();

    cos_http_io_deinitialize();

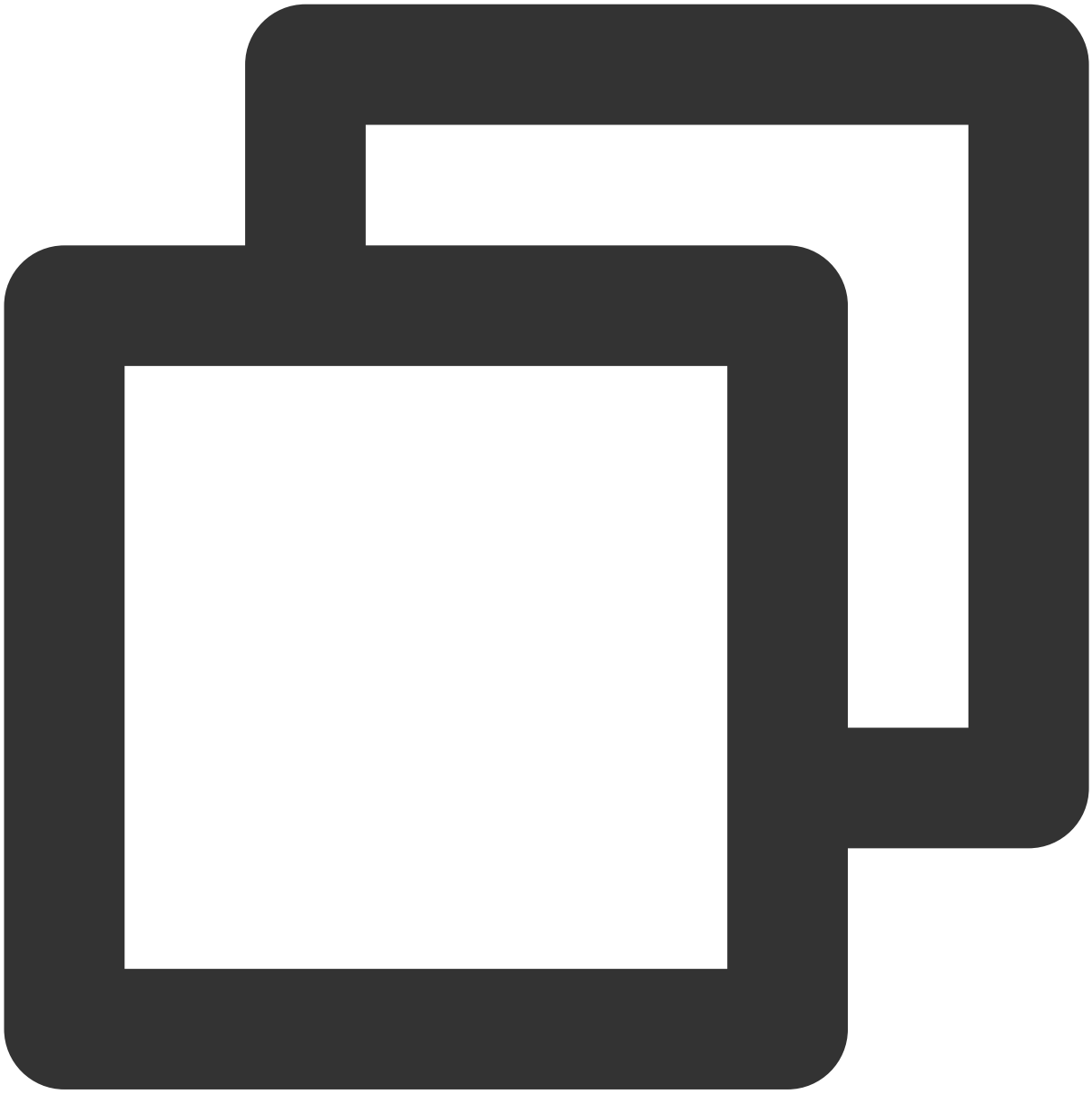
    return 0;
}
```

完成分块上传

功能说明

完成整个文件的分块上传。当您已经使用 Upload Parts 上传所有块以后，您可以用该 API 完成上传。在使用该 API 时，您必须在 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。

方法原型



```
cos_status_t *cos_complete_multipart_upload(const cos_request_options_t *options,
                                            const cos_string_t *bucket,
                                            const cos_string_t *object,
                                            const cos_string_t *upload_id,
                                            cos_list_t *part_list,
                                            cos_table_t *headers,
```

```
                                                cos_table_t **resp_headers);
```

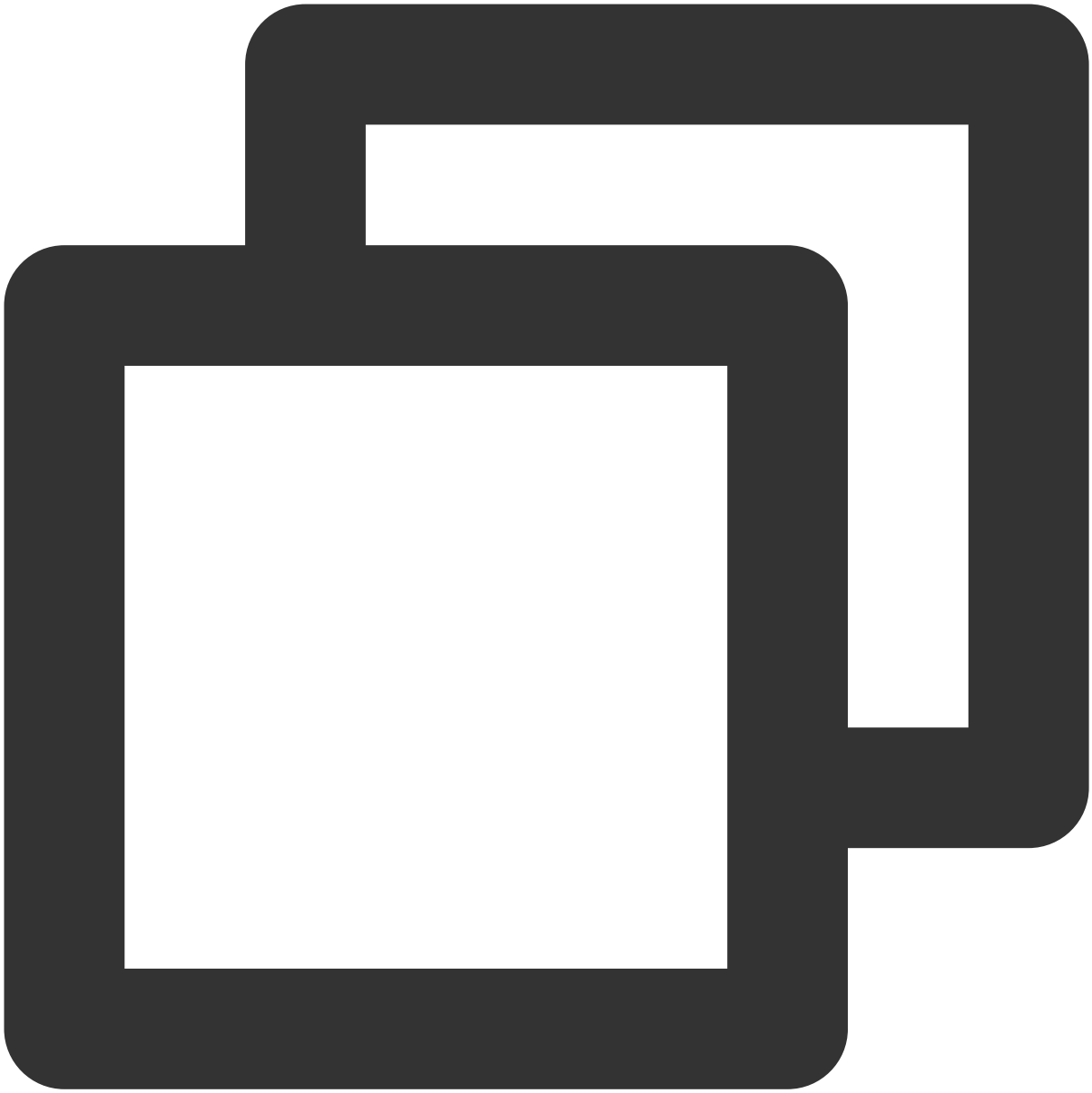
参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
upload_id	上传任务编号	String
part_list	完成分块上传的参数	Struct
part_number	分块编号	String
etag	分块的 ETag 值，为 sha1 校验值，需要在校验值前后加上双引号，如 "3a0f1fd698c235af9cf098cb74aa25bc"	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_OBJECT[] = "multipart.dat";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_complete_multipart_upload()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_list_upload_part_params_t *params = NULL;
    cos_list_t complete_part_list;
    cos_string_t upload_id = cos_string("xxxxxxxxxxxxxxxxxx");  // 替换您自己的uploa
    cos_table_t *resp_headers = NULL;
    cos_list_part_content_t *part_content = NULL;
    cos_complete_part_content_t *complete_part_content = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
```

```
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_MULTIPART_OBJECT);

    //查询已上传分块
    params = cos_create_list_upload_part_params(p);
    params->max_ret = 1000;
    cos_list_init(&complete_part_list);
    s = cos_list_upload_part(options, &bucket, &object, &upload_id,
                             params, &resp_headers);

    if (cos_status_is_ok(s)) {
        printf("List multipart succeeded\\n");
    } else {
        printf("List multipart failed\\n");
        cos_pool_destroy(p);
        return;
    }

    cos_list_for_each_entry(cos_list_part_content_t, part_content, &params->part_li
        complete_part_content = cos_create_complete_part_content(p);
        cos_str_set(&complete_part_content->part_number, part_content->part_number.
        cos_str_set(&complete_part_content->etag, part_content->etag.data);
        cos_list_add_tail(&complete_part_content->node, &complete_part_list);
    }

    //完成分块上传
    s = cos_complete_multipart_upload(options, &bucket, &object, &upload_id,
                                      &complete_part_list, NULL, &resp_headers);

    if (cos_status_is_ok(s)) {
        printf("Complete multipart upload from file succeeded, upload_id:%.*s\\n",
            upload_id.len, upload_id.data);
    } else {
        printf("Complete multipart upload from file failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");
```

```
    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
        exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_complete_multipart_upload();

    cos_http_io_deinitialize();

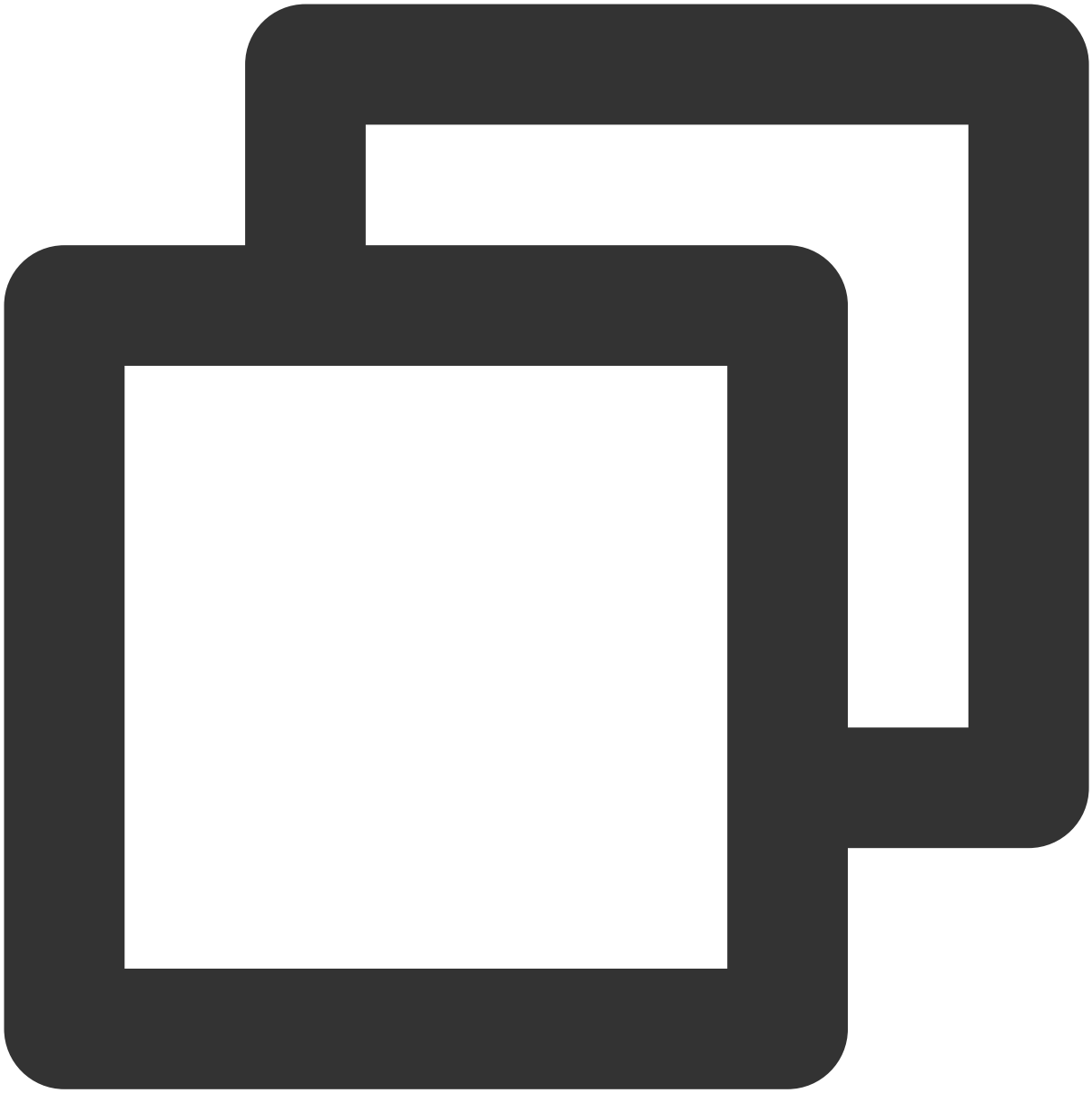
    return 0;
}
```

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块。当您调用 Abort Multipart Upload 时，如果有正在使用这个 Upload Parts 上传块的请求，则 Upload Parts 会返回失败。

方法原型



```
cos_status_t *cos_abort_multipart_upload(const cos_request_options_t *options,
                                         const cos_string_t *bucket,
                                         const cos_string_t *object,
                                         cos_string_t *upload_id,
                                         cos_table_t **resp_headers);
```

参数说明

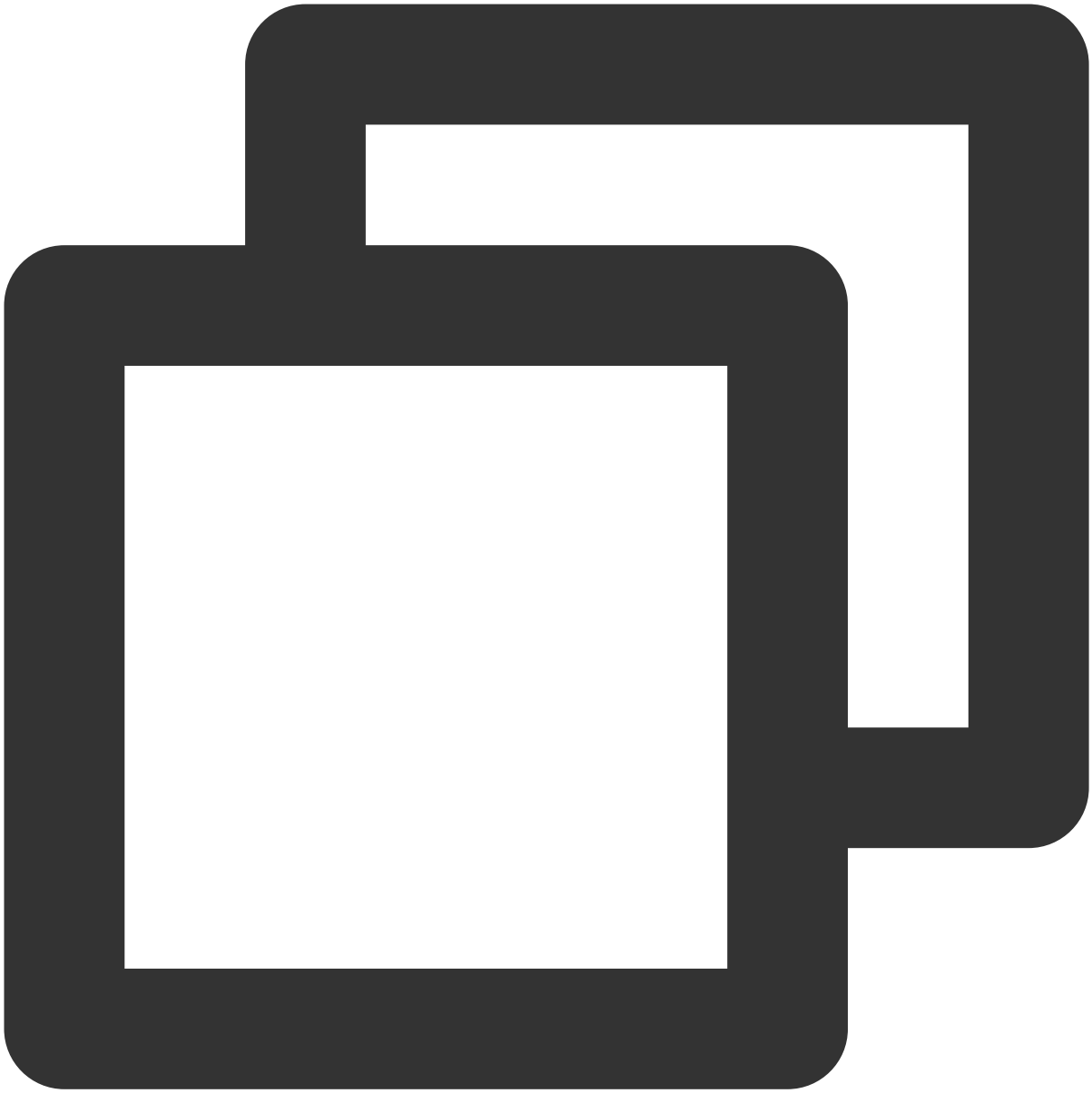
参数名称	参数描述	类型

options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
upload_id	上传任务编号	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_OBJECT[] = "multipart.dat";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void abort_multipart_upload()
{
    cos_pool_t *p = NULL;
    cos_string_t bucket;
    cos_string_t object;
    int is_cname = 0;
    cos_table_t *headers = NULL;
    cos_table_t *resp_headers = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t upload_id;
    cos_status_t *s = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    headers = cos_table_make(p, 1);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_MULTIPART_OBJECT);
```

```
    s = cos_init_multipart_upload(options, &bucket, &object,
                                  &upload_id, headers, &resp_headers);

    if (cos_status_is_ok(s)) {
        printf("Init multipart upload succeeded, upload_id:%.*s\\n",
               upload_id.len, upload_id.data);
    } else {
        printf("Init multipart upload failed\\n");
        cos_pool_destroy(p);
        return;
    }

    s = cos_abort_multipart_upload(options, &bucket, &object, &upload_id,
                                   &resp_headers);

    if (cos_status_is_ok(s)) {
        printf("Abort multipart upload succeeded, upload_id::%.*s\\n",
               upload_id.len, upload_id.data);
    } else {
        printf("Abort multipart upload failed\\n");
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    abort_multipart_upload();

    cos_http_io_deinitialize();

    return 0;
}
```

版权所有：腾讯云计算（北京）有限责任公司

复制与移动对象

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于对象的复制与移动 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object - Copy	设置对象复制（修改属性）	复制文件到目标路径
DELETE Object	删除单个对象	在存储桶中删除指定对象

分块操作

API	操作名	操作描述
Upload Part - Copy	复制分块	将其他对象复制为一个分块

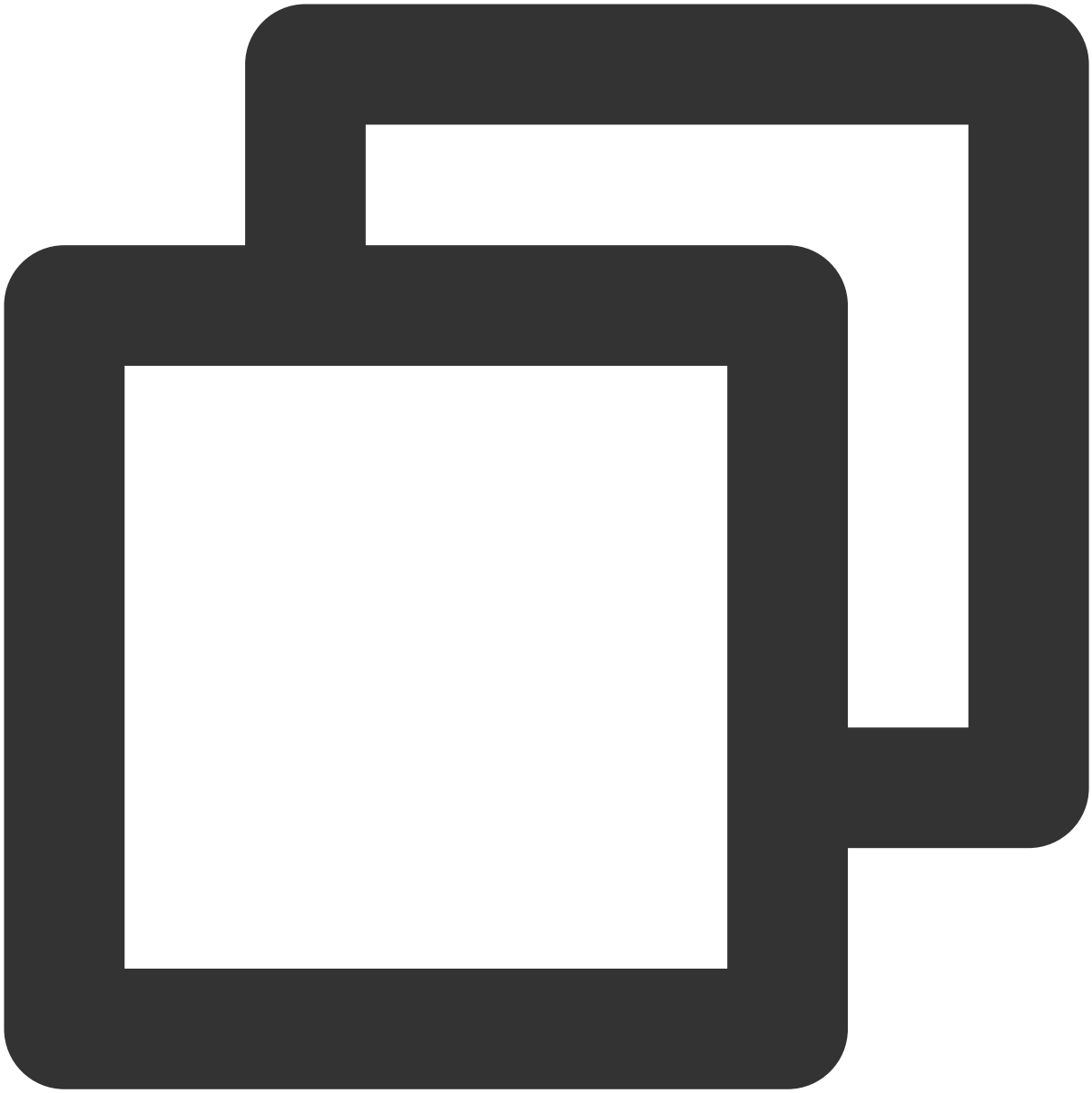
简单复制

设置对象复制（修改属性）

功能说明

复制文件到目标路径。该接口还可以修改对象属性，例如修改对象的存储类型。

方法原型



```
cos_status_t *cos_copy_object(const cos_request_options_t *options,
                              const cos_string_t *src_bucket,
                              const cos_string_t *src_object,
                              const cos_string_t *src_endpoint,
                              const cos_string_t *dest_bucket,
                              const cos_string_t *dest_object,
                              cos_table_t *headers,
                              cos_copy_object_params_t *copy_object_param,
                              cos_table_t **resp_headers);
```

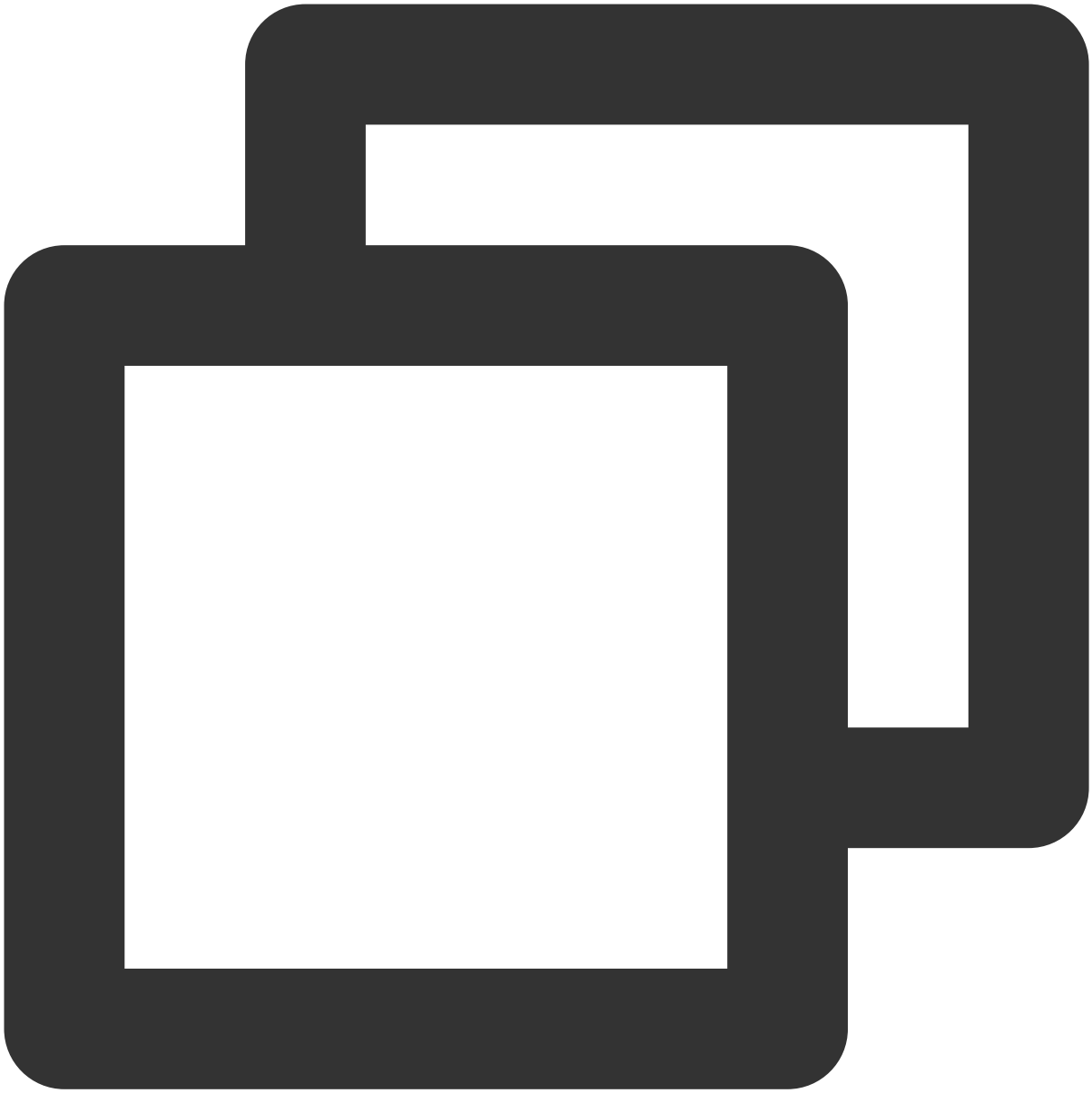
参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
src_bucket	源存储桶	String
src_object	源对象名称	String
src_endpoint	源对象访问域名信息	String
dest_bucket	目的存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
dest_object	目的 Object 名称	String
headers	COS 请求附加头域	Struct
copy_object_param	Put Object Copy 操作参数	Struct
etag	返回文件的 MD5 算法校验值	String
last_modify	返回文件最后修改时间，GMT 格式	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例1：普通对象复制



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";
static char TEST_OBJECT_NAME2[] = "test2.dat";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_copy()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t src_bucket;
    cos_string_t src_object;
    cos_string_t src_endpoint;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //设置对象复制
    cos_str_set(&object, TEST_OBJECT_NAME2);
    cos_str_set(&src_bucket, TEST_BUCKET_NAME);
    cos_str_set(&src_endpoint, TEST_COS_ENDPOINT);
```

```
    cos_str_set(&src_object, TEST_OBJECT_NAME1);

    cos_copy_object_params_t *params = NULL;
    params = cos_create_copy_object_params(p);
    s = cos_copy_object(options, &src_bucket, &src_object, &src_endpoint, &bucket,
    if (cos_status_is_ok(s)) {
        printf("put object copy succeeded\\n");
    } else {
        printf("put object copy failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_copy();

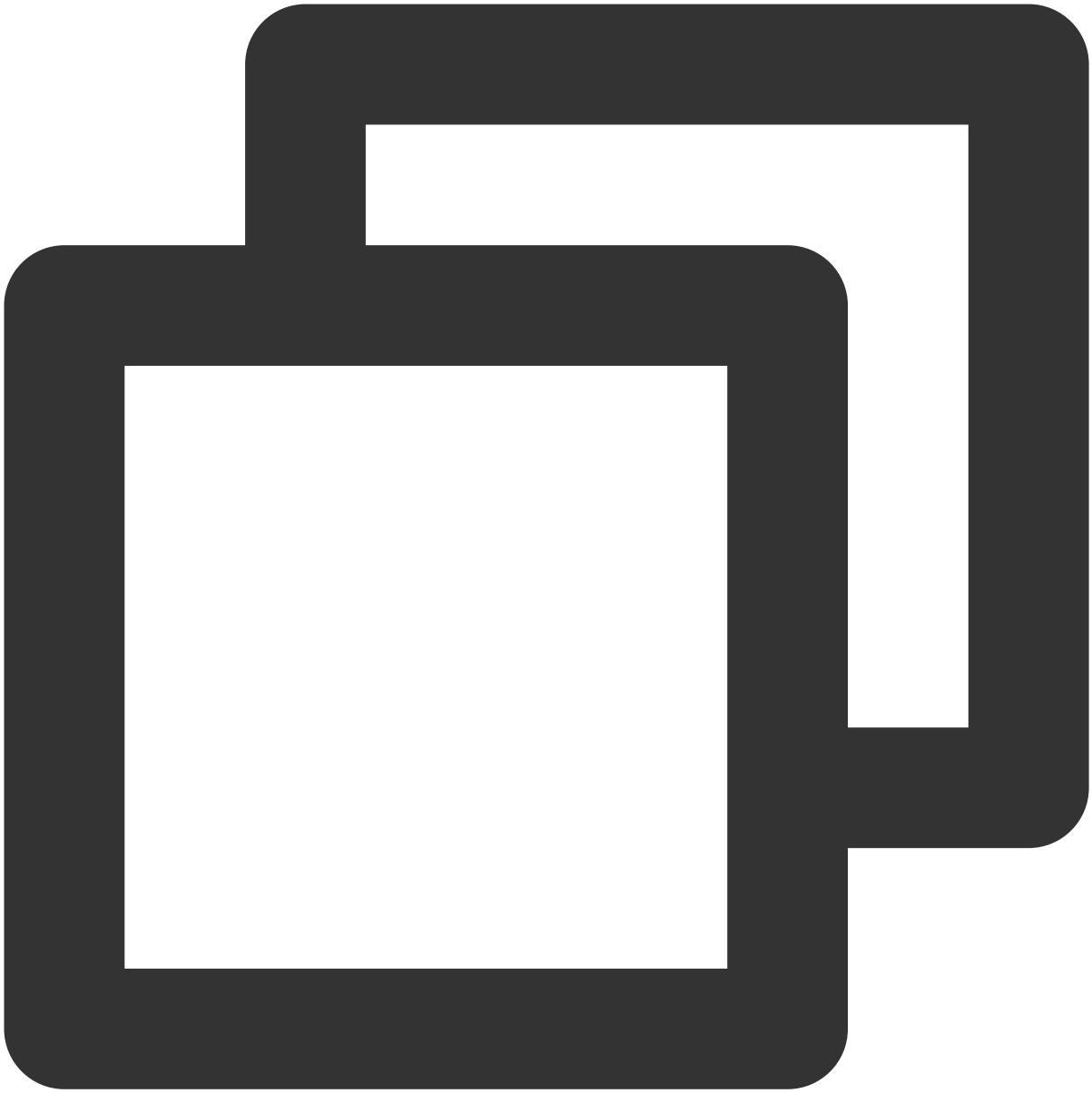
    cos_http_io_deinitialize();

    return 0;
}
```

示例2：修改存储类型

注意

标准存储可以修改为低频存储、智能分层存储、归档存储和深度归档存储等，如果希望将归档存储或深度归档存储的对象修改为其他存储类型，首先需要使用 cos_post_object_restore() 将归档存储或深度归档存储的对象回热，才能使用该接口请求修改存储类型；关于存储类型的详细说明请参见 存储类型概述。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_modify_storage_class()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t src_bucket;
    cos_string_t src_object;
    cos_string_t src_endpoint;
    cos_table_t *resp_headers = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_OBJECT_NAME1);
```

```
    cos_str_set(&src_bucket, TEST_BUCKET_NAME);
    cos_str_set(&src_endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&src_object, TEST_OBJECT_NAME1);

    // 设置x-cos-metadata-directive和x-cos-storage-class头域(替换为自己要更改的存储类型)
    cos_table_t *headers = cos_table_make(p, 2);
    apr_table_add(headers, "x-cos-metadata-directive", "Replaced");
    // 存储类型包括NTELLIGENT_TIERING, MAZ_INTELLIGENT_TIERING, STANDARD_IA, ARCHIVE, I
    apr_table_add(headers, "x-cos-storage-class", "ARCHIVE");

    cos_copy_object_params_t *params = NULL;
    params = cos_create_copy_object_params(p);
    s = cos_copy_object(options, &src_bucket, &src_object, &src_endpoint, &bucket,
    log_status(s);

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_modify_storage_class();

    cos_http_io_deinitialize();

    return 0;
}
```

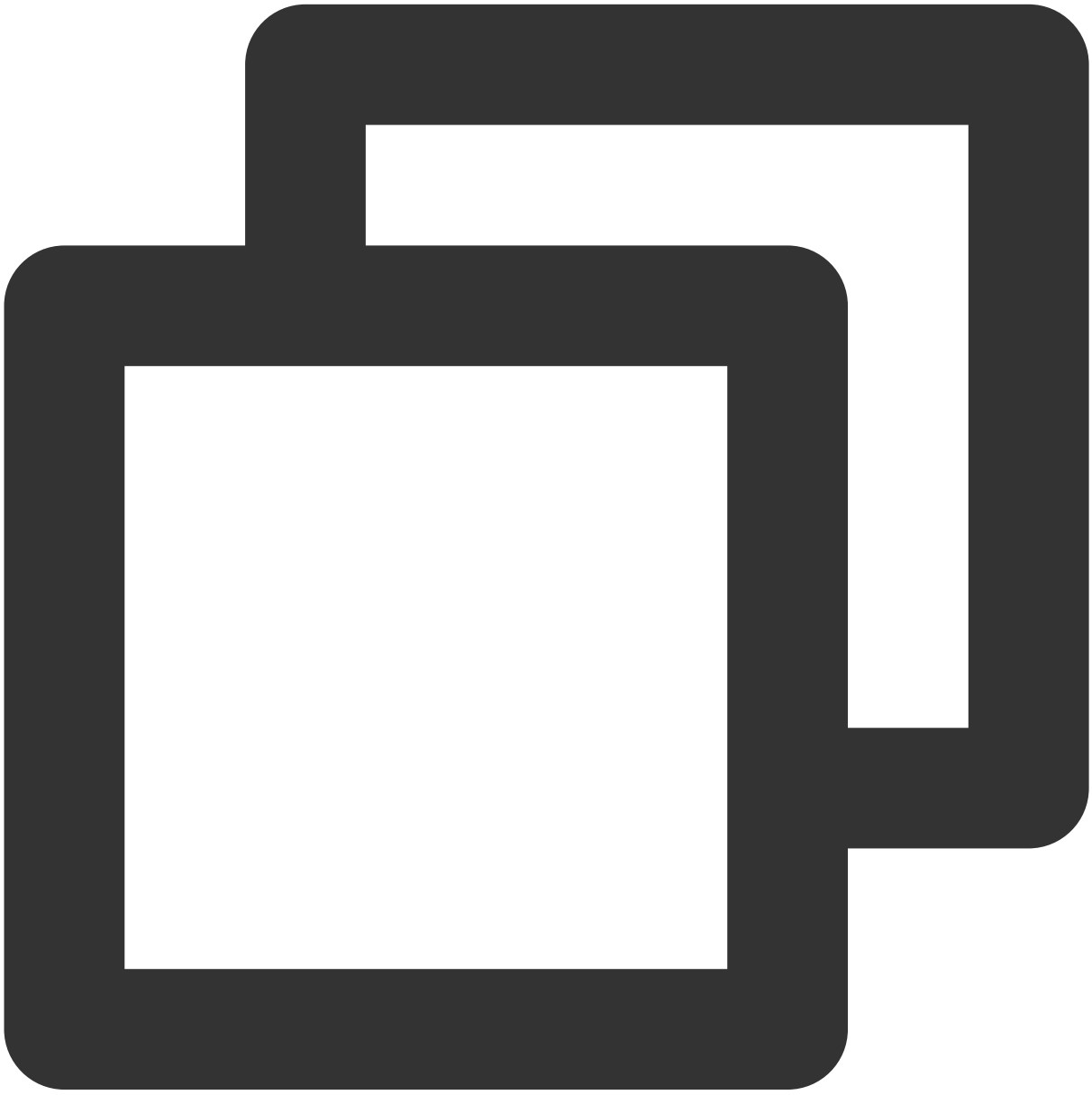
分块复制

复制分块

功能说明

将其他对象复制为一个分块。

方法原型



```
cos_status_t *cos_upload_part_copy(const cos_request_options_t *options,
                                   cos_upload_part_copy_params_t *params,
                                   cos_table_t *headers,
                                   cos_table_t **resp_headers);
```

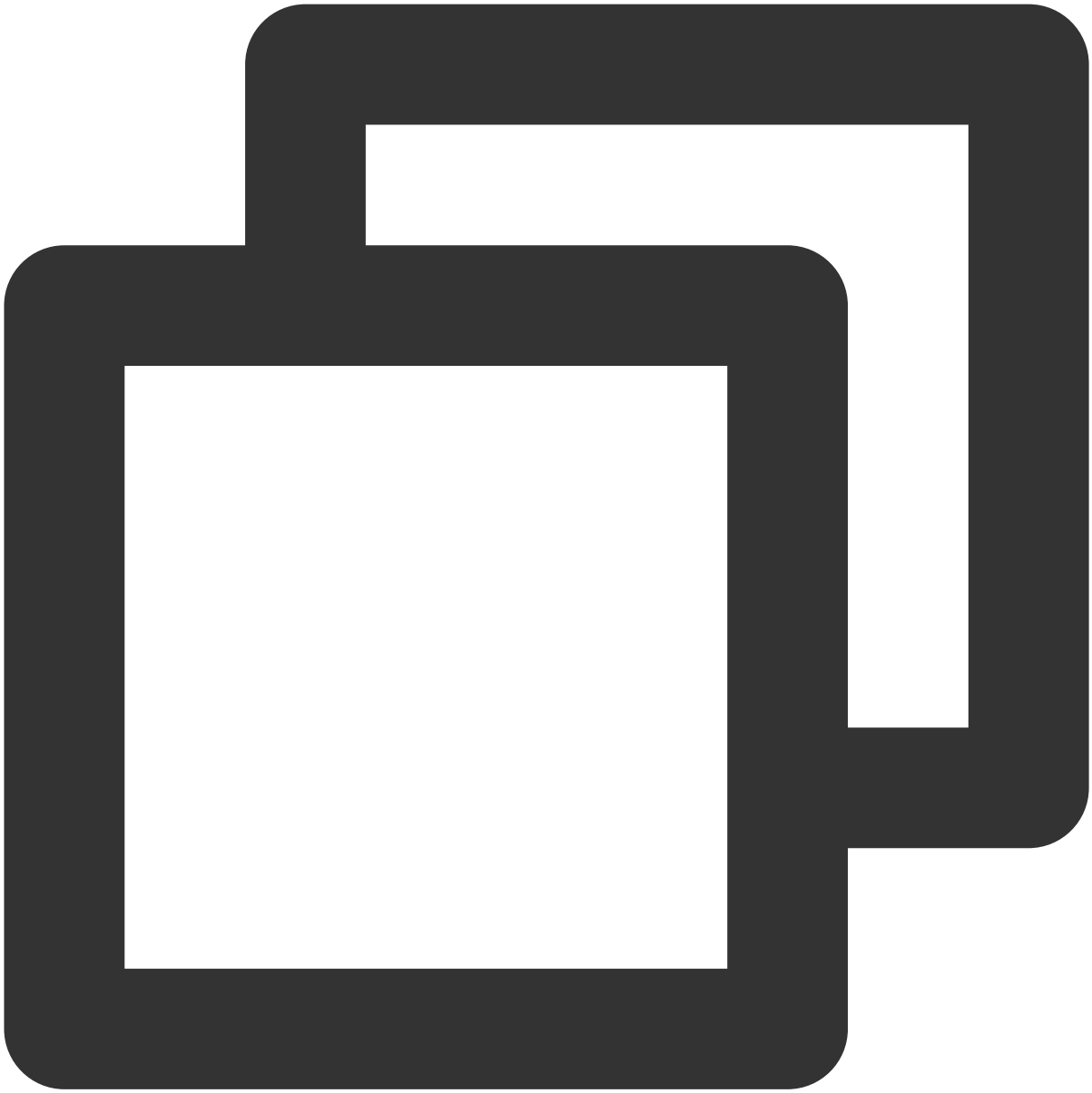
参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
params	复制分块参数信息	Struct
copy_source	源文件路径	String
dest_bucket	目的 存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
dest_object	目的 Object 名称	String
upload_id	上传任务编号	String
part_num	分块编号	Int
range_start	源文件起始偏移	Int
range_end	源文件终止偏移	Int
rsp_content	复制分块结果信息	Struct
etag	返回文件的 MD5 算法校验值	String
last_modify	返回文件最后修改时间，GMT 格式	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <sys/stat.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void make_rand_string(cos_pool_t *p, int len, cos_string_t *data)
{
    char *str = NULL;
    int i = 0;
    str = (char *)cos_palloc(p, len + 1);
    for ( ; i < len; i++) {
        str[i] = 'a' + rand() % 32;
    }
    str[len] = '\\0';
    cos_str_set(data, str);
}

unsigned long get_file_size(const char *file_path)
{
    unsigned long filesize = -1;
    struct stat statbuff;

    if(stat(file_path, &statbuff) < 0){
        return filesize;
```

```
    } else {
        filesize = statbuff.st_size;
    }

    return filesize;
}

void test_part_copy()
{
    cos_pool_t *p = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t file;
    int is_cname = 0;
    cos_string_t upload_id;
    cos_list_upload_part_params_t *list_upload_part_params = NULL;
    cos_upload_part_copy_params_t *upload_part_copy_params1 = NULL;
    cos_upload_part_copy_params_t *upload_part_copy_params2 = NULL;
    cos_table_t *headers = NULL;
    cos_table_t *query_params = NULL;
    cos_table_t *resp_headers = NULL;
    cos_table_t *list_part_resp_headers = NULL;
    cos_list_t complete_part_list;
    cos_list_part_content_t *part_content = NULL;
    cos_complete_part_content_t *complete_content = NULL;
    cos_table_t *complete_resp_headers = NULL;
    cos_status_t *s = NULL;
    int part1 = 1;
    int part2 = 2;
    char *local_filename = "test_upload_part_copy.file";
    char *download_filename = "test_upload_part_copy.file.download";
    char *source_object_name = "cos_test_upload_part_copy_source_object";
    char *dest_object_name = "cos_test_upload_part_copy_dest_object";
    FILE *fd = NULL;
    cos_string_t download_file;
    cos_string_t dest_bucket;
    cos_string_t dest_object;
    int64_t range_start1 = 0;
    int64_t range_end1 = 6000000;
    int64_t range_start2 = 6000001;
    int64_t range_end2;
    cos_string_t data;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
```

```
    // create multipart upload local file
    make_rand_string(p, 10 * 1024 * 1024, &data);
    fd = fopen(local_filename, "w");
    fwrite(data.data, sizeof(data.data[0]), data.len, fd);
    fclose(fd);

    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, source_object_name);
    cos_str_set(&file, local_filename);
    s = cos_put_object_from_file(options, &bucket, &object, &file, NULL, &resp_head
    log_status(s);

    //init mulitipart
    cos_str_set(&object, dest_object_name);
    s = cos_init_multipart_upload(options, &bucket, &object,
                                  &upload_id, NULL, &resp_headers);
    log_status(s);

    //upload part copy 1
    upload_part_copy_params1 = cos_create_upload_part_copy_params(p);
    cos_str_set(&upload_part_copy_params1->copy_source, "bucket-appid.cn-south.myqc
    cos_str_set(&upload_part_copy_params1->dest_bucket, TEST_BUCKET_NAME);
    cos_str_set(&upload_part_copy_params1->dest_object, dest_object_name);
    cos_str_set(&upload_part_copy_params1->upload_id, upload_id.data);
    upload_part_copy_params1->part_num = part1;
    upload_part_copy_params1->range_start = range_start1;
    upload_part_copy_params1->range_end = range_end1;
    headers = cos_table_make(p, 0);
    s = cos_upload_part_copy(options, upload_part_copy_params1, headers, &resp_head
    log_status(s);
    printf("last modified:%s, etag:%s\\n", upload_part_copy_params1->rsp_content->l

    //upload part copy 2
    resp_headers = NULL;
    range_end2 = get_file_size(local_filename) - 1;
    upload_part_copy_params2 = cos_create_upload_part_copy_params(p);
    cos_str_set(&upload_part_copy_params2->copy_source, "bucket-appid.cn-south.myqc
    cos_str_set(&upload_part_copy_params2->dest_bucket, TEST_BUCKET_NAME);
    cos_str_set(&upload_part_copy_params2->dest_object, dest_object_name);
    cos_str_set(&upload_part_copy_params2->upload_id, upload_id.data);
    upload_part_copy_params2->part_num = part2;
    upload_part_copy_params2->range_start = range_start2;
    upload_part_copy_params2->range_end = range_end2;
    headers = cos_table_make(p, 0);
    s = cos_upload_part_copy(options, upload_part_copy_params2, headers, &resp_head
    log_status(s);
```

```
    printf("last modified:%s, etag:%s\\n", upload_part_copy_params1->rsp_content->l

    //list part
    list_upload_part_params = cos_create_list_upload_part_params(p);
    list_upload_part_params->max_ret = 10;
    cos_list_init(&complete_part_list);

    cos_str_set(&dest_bucket, TEST_BUCKET_NAME);
    cos_str_set(&dest_object, dest_object_name);
    s = cos_list_upload_part(options, &dest_bucket, &dest_object, &upload_id,
                             list_upload_part_params, &list_part_resp_headers);
    log_status(s);
    cos_list_for_each_entry(cos_list_part_content_t, part_content, &list_upload_par
        complete_content = cos_create_complete_part_content(p);
        cos_str_set(&complete_content->part_number, part_content->part_number.data)
        cos_str_set(&complete_content->etag, part_content->etag.data);
        cos_list_add_tail(&complete_content->node, &complete_part_list);
    }

    //complete multipart
    headers = cos_table_make(p, 0);
    s = cos_complete_multipart_upload(options, &dest_bucket, &dest_object,
            &upload_id, &complete_part_list, headers, &complete_resp_headers);
    log_status(s);

    //check upload copy part content equal to local file
    headers = cos_table_make(p, 0);
    cos_str_set(&download_file, download_filename);
    s = cos_get_object_to_file(options, &dest_bucket, &dest_object, headers,
                               query_params, &download_file, &resp_headers);
    log_status(s);
    printf("local file len = %"APR_INT64_T_FMT", download file len = %"APR_INT64_T_
    remove(download_filename);
    remove(local_filename);
    cos_pool_destroy(p);

    printf("test part copy ok\\n");
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
```

```
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_part_copy();

    cos_http_io_deinitialize();

    return 0;
}
```

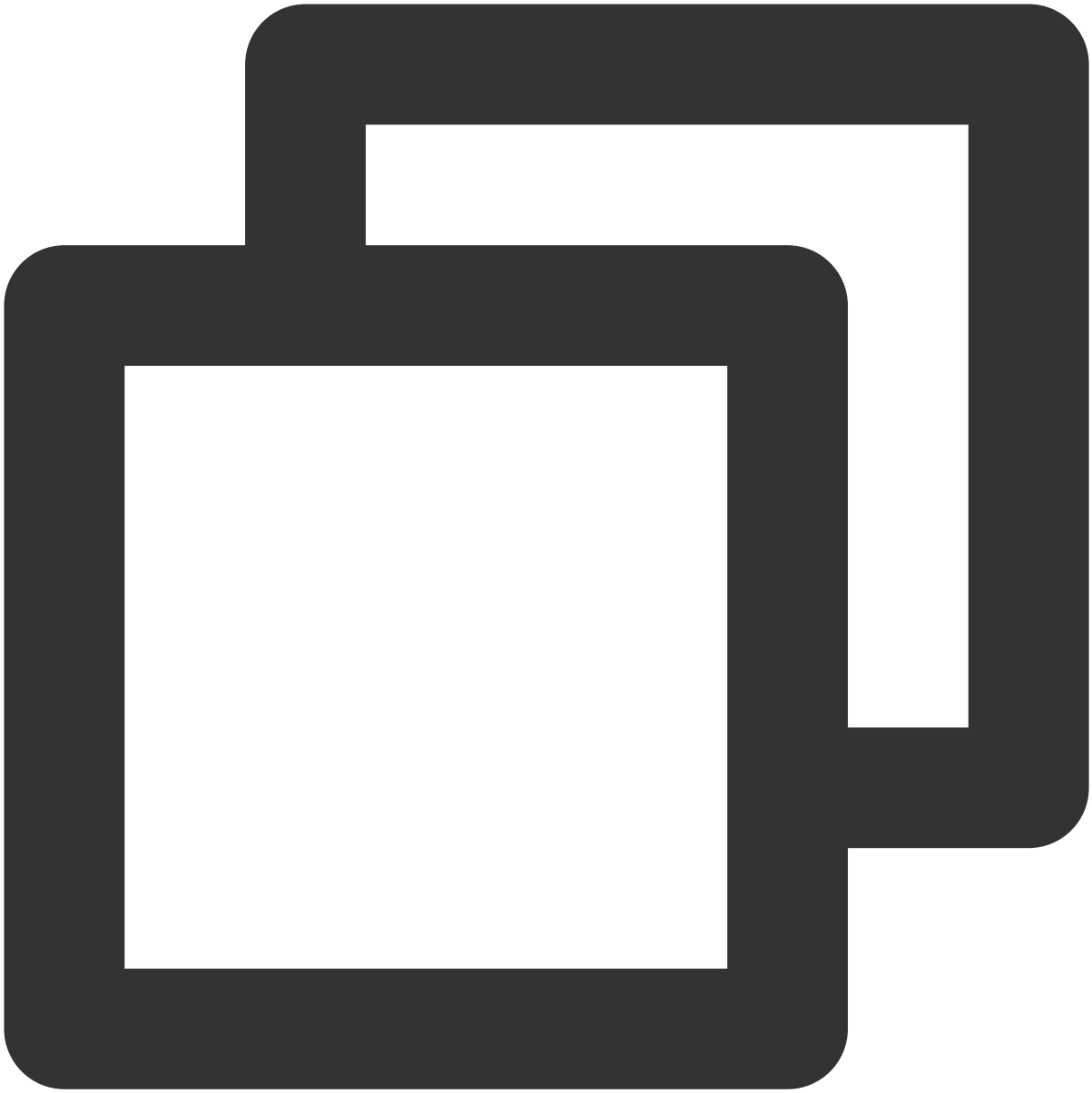
移动对象

移动对象

功能说明

移动对象主要包括两个操作：复制源对象到目标位置，删除源对象。
由于 COS 通过存储桶名称（Bucket）和对象键（ObjectKey）来标识对象。移动对象也就意味着修改这个对象的标识，COS C SDK 目前没有提供修改对象唯一标识名的单独接口，但是可以通过组合**复制对象**加上**删除对象**的基本操作，来达到修改对象标识的目的，从而实现移动对象。
复制对象方法详细说明请参见 设置对象复制
删除对象方法详细说明请参见：删除对象

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";
static char TEST_OBJECT_NAME2[] = "test2.dat";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_move()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t src_object;
    cos_string_t src_endpoint;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
```

```
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //设置对象复制
    cos_str_set(&object, TEST_OBJECT_NAME1);
    cos_str_set(&src_endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&src_object, TEST_OBJECT_NAME2);

    cos_copy_object_params_t *params = NULL;
    params = cos_create_copy_object_params(p);
    s = cos_copy_object(options, &bucket, &src_object, &src_endpoint, &bucket, &obj
    log_status(s);
    if (cos_status_is_ok(s)) {
        s = cos_delete_object(options, &bucket, &src_object, &resp_headers);
        log_status(s);
        printf("move object succeeded\\n");
    } else {
        printf("move object failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_move();

    cos_http_io_deinitialize();

    return 0;
}
```

下载对象

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地

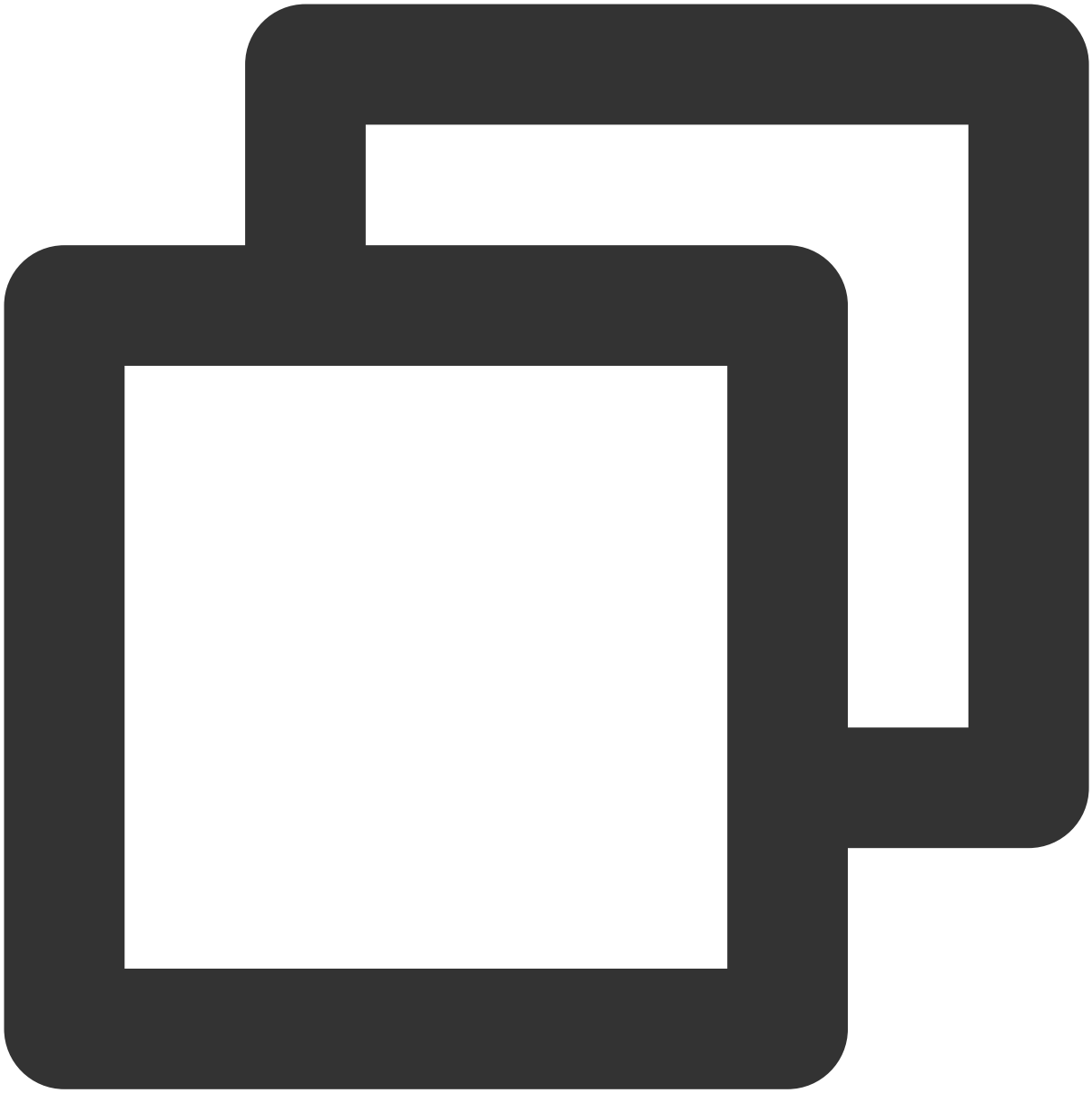
高级接口（推荐）

下载对象（断点续传）

功能说明

分块下载接口根据用户对象的长度，自动使用 Range 下载数据，实现并发下载，分块大小默认1048576（1MB），可通过 part_size 参数调整。

方法原型



```
cos_status_t *cos_resumable_download_file(cos_request_options_t *options,
                                          cos_string_t *bucket,
                                          cos_string_t *object,
                                          cos_string_t *filepath,
                                          cos_table_t *headers,
                                          cos_table_t *params,
                                          cos_resumable_clt_params_t *clt_params,
                                          cos_progress_callback progress_callback);
```

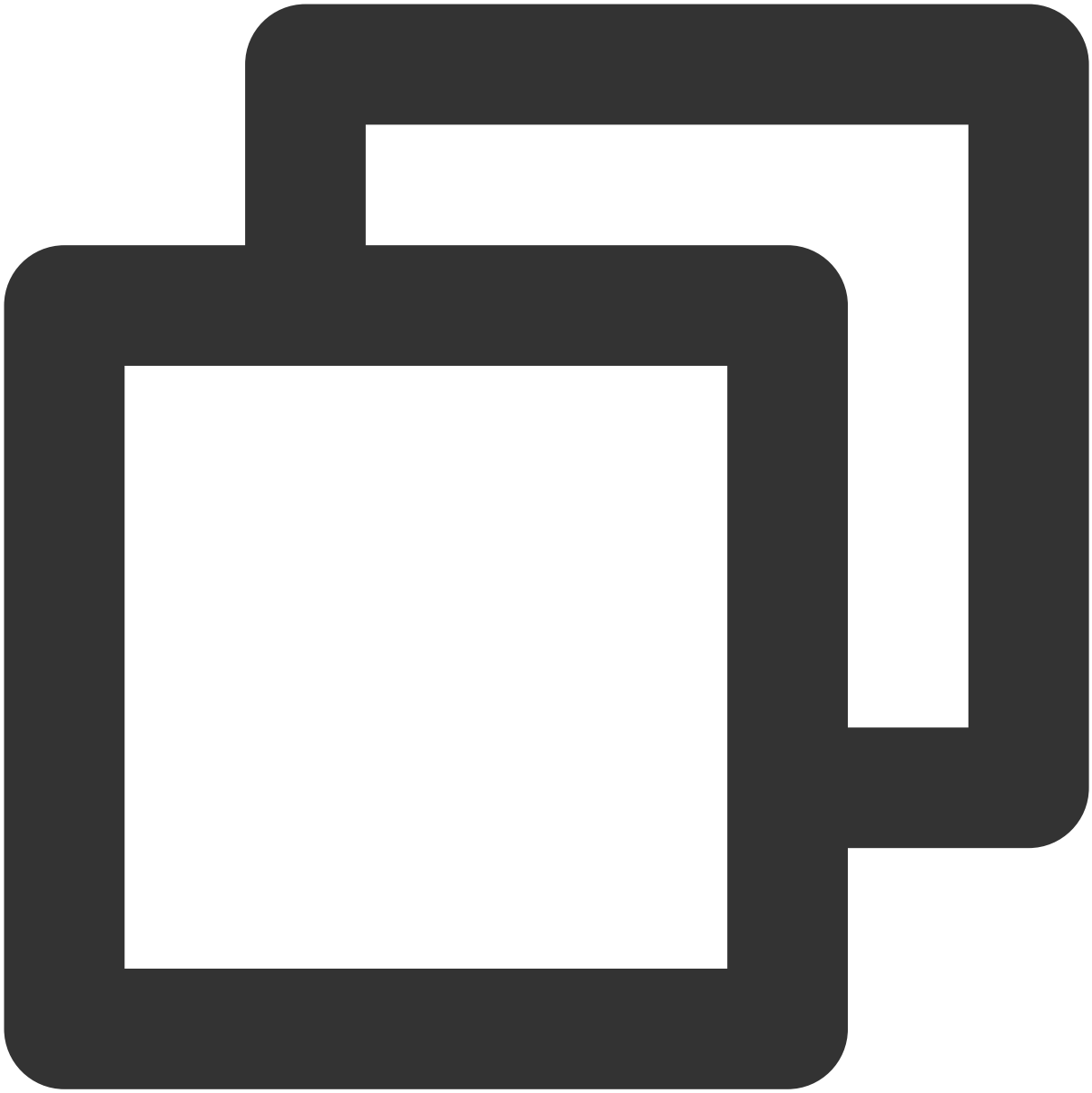
参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
filepath	Object 本地文件名称	String
headers	COS 请求附加头域	Struct
params	COS 请求操作参数	Struct
clt_params	下载对象控制参数	Struct
part_size	块大小，单位为 bytes，如果用户指定的 part_size 小于4194304（4MB），则按4194304（4MB）处理	Int
thread_num	线程池大小，默认为1	Int
enable_checkpoint	是否使能断点续传	Int
checkpoint_path	当使能断点续传时，表示保存上传进度的文件路径，默认路径为 `<filepath>.cp` ，其中 filepath 为 Object 本地文件名称	String
progress_callback	下载进度回调函数	Function

结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_DOWNLOAD_NAME4[] = "multipart_download.dat";
static char TEST_MULTIPART_OBJECT4[] = "multipart4.dat";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_resumable()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t filepath;
    cos_resumable_clt_params_t *clt_params;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_MULTIPART_OBJECT4);
    cos_str_set(&filepath, TEST_DOWNLOAD_NAME4);
```

```
    clt_params = cos_create_resumable_clt_params_content(p, 5*1024*1024, 3, COS_FAL
    s = cos_resumable_download_file(options, &bucket, &object, &filepath, NULL, NUL
    log_status(s);

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_resumable();

    cos_http_io_deinitialize();

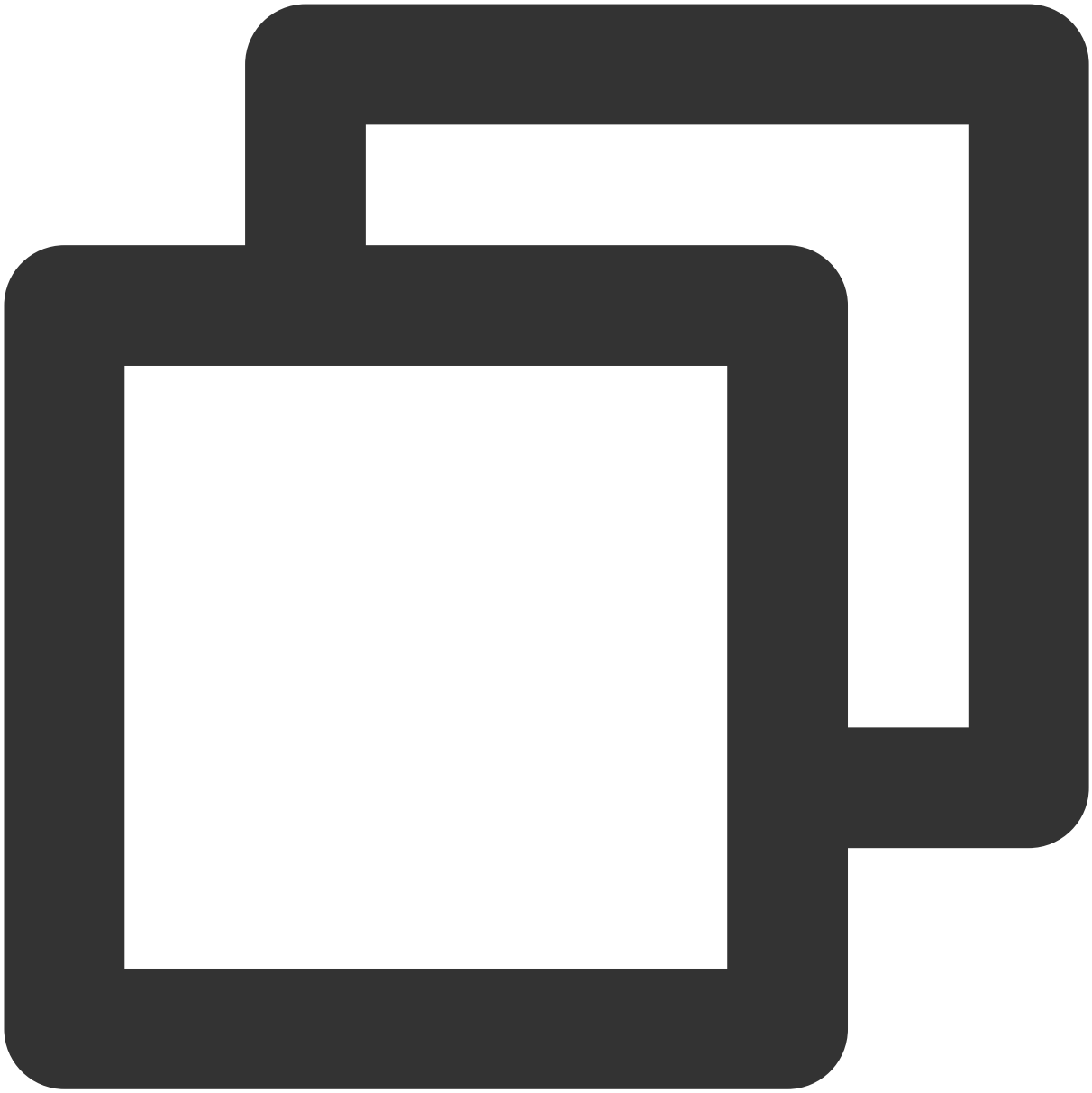
    return 0;
}
```

批量下载（从 COS 下载目录）

功能说明

将 COS 目录及其文件下载到本地磁盘。

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_download_directory()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t file_name;
    cos_string_t suffix = cos_string("/");
    cos_table_t *resp_headers;
    cos_table_t *headers = NULL;
    cos_table_t *params = NULL;
    int is_truncated = 1;
    cos_string_t marker;
    apr_status_t status;

    //初始化请求选项
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
```

```
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //list object (get bucket)
    cos_list_object_params_t *list_params = NULL;
    list_params = cos_create_list_object_params(p);
    cos_str_set(&list_params->prefix, "folder/");    //替换为您自己的目录名称
    cos_str_set(&marker, "");
    while (is_truncated) {
        list_params->marker = marker;
        s = cos_list_object(options, &bucket, list_params, &resp_headers);
        log_status(s);
        if (!cos_status_is_ok(s)) {
            printf("list object failed, req_id:%s\\n", s->req_id);
            break;
        }
        cos_list_object_content_t *content = NULL;
        cos_list_for_each_entry(cos_list_object_content_t, content, &list_params->o
            cos_str_set(&file_name, content->key.data);
            if (cos_ends_with(&content->key, &suffix)) {
                //如果是目录需要先创建，0x0755权限可以自己按需修改，参考apr_file_info.h中定
                status = apr_dir_make(content->key.data, 0x0755, options->pool);
                if (status != APR_SUCCESS && !APR_STATUS_IS_EEXIST(status)) {
                    printf("mkdir: %s failed, status: %d\\n", content->key.data, st
                }
            } else {
                //下载对象到本地目录，这里默认下载在程序运行的当前目录
                s = cos_get_object_to_file(options, &bucket, &content->key, headers
                if (!cos_status_is_ok(s)) {
                    printf("get object[%s] failed, req_id:%s\\n", content->key.data
                }
            }
        }
        is_truncated = list_params->truncated;
        marker = list_params->next_marker;
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
```

```
        exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_download_directory();

    cos_http_io_deinitialize();

    return 0;
}
```

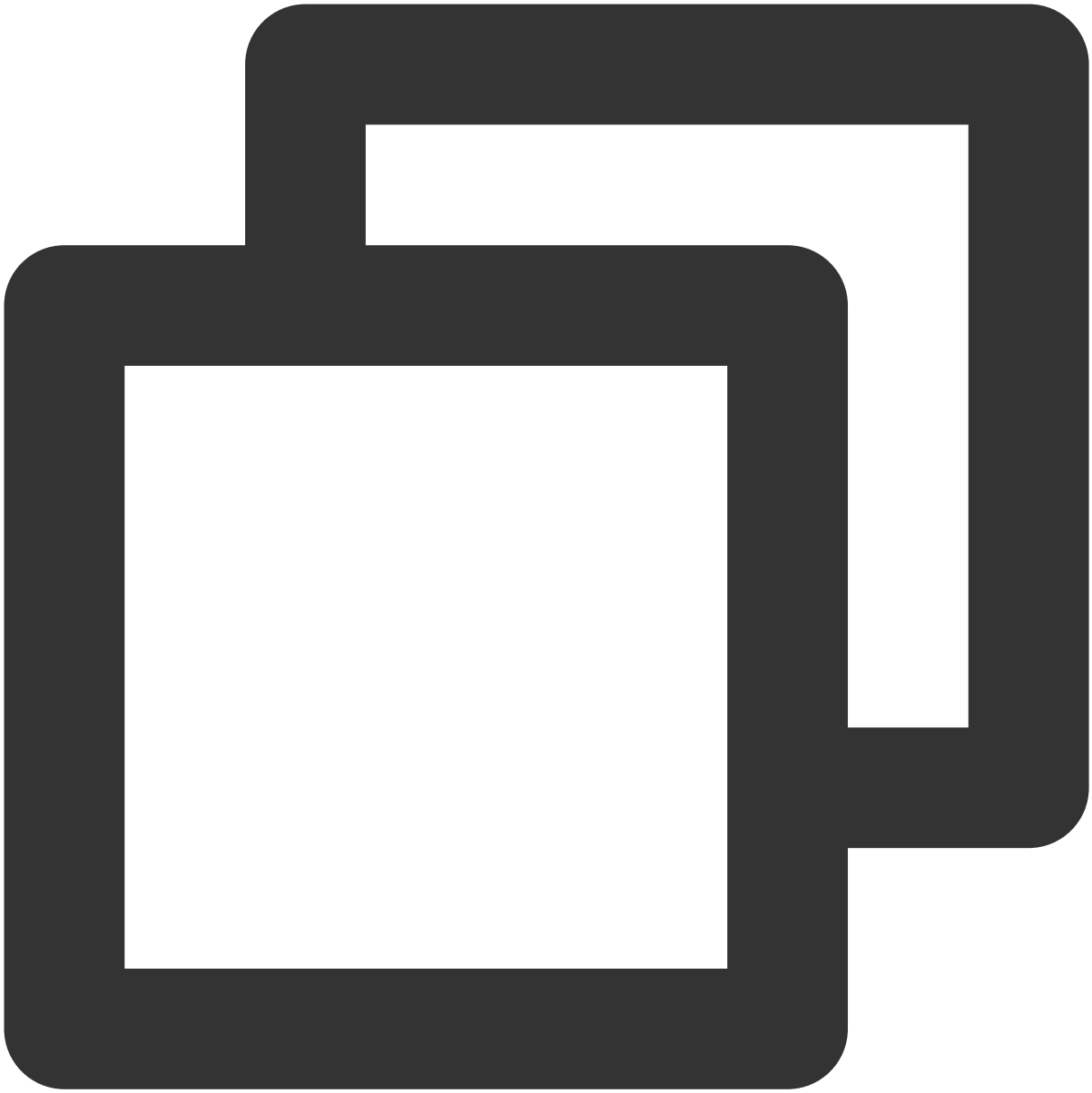
简单操作

下载对象

功能说明

下载一个对象至本地。该操作需要对目标对象具有读权限或该目标对象已对所有人都开放了读权限（公有读）。

方法原型



```
cos_status_t *cos_get_object_to_file(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     const cos_string_t *object,
                                     cos_table_t *headers,
                                     cos_table_t *params,
                                     cos_string_t *filename,
                                     cos_table_t **resp_headers);
```

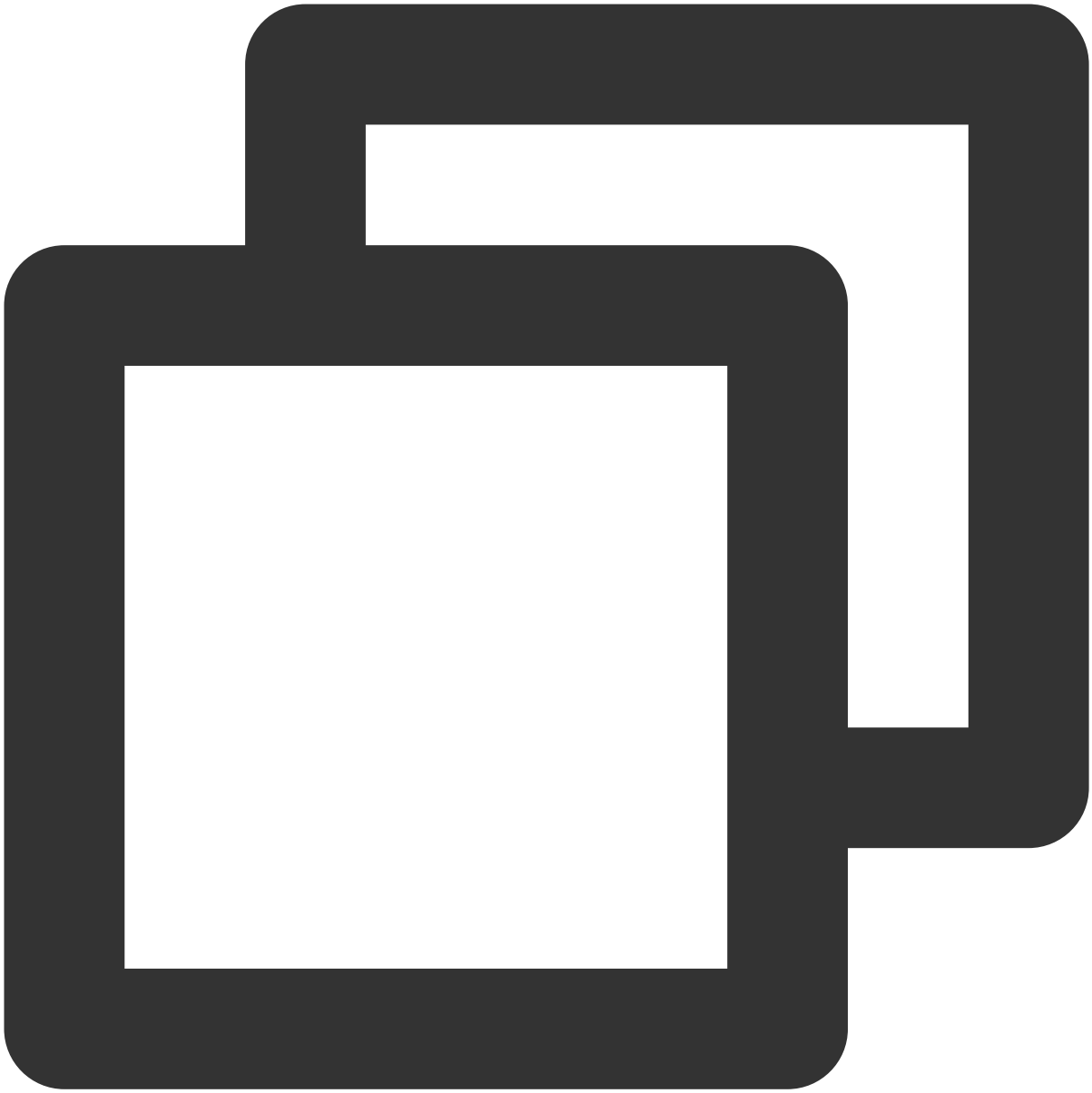
参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
headers	COS 请求附加头域	Struct
params	COS 请求操作参数	Struct
filename	Object 本地保存文件名称	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例：普通下载



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";
static char TEST_DOWNLOAD_NAME[] = "download_test3.dat";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_download_object()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t file;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
```

```
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //获取对象
    cos_str_set(&file, TEST_DOWNLOAD_NAME);
    cos_str_set(&object, TEST_OBJECT_NAME1);
    s = cos_get_object_to_file(options, &bucket, &object, NULL, NULL, &file, &resp_
    if (cos_status_is_ok(s)) {
        printf("get object succeeded\\n");
    } else {
        printf("get object failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_download_object();

    cos_http_io_deinitialize();

    return 0;
}
```

版权所有：腾讯云计算（北京）有限责任公司

列出对象

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于对象列举相关的 API 概览以及 SDK 示例代码。

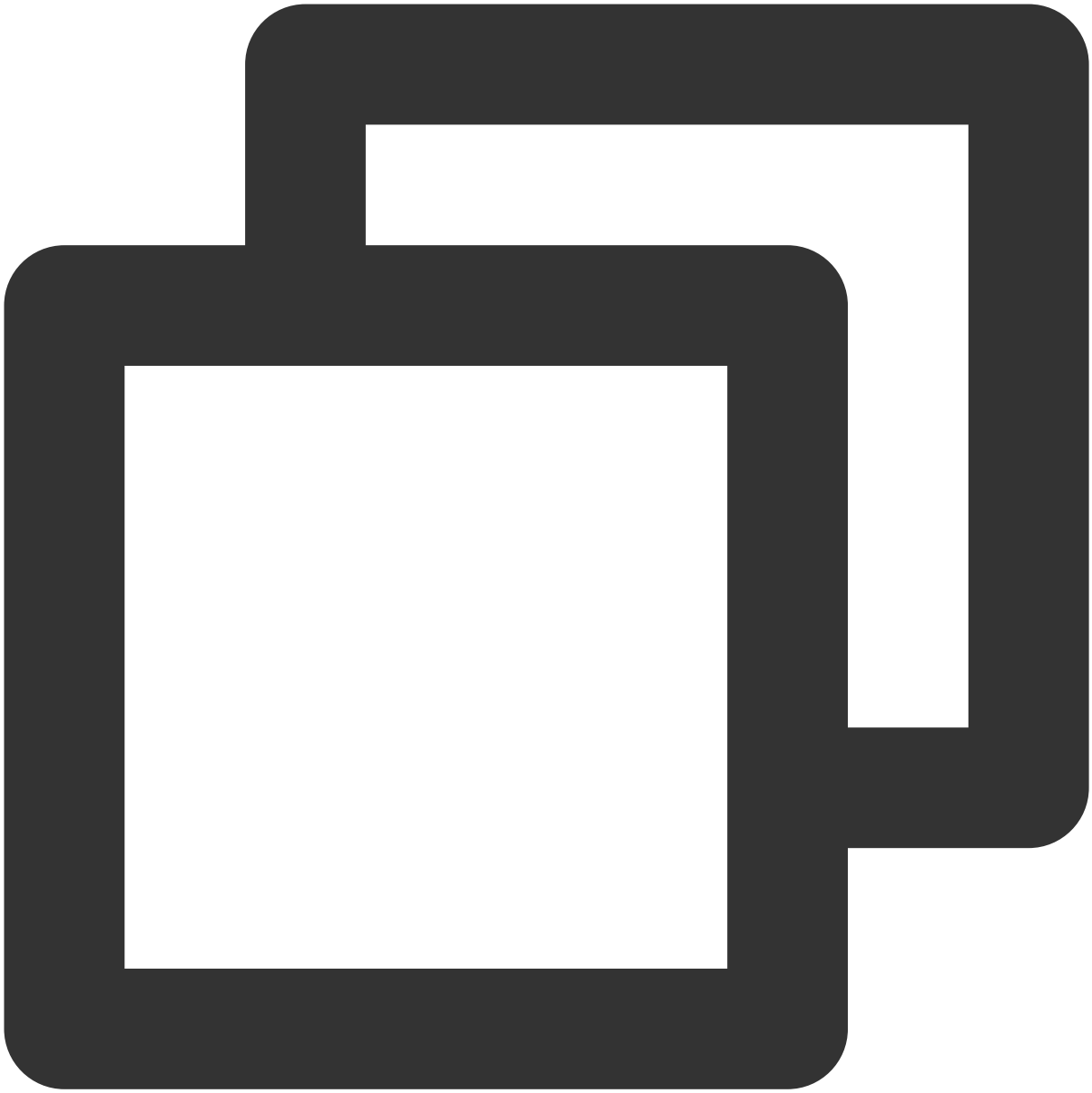
API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

方法原型



```
cos_status_t *cos_list_object(const cos_request_options_t *options,
                              const cos_string_t *bucket,
                              cos_list_object_params_t *params,
                              cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct

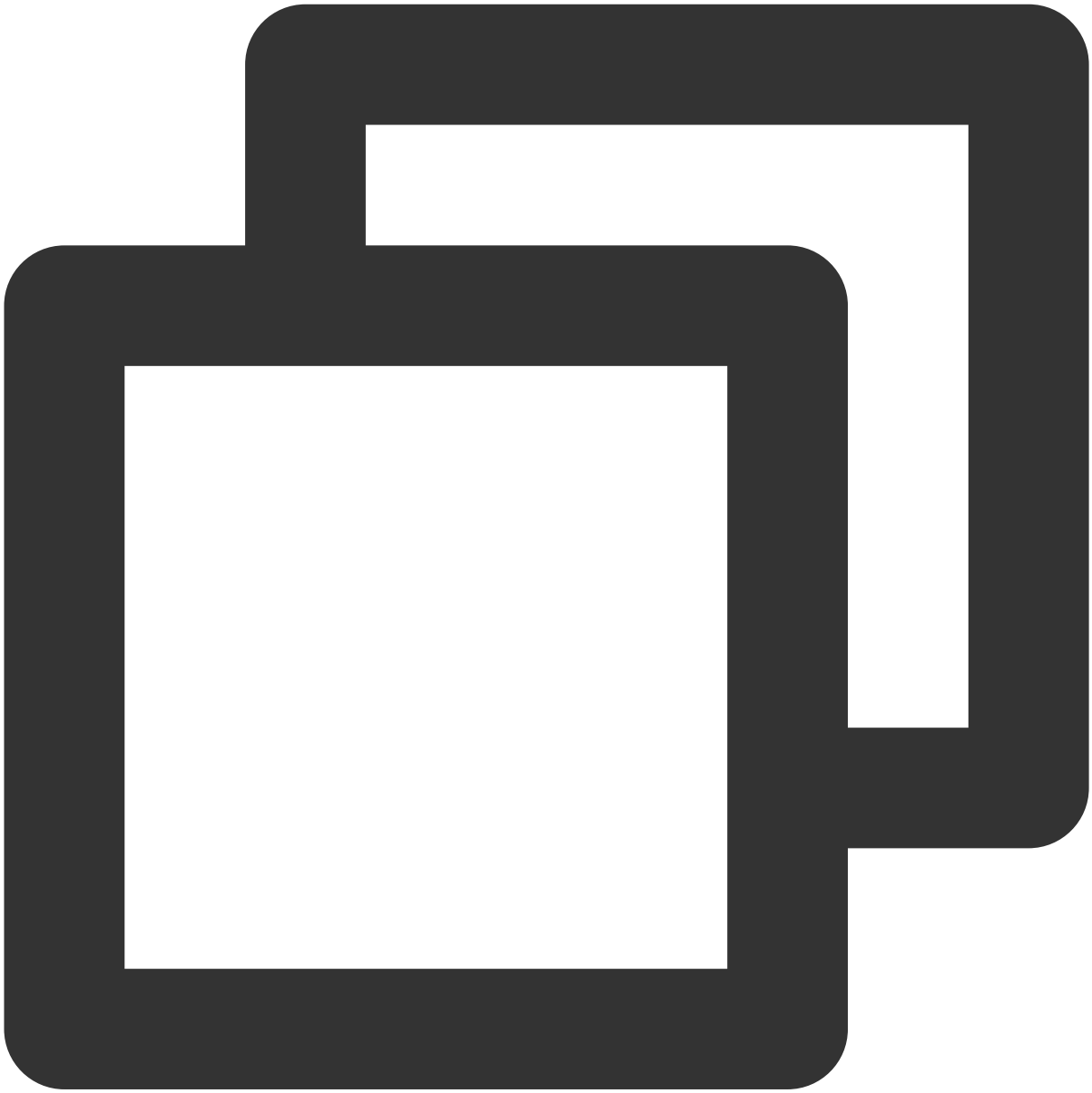
bucket	存储桶名称，Bucket 的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
params	列表操作参数信息	Struct
encoding_type	规定返回值的编码方式	String
prefix	前缀匹配，用来规定返回的文件前缀地址	String
marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
delimiter	查询分隔符，用于对对象键进行分组	String
max_ret	单次返回最大的条目数量，默认1000	Struct
truncated	返回条目是否被截断，'true' 或者 'false'	Boolean
next_marker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
object_list	Get Bucket 操作返回的对象信息列表	Struct
key	Get Bucket 操作返回的 Object 名称	Struct
last_modified	Get Bucket 操作返回的 Object 最后修改时间	Struct
etag	Get Bucket 操作返回的对象的 SHA-1 算法校验值	Struct
size	Get Bucket 操作返回的对象大小，单位 Byte	Struct
owner_id	Get Bucket 操作返回的对象拥有者 UID 信息	Struct
storage_class	Get Bucket 操作返回的对象存储级别	Struct
common_prefix_list	将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

版权所有：腾讯云计算（北京）有限责任公司

示例1：列出第一页对象



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
```

```
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_list_objects()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //获取对象列表
    cos_list_object_params_t *list_params = NULL;
    cos_list_object_content_t *content = NULL;
    list_params = cos_create_list_object_params(p);
    s = cos_list_object(options, &bucket, list_params, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("list object succeeded\\n");
        cos_list_for_each_entry(cos_list_object_content_t, content, &list_params->o
           printf("object: %.*s\\n", content->key.len, content->key.data);
```

```
        }
    } else {
        printf("list object failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

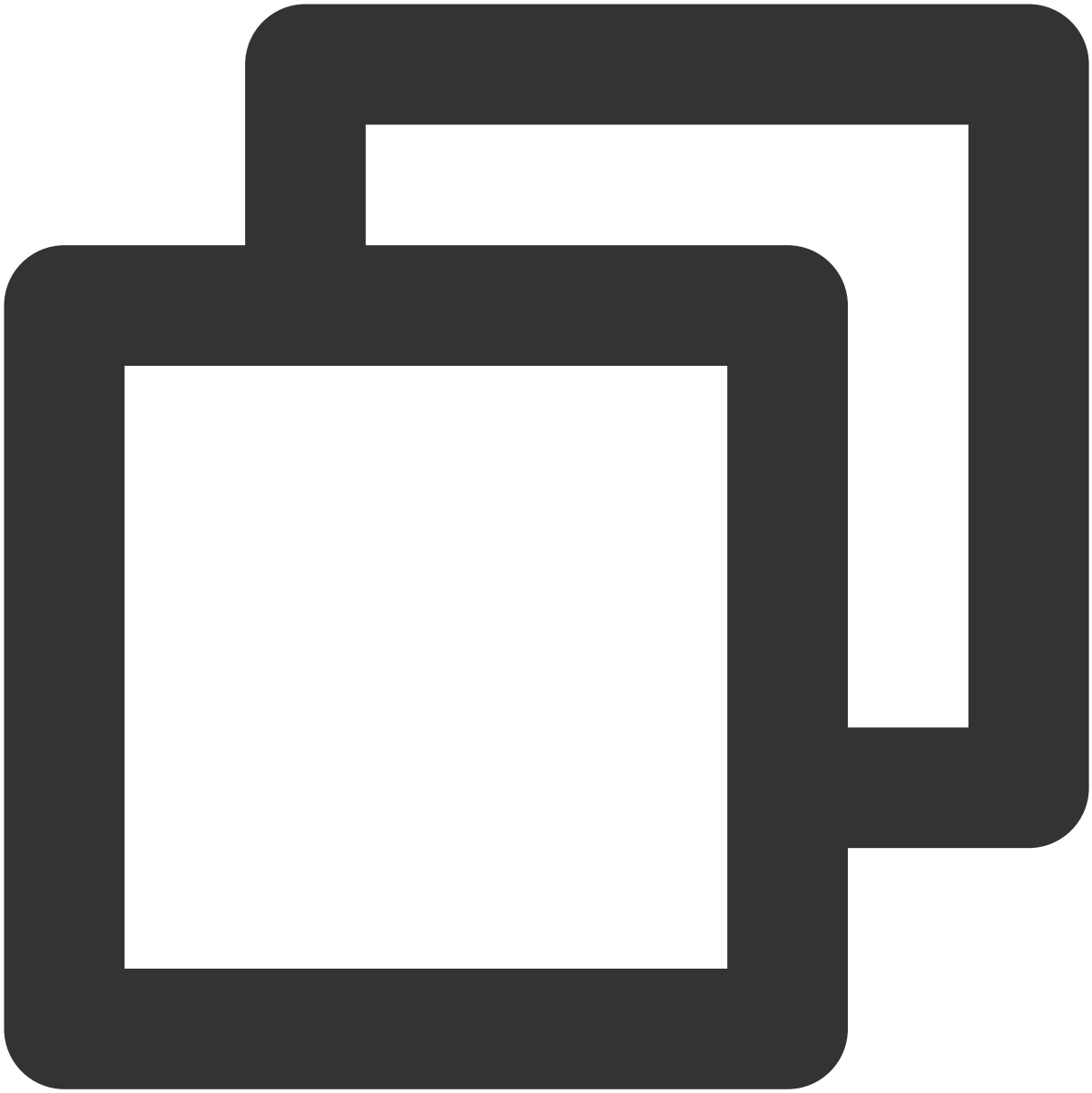
    test_list_objects();

    cos_http_io_deinitialize();

    return 0;
}
```

示例2：列出目录下的对象

对象存储中本身是没有文件夹和目录的概念的，为了满足用户使用习惯，用户可通过分隔符/来模拟“文件夹”。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_list_directory()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers;
    int is_truncated = 1;
    cos_string_t marker;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //list object (get bucket)
    cos_list_object_params_t *list_params = NULL;
    list_params = cos_create_list_object_params(p);
    // prefix表示列出的object的key以prefix开始
    cos_str_set(&list_params->prefix, "folder/");
    // deliter表示分隔符, 设置为/表示列出当前目录下的object, 设置为空表示列出所有的object
    cos_str_set(&list_params->delimiter, "/");
    // 设置最大遍历出多少个对象, 一次listobject最大支持1000
    list_params->max_ret = 1000;
    cos_str_set(&marker, "");
    while (is_truncated) {
        list_params->marker = marker;
```

```
        s = cos_list_object(options, &bucket, list_params, &resp_headers);
        if (!cos_status_is_ok(s)) {
            printf("list object failed, req_id:%s\\n", s->req_id);
            break;
        }
        // list_params->object_list 返回列出的object对象。
        cos_list_object_content_t *content = NULL;
        cos_list_for_each_entry(cos_list_object_content_t, content, &list_params->o
            printf("object: %s\\n", content->key.data);
        }
        // list_params->common_prefix_list 表示被delimiter截断的路径, 如delimter设置为/
        cos_list_object_common_prefix_t *common_prefix = NULL;
        cos_list_for_each_entry(cos_list_object_common_prefix_t, common_prefix, &li
            printf("common prefix: %s\\n", common_prefix->prefix.data);
        }

        is_truncated = list_params->truncated;
        marker = list_params->next_marker;
    }
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

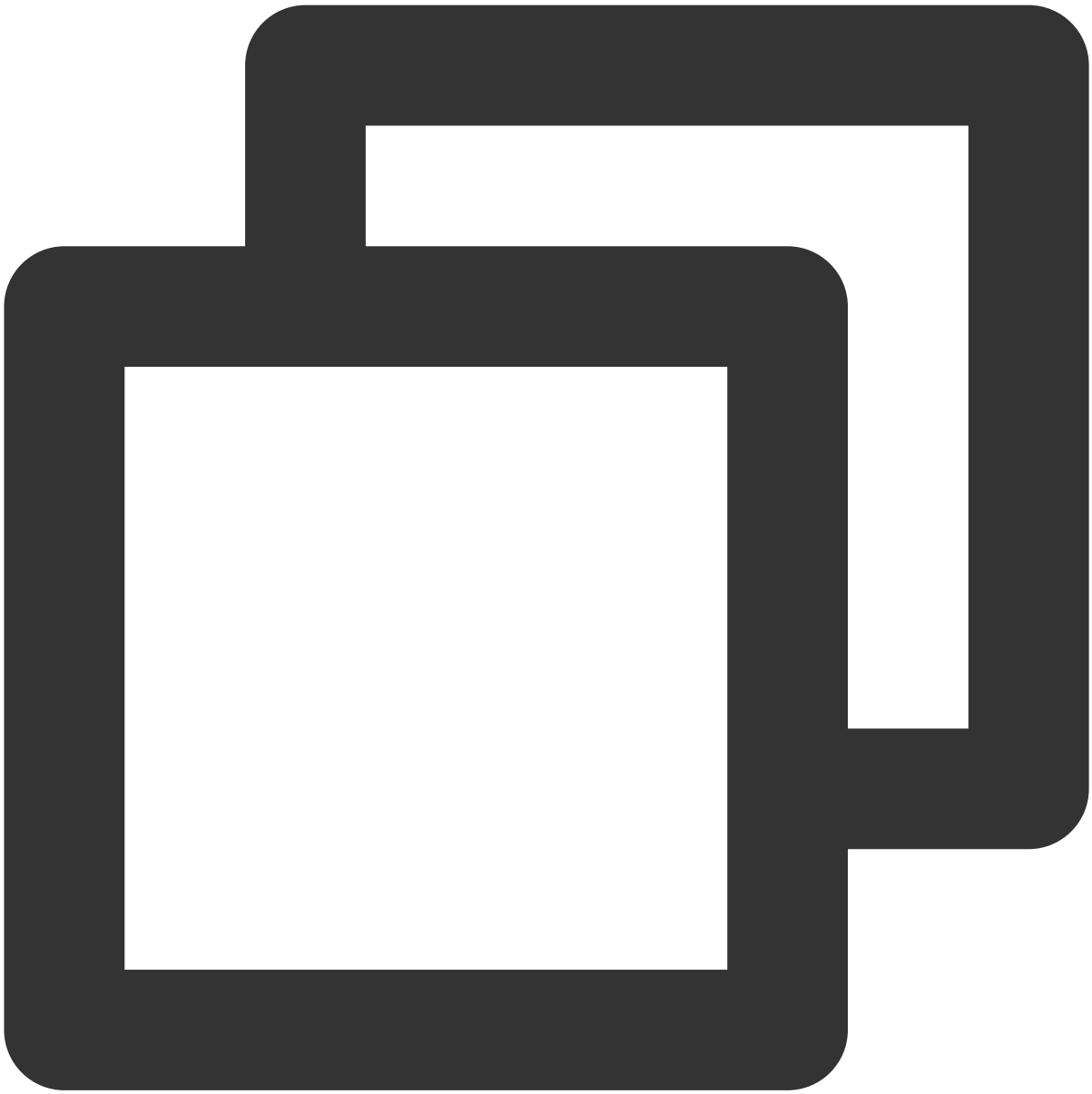
    test_list_directory();

    cos_http_io_deinitialize();

    return 0;
}
```

示例3：列出桶中的所有对象

一次列举的最大对象数为1000，当桶内对象个数超过1000后，需要去循环列举桶中所有的对象。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
```

```
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_list_all_objects()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers;
    int is_truncated = 1;
    cos_string_t marker;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //list object (get bucket)
    cos_list_object_params_t *list_params = NULL;
    list_params = cos_create_list_object_params(p);
    // 设置最大遍历出多少个对象，一次listobject最大支持1000
    list_params->max_ret = 1000;
    cos_str_set(&marker, "");
    while (is_truncated) {
        list_params->marker = marker;
        cos_list_init(&list_params->object_list);
        s = cos_list_object(options, &bucket, list_params, &resp_headers);
```

```
        if (!cos_status_is_ok(s)) {
            printf("list object failed, req_id:%s\\n", s->req_id);
            break;
        }
        // list_params->object_list 返回列出的object对象。
        cos_list_object_content_t *content = NULL;
        cos_list_for_each_entry(cos_list_object_content_t, content, &list_params->o
            printf("object: %s\\n", content->key.data);
        }

        is_truncated = list_params->truncated;
        marker = list_params->next_marker;
    }
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_list_all_objects();

    cos_http_io_deinitialize();

    return 0;
}
```

删除对象

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于对象删除操作相关的 API 概览以及 SDK 示例代码。

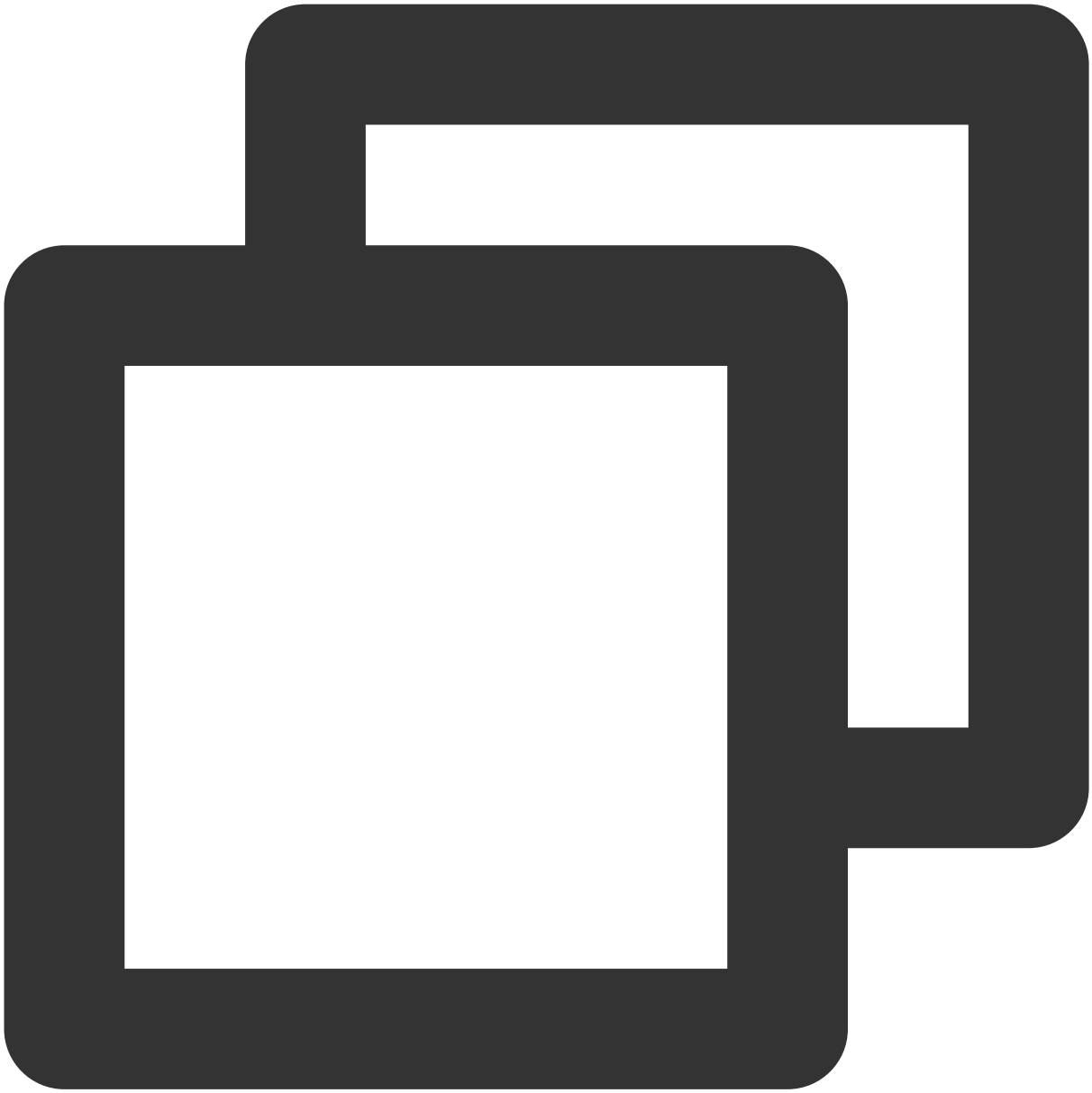
API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

删除单个对象

功能说明

在存储桶中删除指定对象。

方法原型



```
cos_status_t *cos_delete_object(const cos_request_options_t *options,
                                const cos_string_t *bucket,
                                const cos_string_t *object,
                                cos_table_t **resp_headers);
```

参数说明

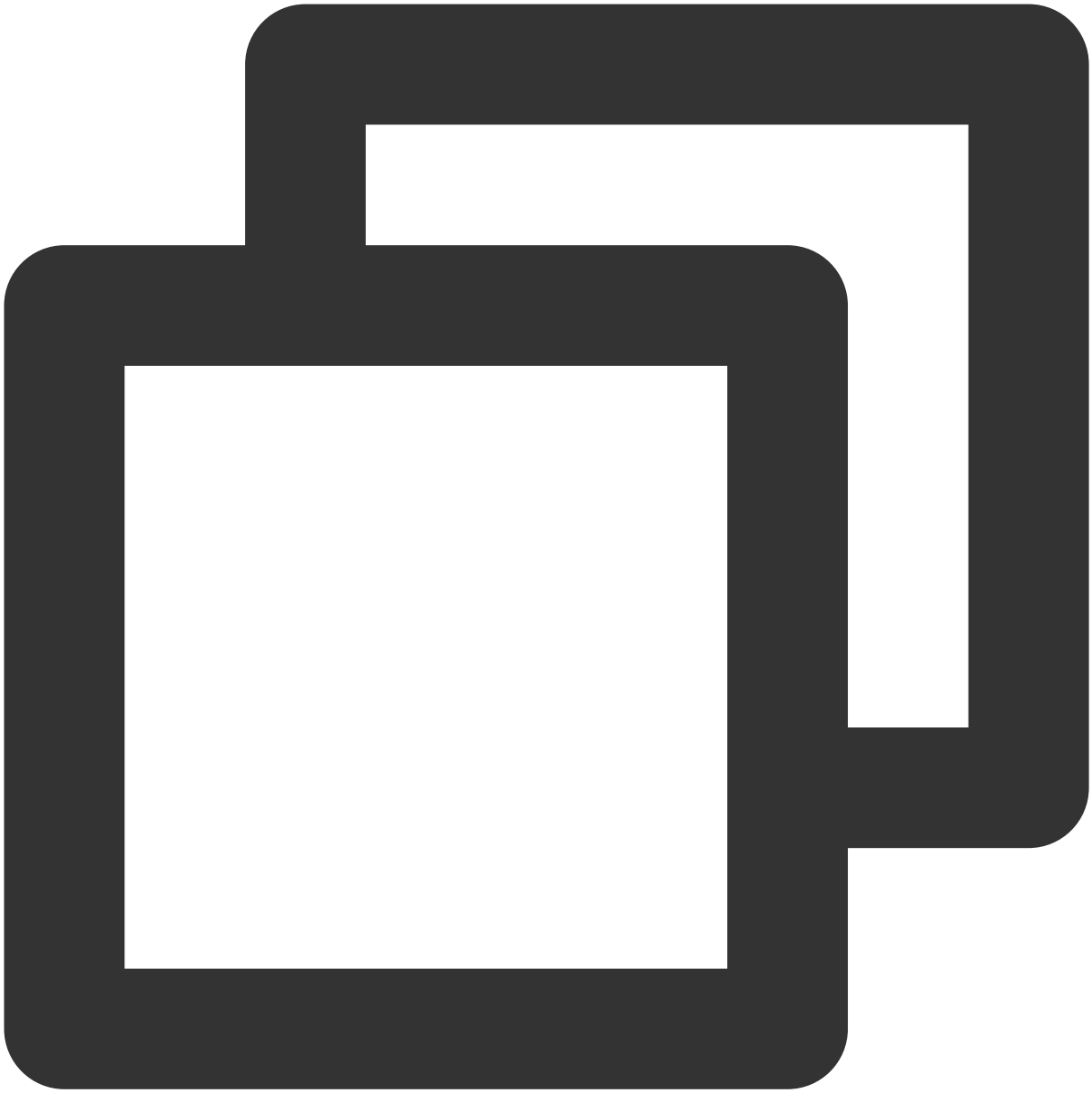
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例1：删除对象



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_object()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //删除单个对象
    cos_str_set(&object, TEST_OBJECT_NAME1);
    s = cos_delete_object(options, &bucket, &object, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("delete object succeeded\\n");
    } else {
        printf("delete object failed\\n");
    }
```

```
    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

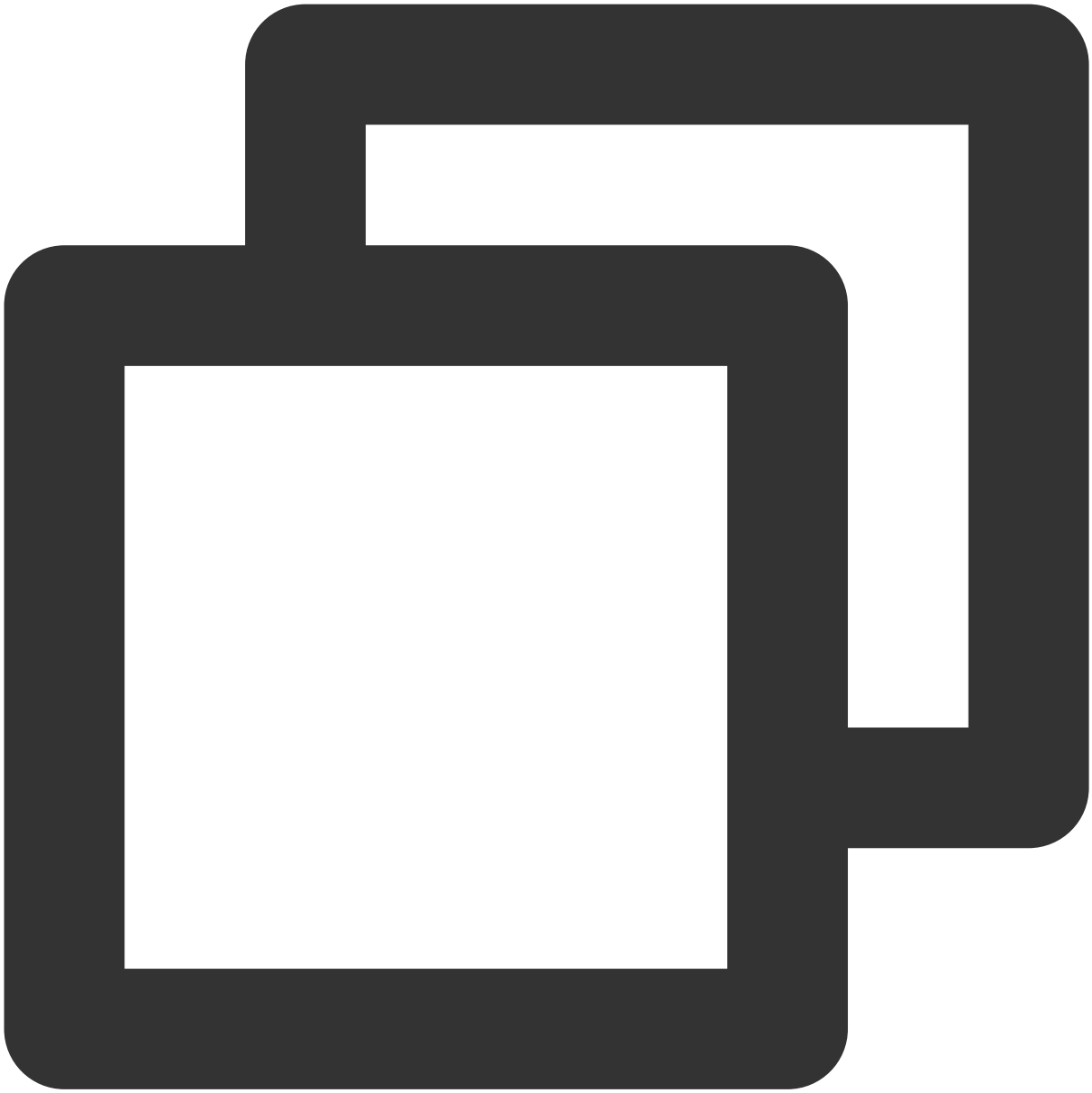
    test_delete_object();

    cos_http_io_deinitialize();

    return 0;
}
```

示例2：删除文件夹

该请求不会删除文件夹内的对象，只会删除指定的 key。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_dir()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //删除目录对象
    cos_str_set(&object, "folder/");
    s = cos_delete_object(options, &bucket, &object, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("delete object succeeded\\n");
    } else {
        printf("delete object failed\\n");
    }

    //销毁内存池
```

```
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_delete_dir();

    cos_http_io_deinitialize();

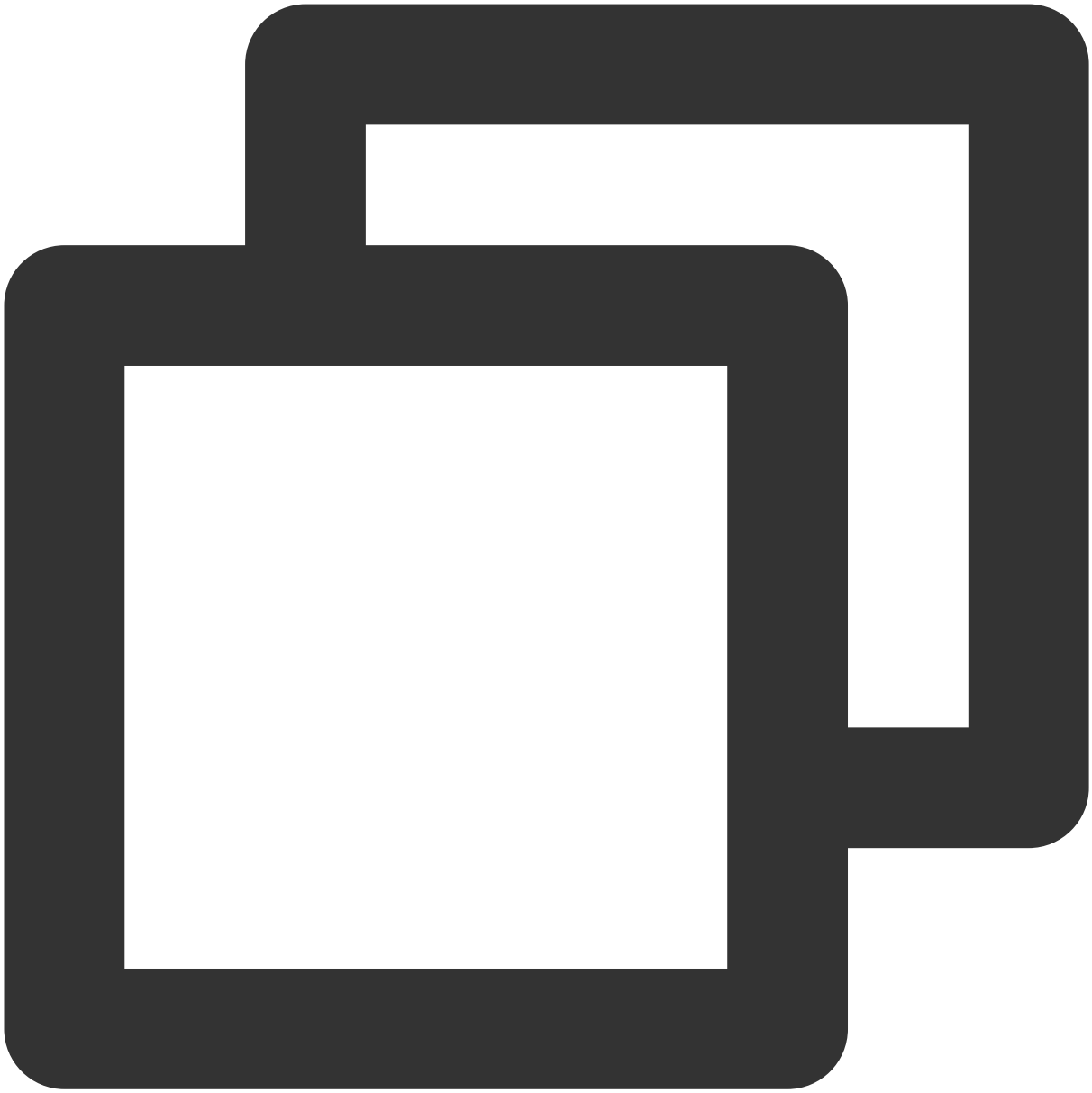
    return 0;
}
```

删除多个对象

功能说明

在存储桶中批量删除对象，最大支持单次删除1000个对象。对于返回结果，COS 提供 Verbose 和 Quiet 两种结果模式。Verbose 模式将返回每个 Object 的删除结果。Quiet 模式只返回报错的 Object 信息。

方法原型



```
cos_status_t *cos_delete_objects(const cos_request_options_t *options,
                                 const cos_string_t *bucket,
                                 cos_list_t *object_list,
                                 int is_quiet,
                                 cos_table_t **resp_headers,
                                 cos_list_t *deleted_object_list);
```

参数说明

参数名称	参数描述	类型

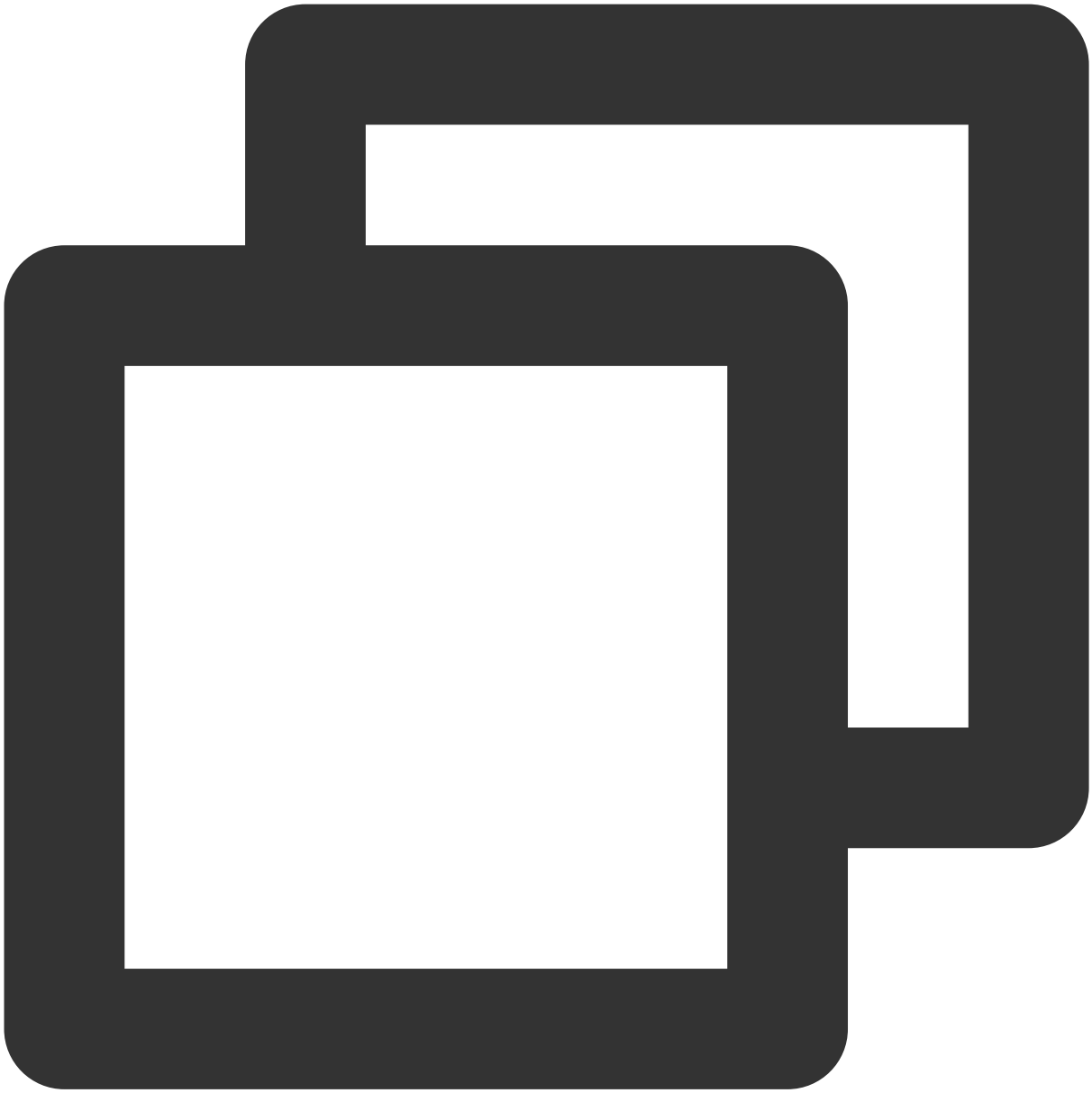
版权所有：腾讯云计算（北京）有限责任公司

options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object_list	Object 待删除列表	Struct
key	待删除 Object 名称	String
is_quiet	决定是否启动 Quiet 模式： True(1)：启动 Quiet 模式，False(0)：启动 Verbose 模式。默认为 False(0)	Boolean
resp_headers	返回 HTTP 响应消息的头域	Struct
deleted_object_list	Object 删除信息列表	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例1：删除多个对象



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME2[] = "test2.dat";
static char TEST_OBJECT_NAME3[] = "test3.dat";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_objects()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_string_t bucket;
    cos_status_t *s = NULL;
    cos_table_t *resp_headers = NULL;
    cos_request_options_t *options = NULL;
    char *object_name1 = TEST_OBJECT_NAME2;
    char *object_name2 = TEST_OBJECT_NAME3;
    cos_object_key_t *content1 = NULL;
    cos_object_key_t *content2 = NULL;
    cos_list_t object_list;
    cos_list_t deleted_object_list;
    int is_quiet = COS_TRUE;

    cos_pool_create(&p, NULL);
```

```
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    cos_list_init(&object_list);
    cos_list_init(&deleted_object_list);
    content1 = cos_create_cos_object_key(p);
    cos_str_set(&content1->key, object_name1);
    cos_list_add_tail(&content1->node, &object_list);
    content2 = cos_create_cos_object_key(p);
    cos_str_set(&content2->key, object_name2);
    cos_list_add_tail(&content2->node, &object_list);

    s = cos_delete_objects(options, &bucket, &object_list, is_quiet,
        &resp_headers, &deleted_object_list);
    log_status(s);

    cos_pool_destroy(p);

    if (cos_status_is_ok(s)) {
        printf("delete objects succeeded\\n");
    } else {
        printf("delete objects failed\\n");
    }
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_delete_objects();

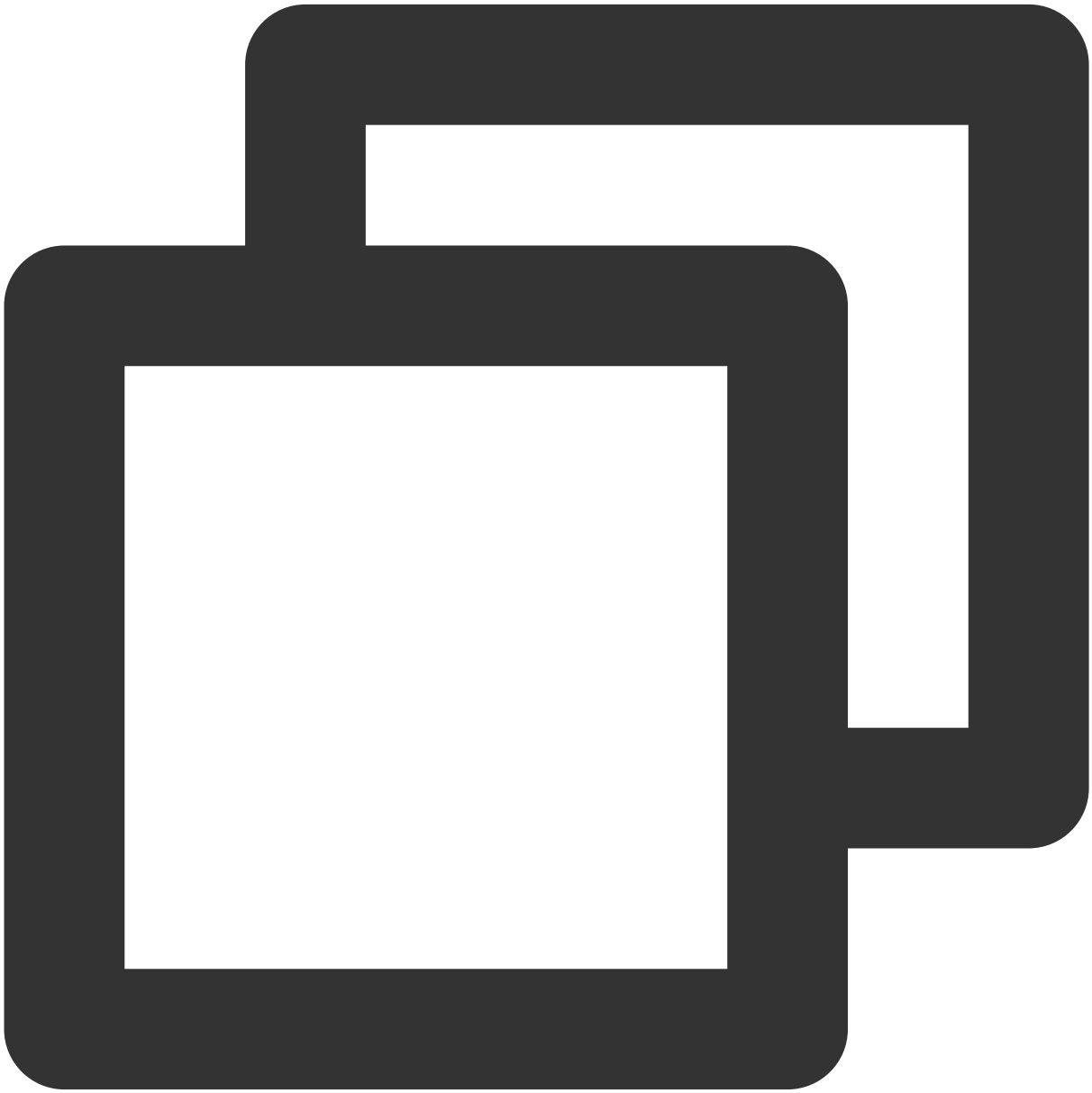
    cos_http_io_deinitialize();

    return 0;
```

```
}
```

示例2：删除文件夹及其文件

对象存储中本身是没有文件夹和目录的概念的，为了满足用户使用习惯，用户可通过分隔符/来模拟“文件夹”。
删除文件夹及其文件这一场景，实际在 COS 上相当于删除一批有着同样前缀的对象。目前 COS C SDK 没有提供一个接口去实现这样的操作，但是可以通过组合**查询对象列表**加上**批量删除对象**的基本操作，达到删除文件夹及其文件的效果。



```
#include "cos_http_io.h"
#include "cos_api.h"
```

```
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_directory()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers;
    int is_truncated = 1;
    cos_string_t marker;
    cos_list_t deleted_object_list;
```

```
    int is_quiet = COS_TRUE;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //list object (get bucket)
    cos_list_object_params_t *list_params = NULL;
    list_params = cos_create_list_object_params(p);
    cos_str_set(&list_params->prefix, "folder/");
    cos_str_set(&marker, "");
    while (is_truncated) {
        list_params->marker = marker;
        s = cos_list_object(options, &bucket, list_params, &resp_headers);
        if (!cos_status_is_ok(s)) {
            printf("list object failed, req_id:%s\\n", s->req_id);
            break;
        }

        s = cos_delete_objects(options, &bucket, &list_params->object_list, is_quie
        log_status(s);
        if (!cos_status_is_ok(s)) {
            printf("delete objects failed, req_id:%s\\n", s->req_id);
        }

        is_truncated = list_params->truncated;
        marker = list_params->next_marker;
    }
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);
```

```
    test_delete_directory();

    cos_http_io_deinitialize();

    return 0;
}
```

对象访问 URL

最近更新时间：2024-01-04 17:06:09

简介

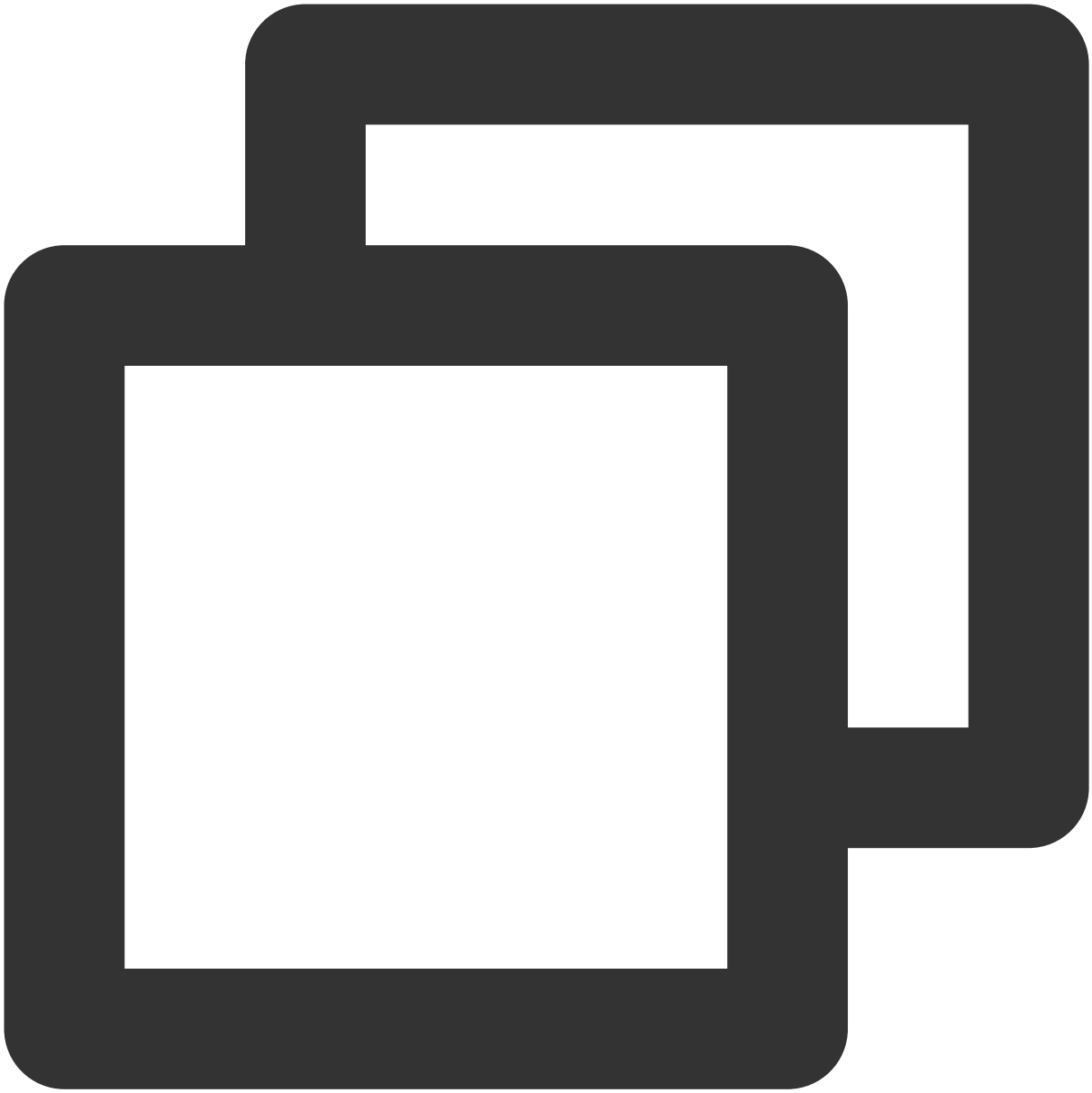
本文档提供获取对象访问 URL 的代码示例。

获取对象访问 URL

功能说明

获取对象访问 URL 用于匿名下载或分发。

方法原型



```
const char *cos_gen_object_url(const cos_request_options_t *options,
                               const cos_string_t *bucket,
                               const cos_string_t *object)
```

参数说明

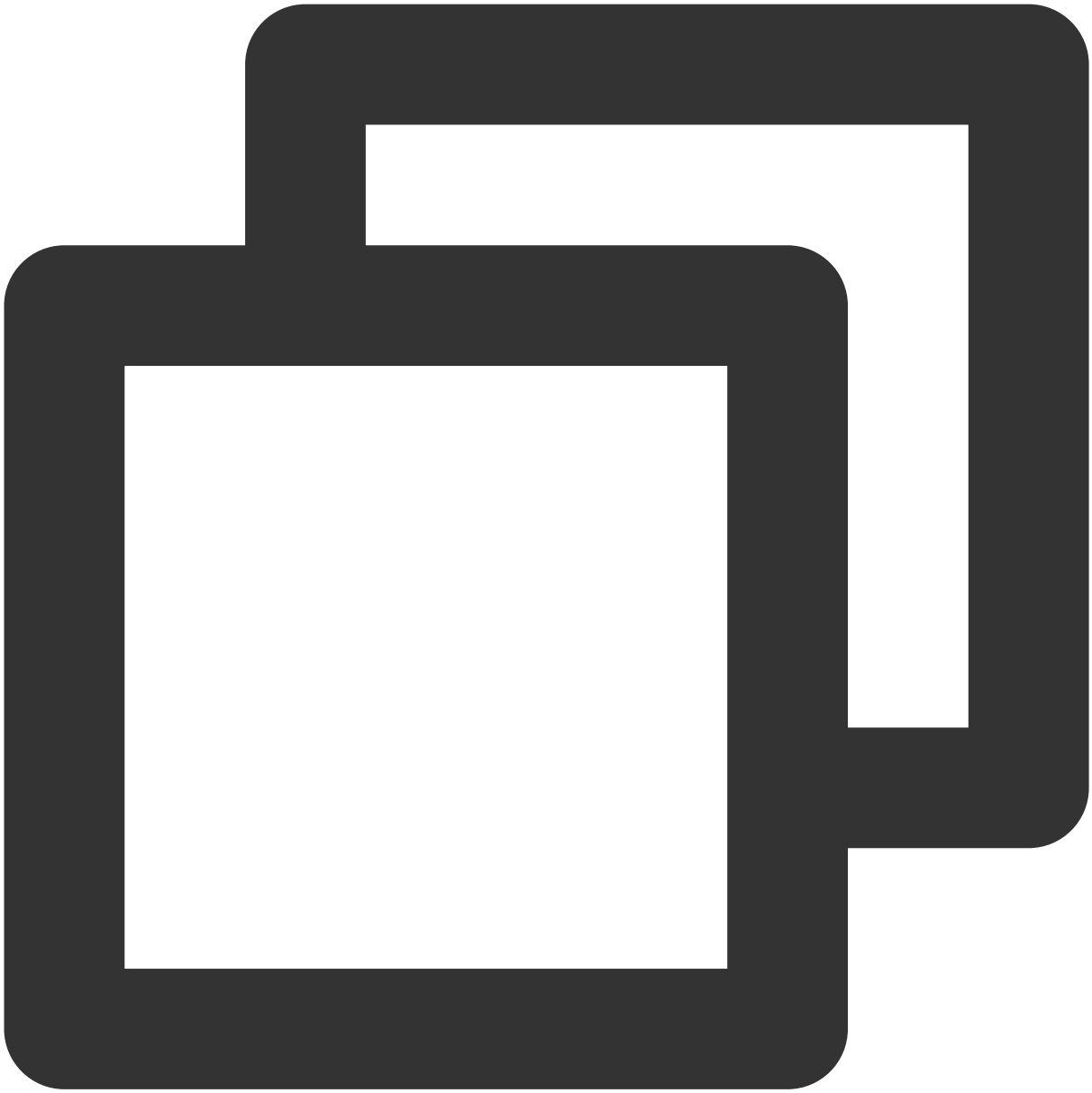
参数名称	参数描述	类型	是否必填
options	COS 请求选项	Struct	是

bucket	存储桶名称，Bucket 的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	Struct	是
object	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是

返回结果说明

该方法返回值为对象访问的 URL。

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_gen_object_url()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_OBJECT_NAME1);

    printf("url:%s\\n", cos_gen_object_url(options, &bucket, &object));

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
```

```
        exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_gen_object_url();

    cos_http_io_deinitialize();

    return 0;
}
```

恢复归档对象

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于恢复归档对象操作相关的 API 概览以及 SDK 示例代码。

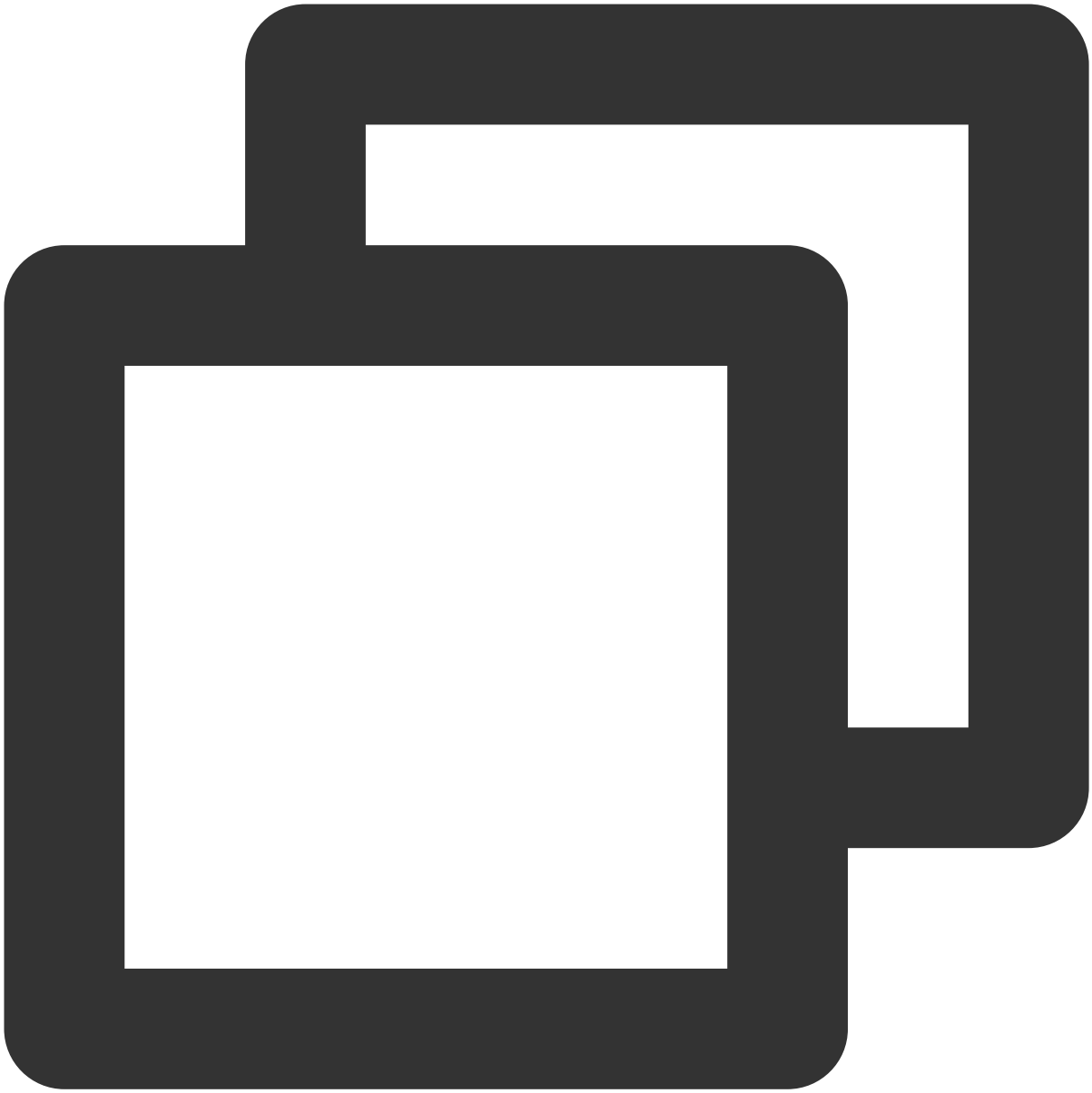
API	操作名	操作描述
POST Object restore	恢复归档对象	将归档类型的对象取回访问

恢复归档对象

功能说明

将归档类型的对象取回访问。

方法原型



```
cos_status_t *cos_post_object_restore(const cos_request_options_t *options,
                                      const cos_string_t *bucket,
                                      const cos_string_t *object,
                                      cos_object_restore_params_t *restore_params,
                                      cos_table_t *headers,
                                      cos_table_t *params,
                                      cos_table_t **resp_headers);
```

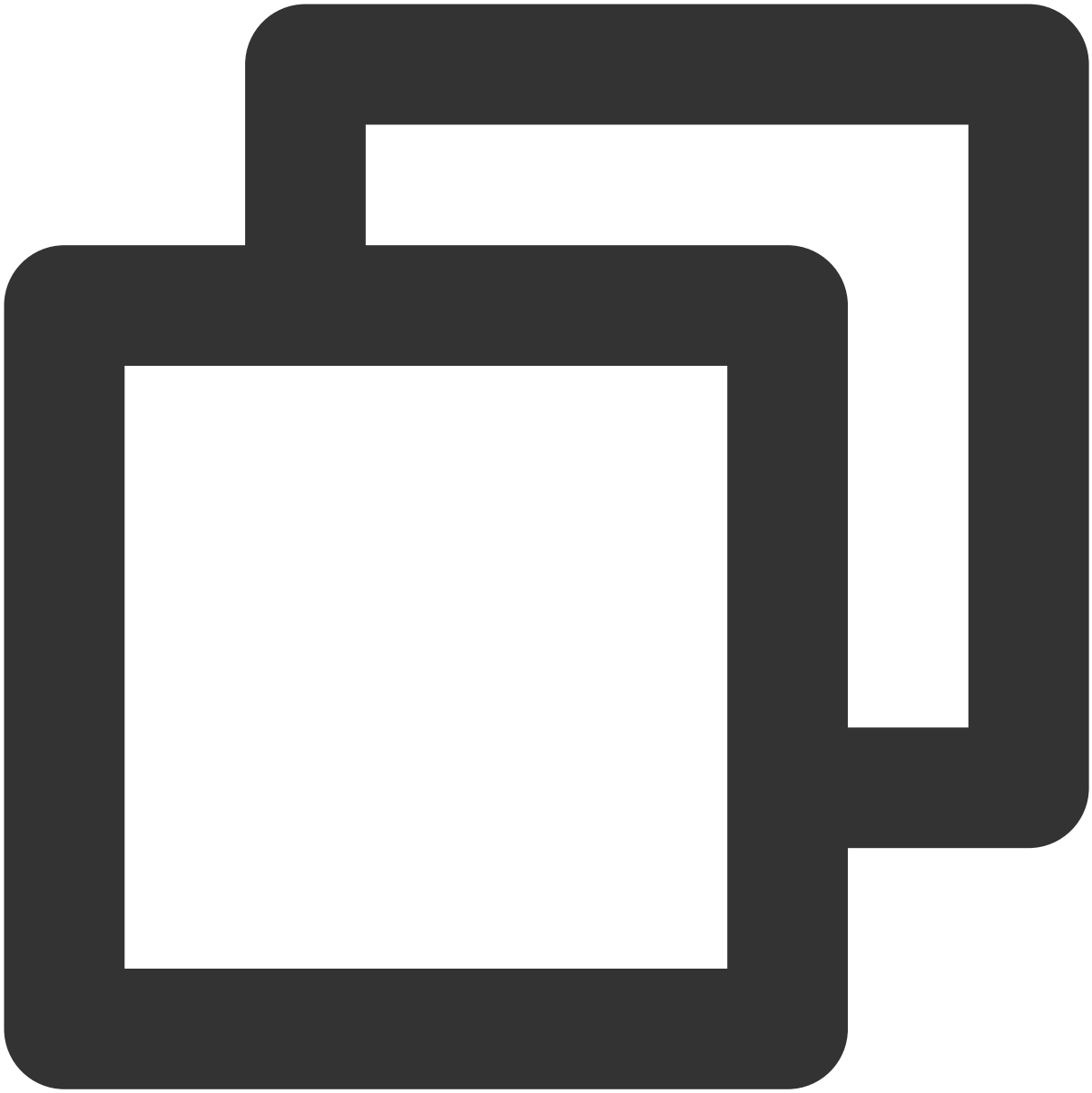
参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
restore_params	Post Object Restore 操作参数	Struct
days	Post Object Restore 操作设置的临时副本过期时间	Int
tier	Post Object Restore 操作指定 CAS 支持的三种恢复类型，分别为 Expedited、Standard、Bulk	String
headers	COS 请求附加头域	Struct
params	COS 请求操作参数	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_object_restore()
{
    cos_pool_t *p = NULL;
    cos_string_t bucket;
    cos_string_t object;
    int is_cname = 0;
    cos_table_t *resp_headers = NULL;
    cos_request_options_t *options = NULL;
    cos_status_t *s = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, "test_restore.dat");

    cos_object_restore_params_t *restore_params = cos_create_object_restore_params(
    restore_params->days = 30;
    cos_str_set(&restore_params->tier, "Standard");
    s = cos_post_object_restore(options, &bucket, &object, restore_params, NULL, NU
```

```
    log_status(s);

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_object_restore();

    cos_http_io_deinitialize();

    return 0;
}
```

查询对象元数据

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于查询对象元数据操作相关的 API 概览以及 SDK 示例代码。

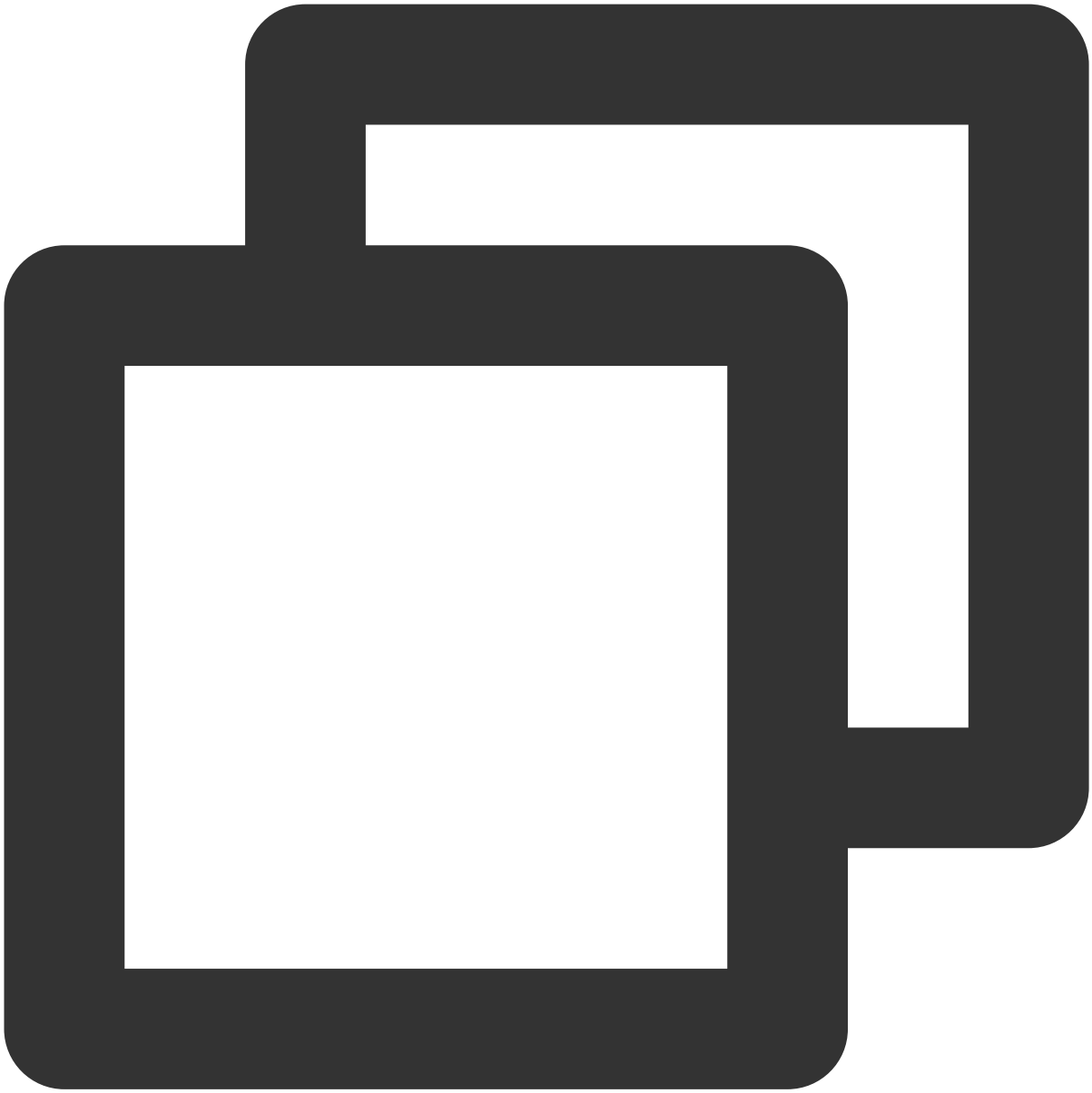
API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

查询对象元数据

功能说明

查询对象的元数据信息。

方法原型



```
cos_status_t *cos_head_object(const cos_request_options_t *options,
                              const cos_string_t *bucket,
                              const cos_string_t *object,
                              cos_table_t *headers,
                              cos_table_t **resp_headers);
```

参数说明

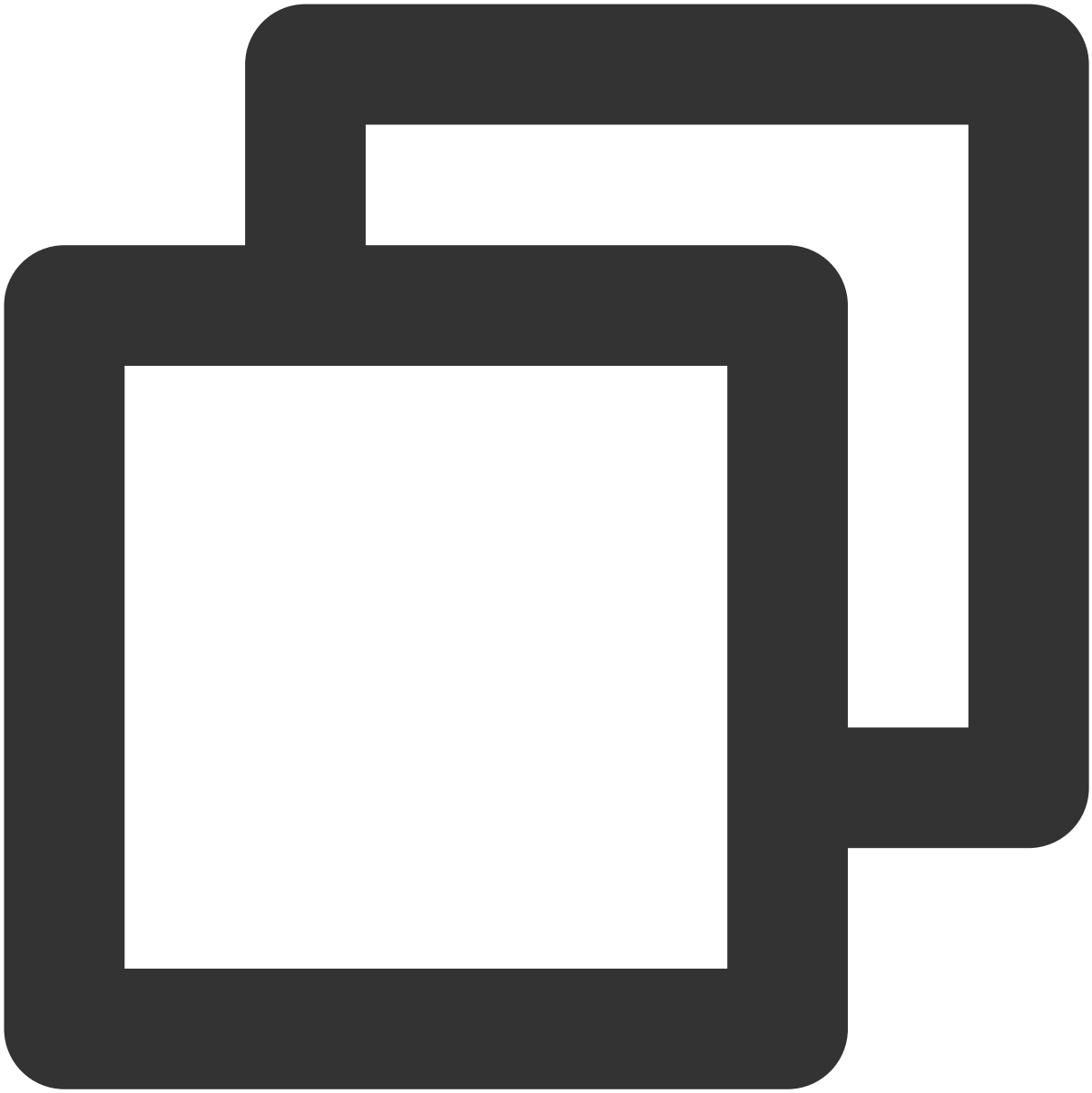
参数名称	参数描述	类型

options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

static void print_headers(cos_table_t *headers)
{
    const cos_array_header_t *tarr;
    const cos_table_entry_t *telts;
    int i = 0;

    if (apr_is_empty_table(headers)) {
        return;
    }

    tarr = cos_table_elts(headers);
    telts = (cos_table_entry_t*)tarr->elts;

    printf("headers:\\n");
    for (; i < tarr->nelts; i++) {
        telts = (cos_table_entry_t*)(tarr->elts + i * tarr->elt_size);
        printf("%s: %s\\n", telts->key, telts->val);
    }
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_head_object()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
```

```
    cos_string_t bucket;
    cos_string_t object;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //获取对象元数据
    cos_str_set(&object, TEST_OBJECT_NAME1);
    s = cos_head_object(options, &bucket, &object, NULL, &resp_headers);
    print_headers(resp_headers);
    if (cos_status_is_ok(s)) {
        printf("head object succeeded\\n");
    } else {
        printf("head object failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_head_object();

    cos_http_io_deinitialize();

    return 0;
```

```
}
```

版权所有：腾讯云计算（北京）有限责任公司

生成预签名 URL

最近更新时间：2024-01-04 17:06:08

简介

C SDK 提供获取请求预签名 URL 接口，详细操作请查看本文说明和示例。

说明

建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。

如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。

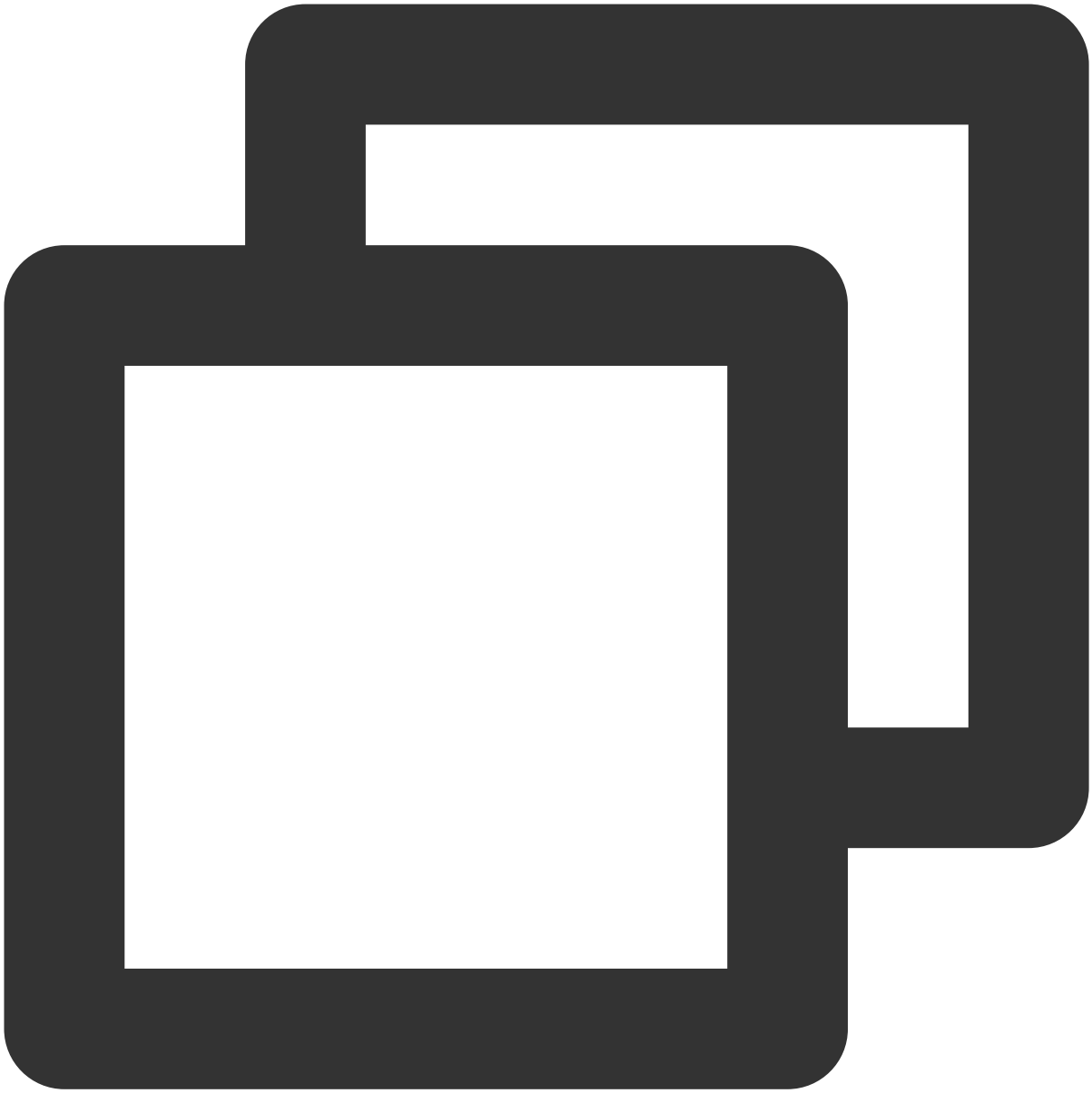
获取请求预签名 URL

生成请求预签名 URL

功能说明

该接口用于生成请求预签名 URL。

方法原型



```
int cos_gen_presigned_url(const cos_request_options_t *options,
                          const cos_string_t *bucket,
                          const cos_string_t *object,
                          const int64_t expire,
                          http_method_e method,
                          cos_string_t *presigned_url);
```

参数说明

参数名称	参数描述	类型

版权所有：腾讯云计算（北京）有限责任公司

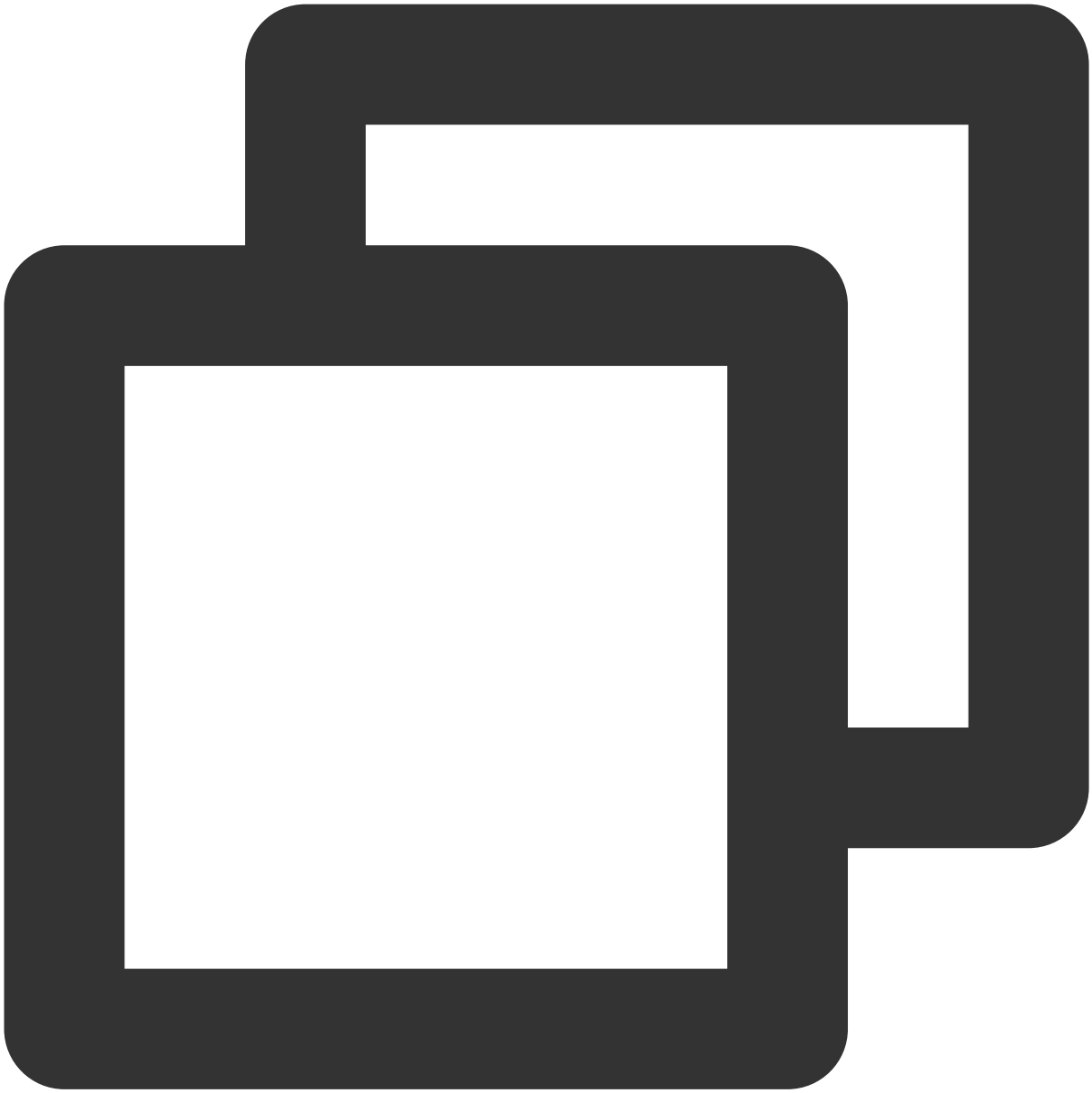
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
expire	签名有效时间，单位为秒	Int
method	HTTP 请求方法枚举类型，分别为 HTTP_GET、HTTP_HEAD、HTTP_PUT、HTTP_POST、HTTP_DELETE	Enum
presigned_url	生成的请求预签名 URL	String

返回结果说明

返回结果	描述	类型
code	错误码	Int

预签名请求示例

可在 options 参数中设置永久密钥或临时密钥来获取预签名 URL：



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_presigned_url()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t presigned_url;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_OBJECT_NAME1);

    cos_gen_presigned_url(options, &bucket, &object, 300, HTTP_GET, &presigned_url)
    printf("presigned_url: %s\\n", presigned_url.data);

    cos_pool_destroy(p);

}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
```

```
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_presigned_url();

    cos_http_io_deinitialize();

    return 0;
}
```

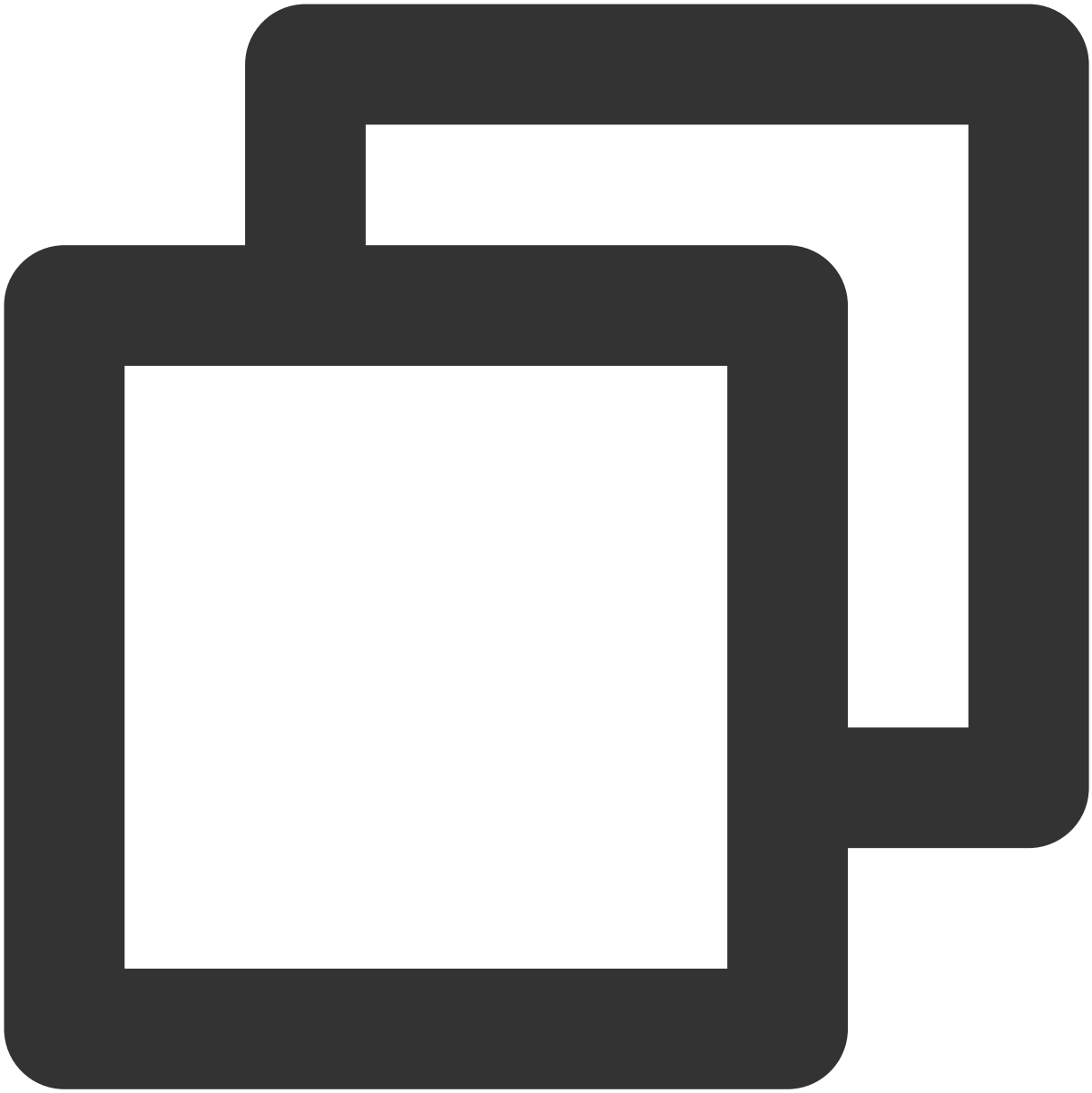
安全获取请求预签名 URL

安全生成请求预签名 URL

功能说明

该接口用于安全的生成请求预签名 URL。

方法原型



```
int cos_gen_presigned_url_safe(const cos_request_options_t *options,
                               const cos_string_t *bucket,
                               const cos_string_t *object,
                               const int64_t expire,
                               http_method_e method,
                               cos_table_t *headers,
                               cos_table_t *params,
                               int sign_host,
                               cos_string_t *presigned_url);
```

参数说明

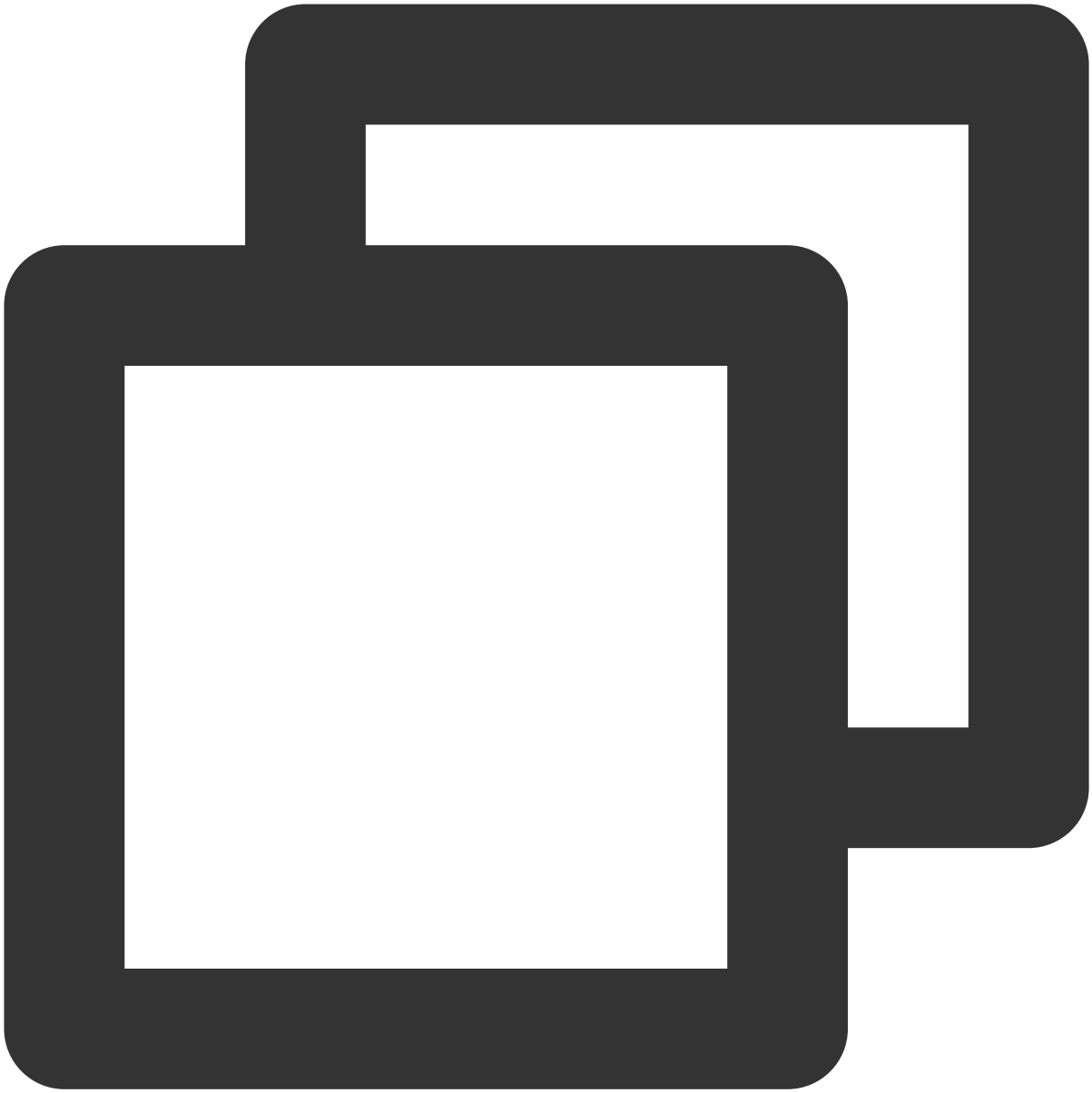
参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
expire	签名有效时间，单位为秒	Int
method	HTTP 请求方法枚举类型，分别为 HTTP_GET、HTTP_HEAD、HTTP_PUT、HTTP_POST、HTTP_DELETE	Enum
headers	COS 请求附加头域	Struct
params	COS 请求操作参数	Struct
sign_host	是否对 host 头域进行签名，为了安全性强烈建议开启	Int
presigned_url	生成的请求预签名 URL	String

返回结果说明

返回结果	描述	类型
code	错误码	Int

请求示例1: 生成预签名URL，并在签名中携带host

可在 options 参数中设置永久密钥或临时密钥来获取预签名 URL：



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_safe_presigned_url()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t presigned_url;
    cos_table_t *params = NULL;
    cos_table_t *headers = NULL;
    int sign_host = 1;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    /* 可以通过设置 sts_token 来使用临时密钥，当使用临时密钥时，access_key_id 和 access_key
    //cos_str_set(&options->config->sts_token, "MyTokenString");
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_OBJECT_NAME1);

    // 强烈建议sign_host为1，这样强制把host头域加入签名列表，防止越权访问问题
    cos_gen_presigned_url_safe(options, &bucket, &object, 300, HTTP_GET, headers, p
    printf("presigned_url_safe: %s\\n", presigned_url.data);

    cos_pool_destroy(p);
}
```

```
int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

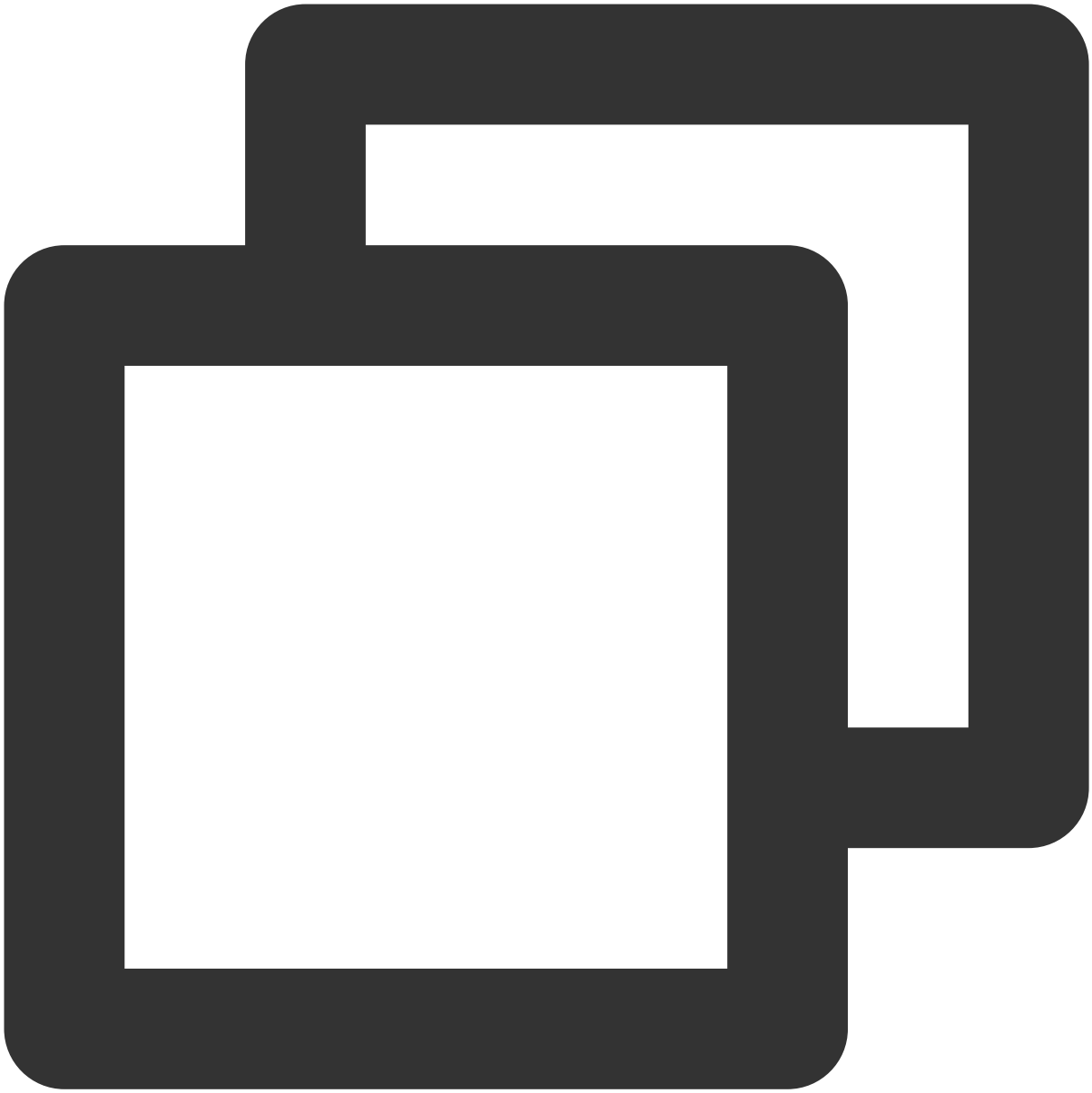
    test_safe_presigned_url();

    cos_http_io_deinitialize();

    return 0;
}
```

请求示例2: 生成预签名URL，并在签名中携带request param和request header

可在 options 参数中设置永久密钥或临时密钥来获取预签名 URL：



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_safe_presigned_url()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t presigned_url;
    cos_table_t *params = NULL;
    cos_table_t *headers = NULL;
    int sign_host = 1;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    /* 可以通过设置 sts_token 来使用临时密钥，当使用临时密钥时，access_key_id 和 access_key
    //cos_str_set(&options->config->sts_token, "MyTokenString");
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_OBJECT_NAME1);

    // 添加您自己的params和headers
    params = cos_table_make(options->pool, 0);
    //cos_table_add(params, "param1", "value");
    headers = cos_table_make(options->pool, 0);
    //cos_table_add(headers, "header1", "value");
```

```
    // 强烈建议sign_host为1，这样强制把host头域加入签名列表，防止越权访问问题
    cos_gen_presigned_url_safe(options, &bucket, &object, 300, HTTP_GET, headers, p
    printf("presigned_url_safe: %s\\n", presigned_url.data);

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_safe_presigned_url();

    cos_http_io_deinitialize();

    return 0;
}
```

服务端加密

最近更新时间：2024-01-04 17:06:08

简介

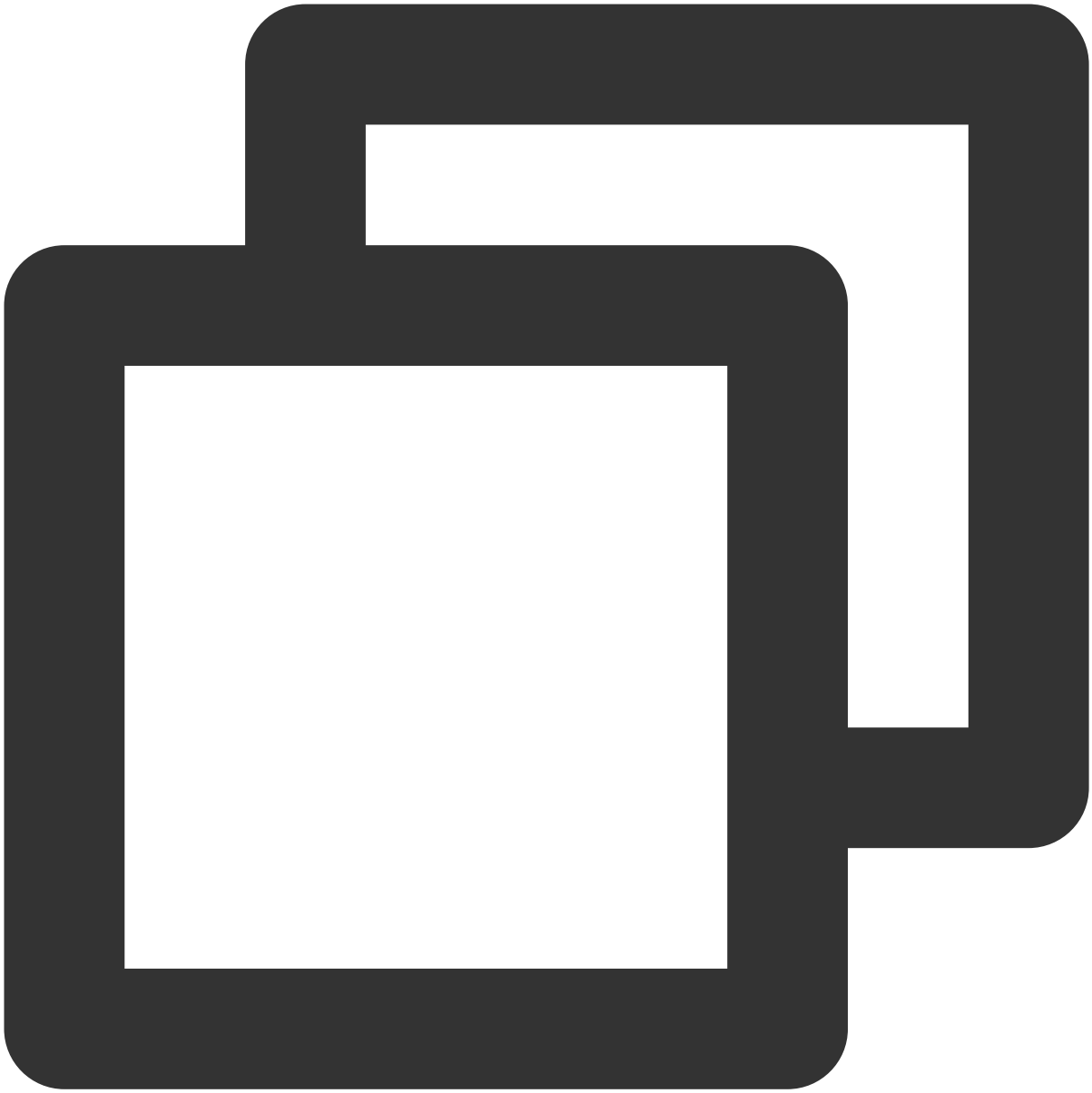
如果需要对上传的对象进行加密，我们支持以下加密方式。

使用 SSE-COS 加密保护数据

功能说明

由腾讯云 COS 托管主密钥和管理数据。COS 会帮助您在数据写入数据中心时自动加密，并在您取用该数据时自动解密。目前支持使用 COS 主密钥对数据进行 AES-256 加密。
C SDK 通过设置 `x-cos-server-side-encryption` 头部来完成。

示例



```
int main(int argc, char *argv[])
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_table_t *resp_headers = NULL;
    cos_table_t *headers = NULL;
    cos_string_t file;
```

```
    cos_string_t download_file;
    // 初始化
    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }
    cos_pool_create(&p, NULL);
    // 初始化请求选项
    options = cos_request_options_create(p);
    options->config = cos_config_create(options->pool);
    cos_str_set(&options->config->endpoint, "cos.ap-guangzhou.myqcloud.com"); // yo
    cos_str_set(&options->config->access_key_id, "SECRETID");   // your SecretId
    cos_str_set(&options->config->access_key_secret, "SECRETKEY"); // your SecretKe
    cos_str_set(&options->config->appid, "APPID");      //your appid
    options->config->is_cname = is_cname;
    options->ctl = cos_http_controller_create(options->pool, 0);
    cos_str_set(&bucket, "<BucketName-APPID>"); //your bucketname, <BucketName-APPI

    // 设置服务端加密头部, SSE-COS
    headers = cos_table_make(p, 1);
    apr_table_add(headers, "x-cos-server-side-encryption", "AES256");

    cos_str_set(&file, "example1.txt");          // your filename
    cos_str_set(&object, "example_object");      // your object name

    // 上传对象
    s = cos_put_object_from_file(options, &bucket, &object, &file, headers, &resp_h
    if (cos_status_is_ok(s)) {
        printf("put object succeeded\\n");
    } else {
        printf("put object failed\\n");
    }

    // 下载对象
    cos_str_set(&download_file, "example2.txt");
    s = cos_get_object_to_file(options, &bucket, &object, NULL, NULL, &download_fil
    if (cos_status_is_ok(s)) {
        printf("get object succeeded\\n");
    } else {
        printf("get object failed\\n");
    }

    cos_pool_destroy(p);

    cos_http_io_deinitialize();
    return 0;
}
```

使用 SSE-C 加密保护数据

加密密钥由用户自己提供，用户在上传对象时，COS 将使用用户提供的加密密钥对用户的数据进行 AES-256 加密。C SDK 通过设置 `x-cos-server-side-encryption-customer-*` 头部来完成。

功能说明

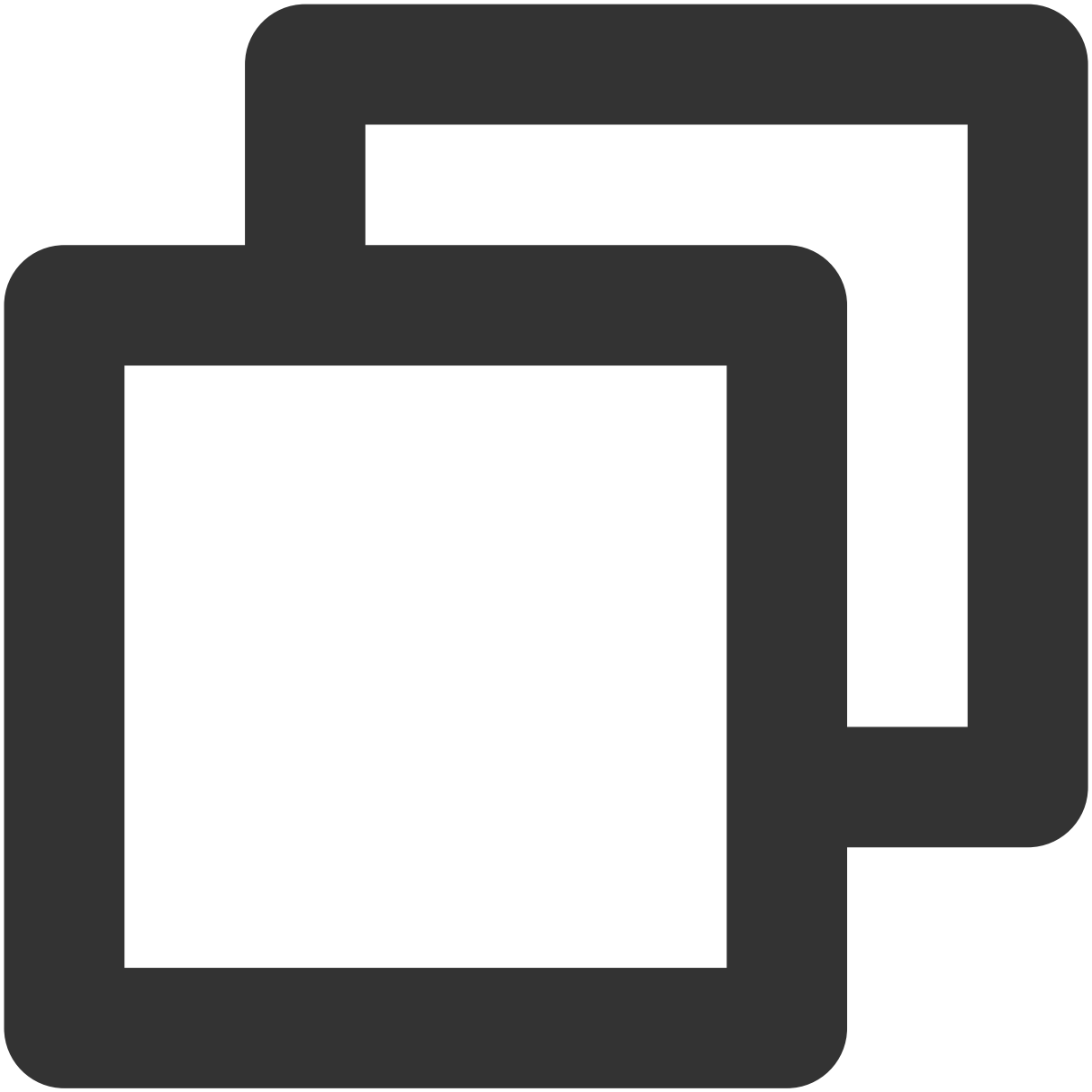
注意

该加密所运行的服务需要使用 HTTPS 请求。

customerKey：用户提供的密钥，传入一个32字节的字符串，支持数字、字母、字符的组合，不支持中文。

如果上传的源文件调用了该方法，那么在使用 GET（下载）、HEAD（查询）时对源对象操作的时候也要调用该方法。

示例



```
int main(int argc, char *argv[])
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_table_t *resp_headers = NULL;
    cos_table_t *headers = NULL;
    cos_string_t file;
```

版权所有：腾讯云计算（北京）有限责任公司

```
    cos_string_t download_file;
    // 初始化
    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
        exit(1);
    }
    cos_pool_create(&p, NULL);
    // 初始化请求选项
    options = cos_request_options_create(p);
    options->config = cos_config_create(options->pool);
    cos_str_set(&options->config->endpoint, "https://cos.ap-guangzhou.myqcloud.com"
    cos_str_set(&options->config->access_key_id, "SECRETID");    // your SecretId
    cos_str_set(&options->config->access_key_secret, "SECRETKEY"); // your SecretKe
    cos_str_set(&options->config->appid, "APPID");       //your appid
    options->config->is_cname = is_cname;
    options->ctl = cos_http_controller_create(options->pool, 0);
    cos_str_set(&bucket, "<BucketName-APPID>"); //your bucketname, <BucketName-APPI

    // 设置服务端加密头部, SSE-C
    headers = cos_table_make(p, 3);
    apr_table_add(headers, "x-cos-server-side-encryption-customer-algorithm", "AES2
    apr_table_add(headers, "x-cos-server-side-encryption-customer-key", "MDEyMzQ1Nj
    apr_table_add(headers, "x-cos-server-side-encryption-customer-key-MD5", "U5L61r

    cos_str_set(&file, "example1.txt");          // your filename
    cos_str_set(&object, "example_object");      // your object name

    // 上传对象
    s = cos_put_object_from_file(options, &bucket, &object, &file, headers, &resp_h
    if (cos_status_is_ok(s)) {
        printf("put object succeeded\\n");
    } else {
        printf("put object failed\\n");
    }

    // 下载对象, 下载对象也需要提供服务端加密头部
    cos_str_set(&download_file, "example2.txt");
    s = cos_get_object_to_file(options, &bucket, &object, headers, NULL, &download_
    if (cos_status_is_ok(s)) {
        printf("get object succeeded\\n");
    } else {
        printf("get object failed\\n");
    }

    cos_pool_destroy(p);

    cos_http_io_deinitialize();
    return 0;
```

```
}
```

版权所有：腾讯云计算（北京）有限责任公司

判断对象是否存在

最近更新时间：2024-01-04 17:06:08

简介

本文档提供快捷查询存储桶中某个对象是否存在的示例代码。

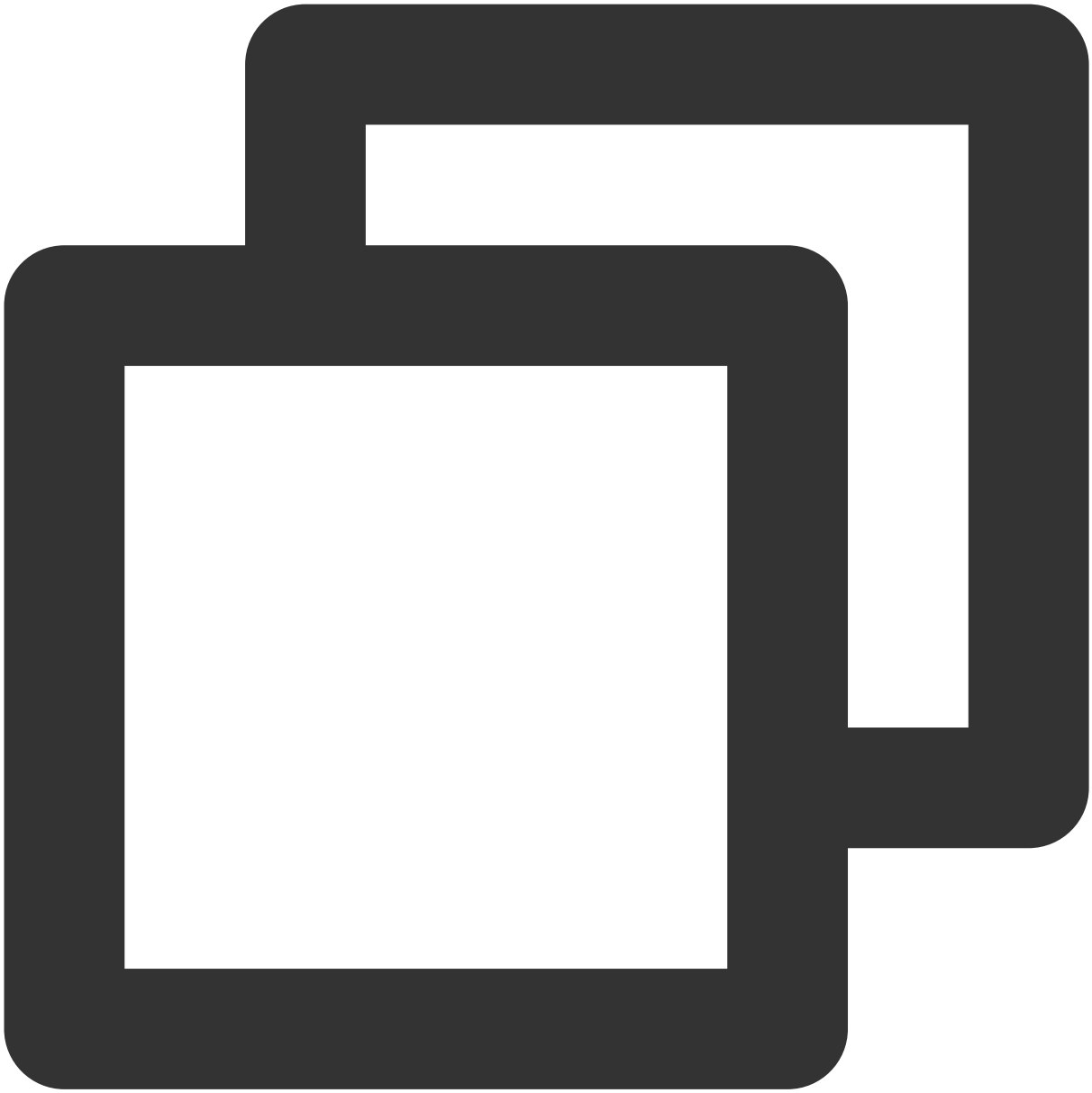
API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

检查对象是否存在

功能说明

检查存储桶中是否存在某个对象，此接口实际是调用了HEAD Object API来检查对象是否存在的。

方法原型



```
cos_status_t *cos_check_object_exist(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     const cos_string_t *object,
                                     cos_table_t *headers,
                                     cos_object_exist_status_e *object_exist,
                                     cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型

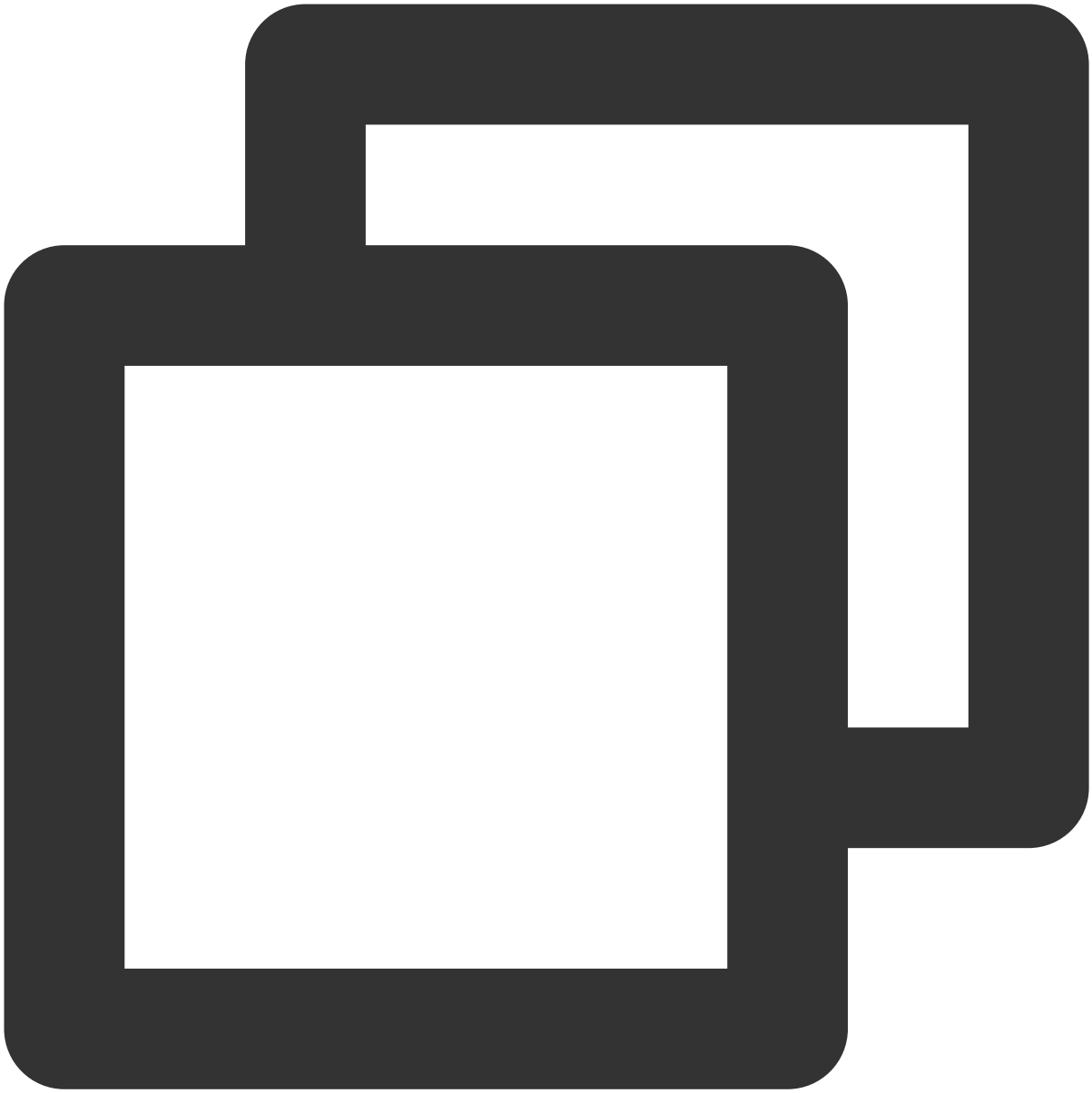
版权所有：腾讯云计算（北京）有限责任公司

options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
headers	COS 请求附加头域	Struct
object_exist	对象是否存在的枚举值，分为存在、不存在、未知状态(没有head到明确的状态)	Enum
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_check_object_exist()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_table_t *resp_headers;
    cos_table_t *headers = NULL;
    cos_object_exist_status_e object_exist;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_OBJECT_NAME1);
```

```
    // 检查对象是否存在
    s = cos_check_object_exist(options, &bucket, &object, headers, &object_exist, &
    if (object_exist == COS_OBJECT_NON_EXIST) {
        printf("object: %.*s non exist.\\n", object.len, object.data);
    } else if (object_exist == COS_OBJECT_EXIST) {
        printf("object: %.*s exist.\\n", object.len, object.data);
    } else {
        printf("object: %.*s unknown status.\\n", object.len, object.data);
        log_status(s);
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_check_object_exist();

    cos_http_io_deinitialize();

    return 0;
}
```

单链接限速

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于调用上传下载接口时对链接进行限速。

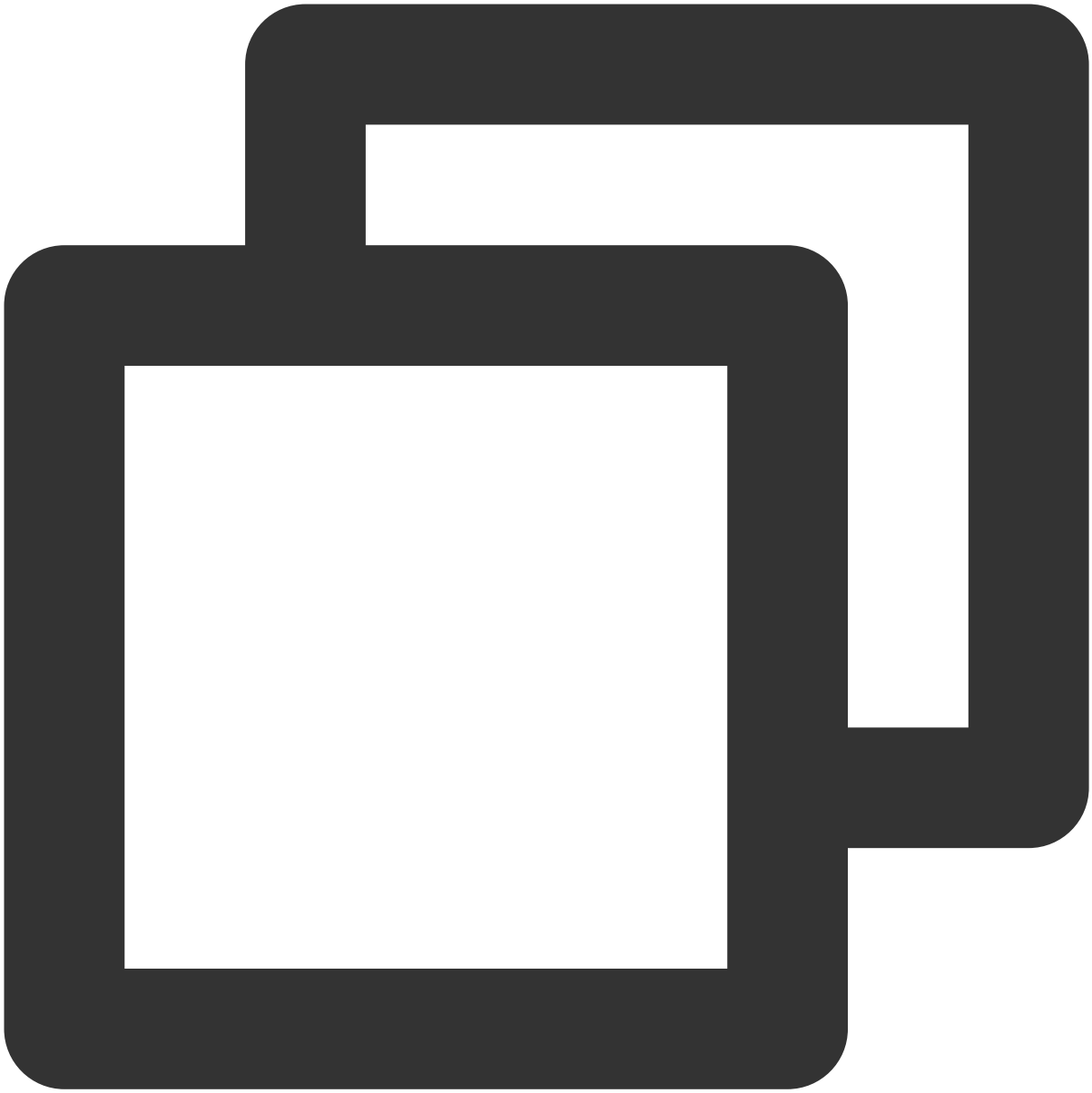
使用说明

限速值设置范围为**819200 - 838860800**，单位默认为 bit/s，即100KB/s - 100MB/s，如果超出该范围将返回400错误。

示例代码一：上传时对单链接限速

说明

关于上传和下载对象的限速说明，请参见 单链接限速。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_object_with_limit()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t file;
    cos_table_t *resp_headers = NULL;
    cos_table_t *headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，如果超出该范围将返回400错误
    headers = cos_table_make(p, 1);
    cos_table_add_int(headers, "x-cos-traffic-limit", 819200);

    //上传对象
    cos_str_set(&file, "test_file.bin");
```

```
    cos_str_set(&object, TEST_OBJECT_NAME1);
    s = cos_put_object_from_file(options, &bucket, &object, &file, headers, &resp_h
    if (cos_status_is_ok(s)) {
        printf("put object succeeded\\n");
    } else {
        printf("put object failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_object_with_limit();

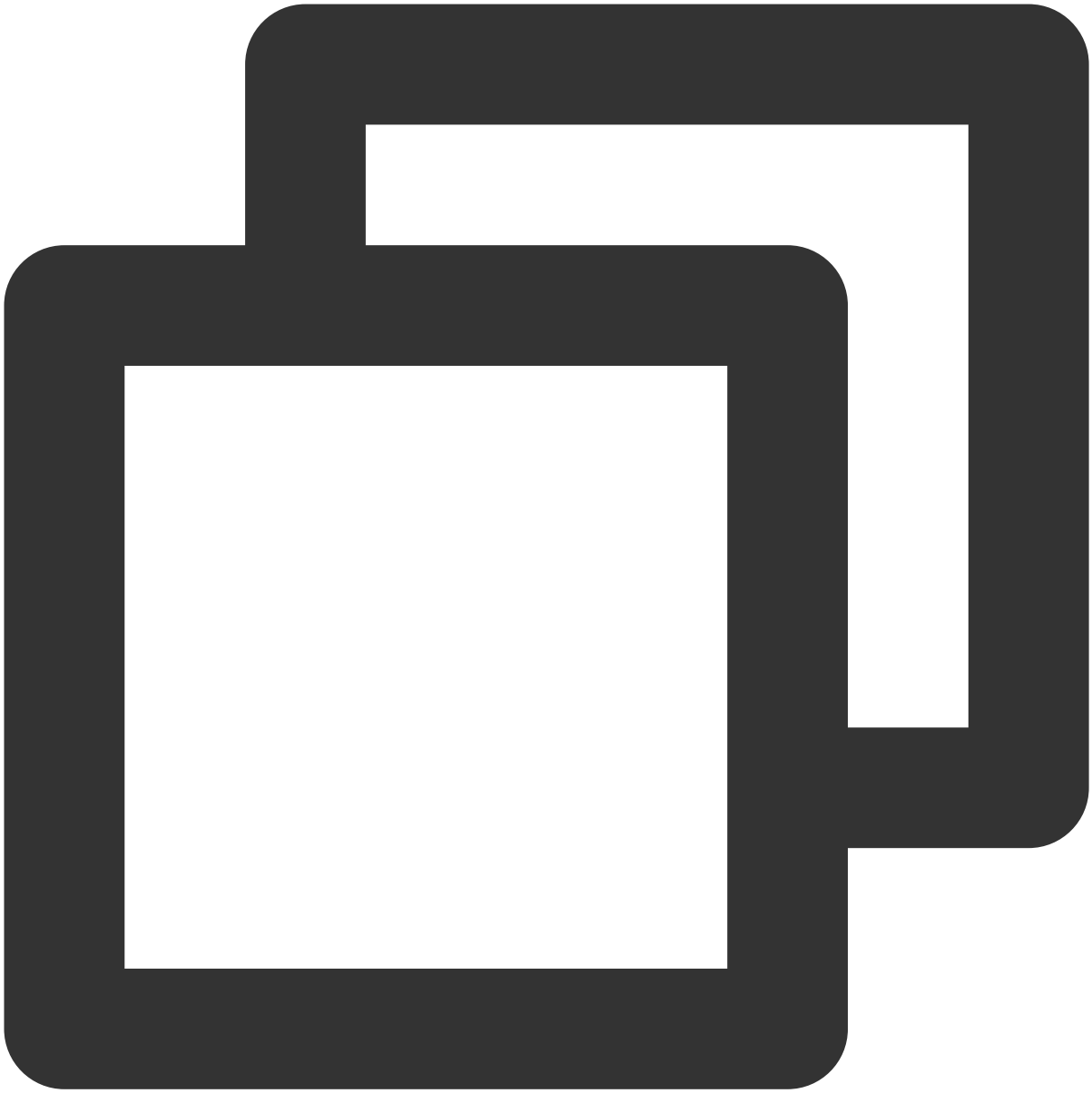
    cos_http_io_deinitialize();

    return 0;
}
```

示例代码二：下载时对单链接限速

说明

关于上传和下载对象的限速说明，请参见 单链接限速。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_get_object_with_limit()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t file;
    cos_table_t *resp_headers = NULL;
    cos_table_t *headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，如果超出该范围将返回400错误
    headers = cos_table_make(p, 1);
    cos_table_add_int(headers, "x-cos-traffic-limit", 819200);

    //下载对象
    cos_str_set(&file, "test_file.bin");
```

```
    cos_str_set(&object, TEST_OBJECT_NAME1);
    s = cos_get_object_to_file(options, &bucket, &object, headers, NULL, &file, &re
    if (cos_status_is_ok(s)) {
        printf("get object succeeded\\n");
    } else {
        printf("get object failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_get_object_with_limit();

    cos_http_io_deinitialize();

    return 0;
}
```

存储桶管理

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于跨域访问、生命周期、版本控制、跨地域复制相关的 API 概览以及 SDK 示例代码。

跨域访问

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域访问配置信息

生命周期

API	操作名	操作描述
PUT Bucket lifecycle	设置生命周期	设置存储桶的生命周期管理的配置
GET Bucket lifecycle	查询生命周期	查询存储桶生命周期管理的配置
DELETE Bucket lifecycle	删除生命周期	删除存储桶生命周期管理的配置

版本控制

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

跨地域复制

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

版权所有：腾讯云计算（北京）有限责任公司

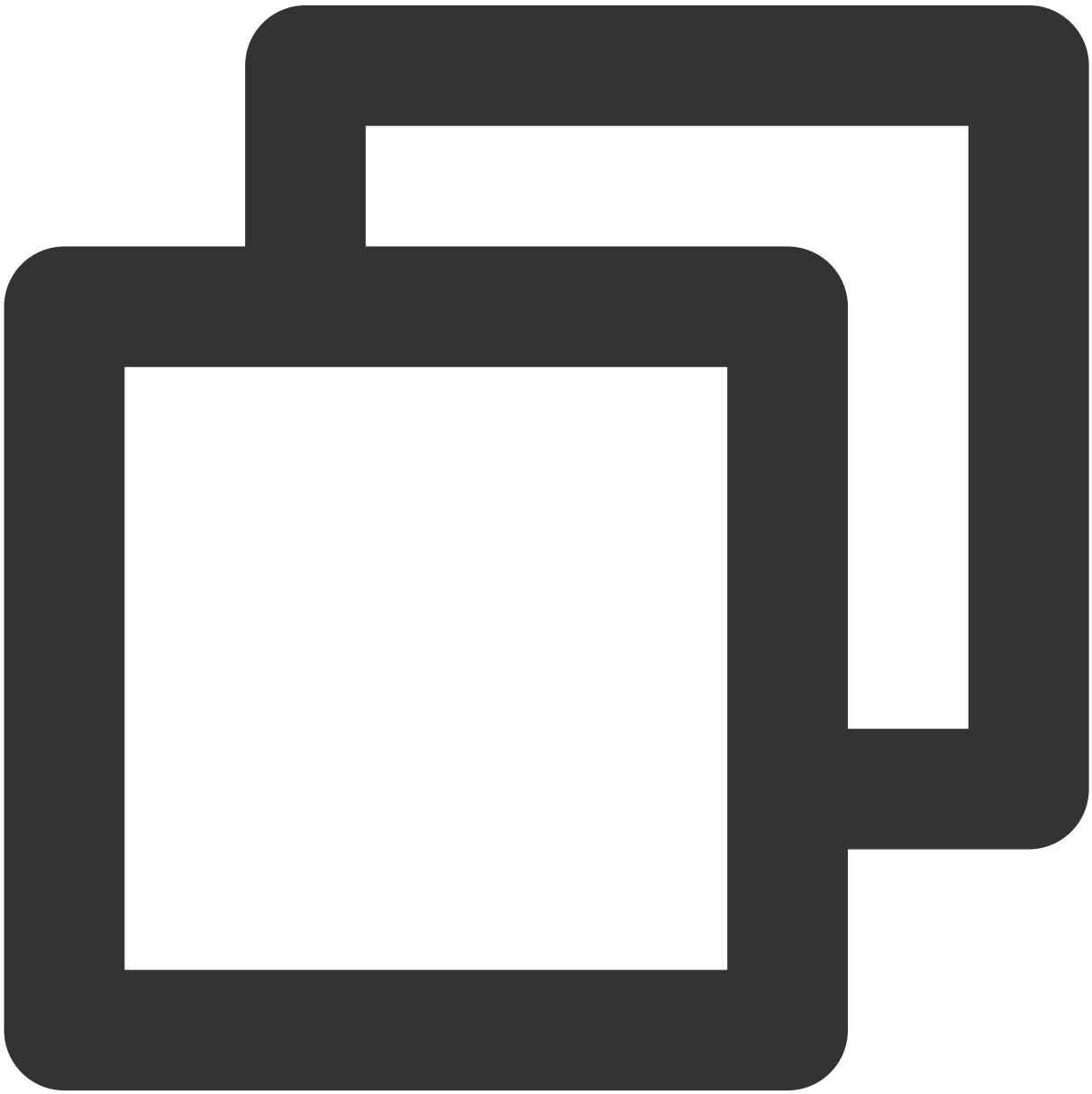
跨域访问

设置跨域配置

功能说明

设置存储桶的跨域访问权限。

方法原型



```
cos_status_t *cos_put_bucket_cors(const cos_request_options_t *options,
```

```
                                    const cos_string_t *bucket,
                                    cos_list_t *cors_rule_list,
                                    cos_table_t **resp_headers);
```

参数说明

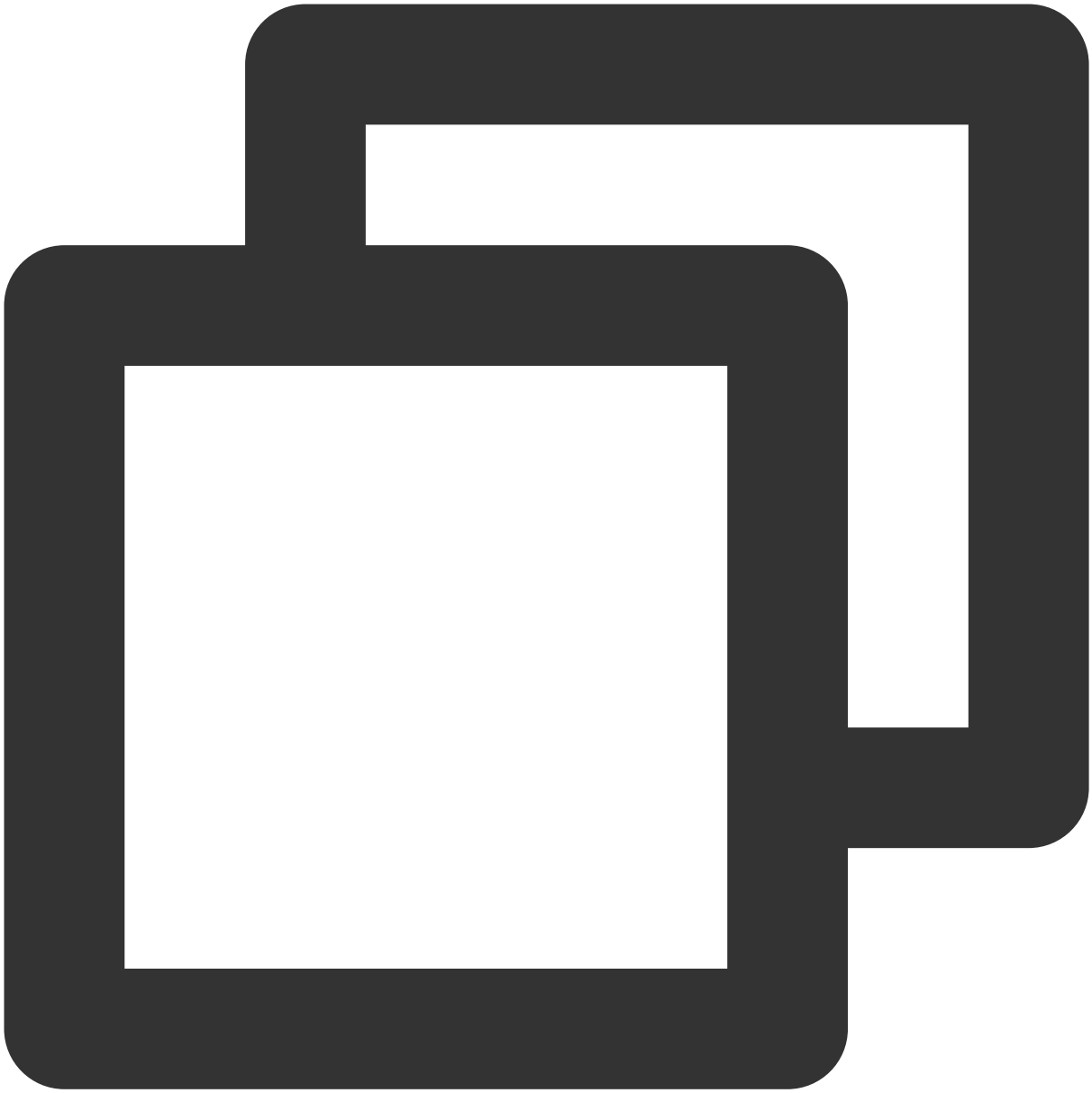
参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
cors_rule_list	存储桶跨域配置信息	Struct
id	配置规则 ID	String
allowed_origin	允许的访问来源，支持通配符 `*`	String
allowed_method	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	String
allowed_header	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 `*`	String
expose_header	设置浏览器可以接收到的来自服务器端的自定义头部信息	String
max_age_seconds	设置 OPTIONS 请求得到结果的有效期	Int
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

版权所有：腾讯云计算（北京）有限责任公司



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_cors()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    cos_list_t rule_list;
    cos_list_init(&rule_list);
    cos_cors_rule_content_t *rule_content = NULL;

    rule_content = cos_create_cors_rule_content(p);
```

```
    cos_str_set(&rule_content->id, "testrule1");
    cos_str_set(&rule_content->allowed_origin, "http://www.test1.com");
    cos_str_set(&rule_content->allowed_method, "GET");
    cos_str_set(&rule_content->allowed_header, "*");
    cos_str_set(&rule_content->expose_header, "xxx");
    rule_content->max_age_seconds = 3600;
    cos_list_add_tail(&rule_content->node, &rule_list);

    rule_content = cos_create_cors_rule_content(p);
    cos_str_set(&rule_content->id, "testrule2");
    cos_str_set(&rule_content->allowed_origin, "http://www.test2.com");
    cos_str_set(&rule_content->allowed_method, "GET");
    cos_str_set(&rule_content->allowed_header, "*");
    cos_str_set(&rule_content->expose_header, "yyy");
    rule_content->max_age_seconds = 7200;
    cos_list_add_tail(&rule_content->node, &rule_list);

    rule_content = cos_create_cors_rule_content(p);
    cos_str_set(&rule_content->id, "testrule3");
    cos_str_set(&rule_content->allowed_origin, "http://www.test3.com");
    cos_str_set(&rule_content->allowed_method, "GET");
    cos_str_set(&rule_content->allowed_header, "*");
    cos_str_set(&rule_content->expose_header, "zzz");
    rule_content->max_age_seconds = 60;
    cos_list_add_tail(&rule_content->node, &rule_list);

    //put cors
    s = cos_put_bucket_cors(options, &bucket, &rule_list, &resp_headers);
    log_status(s);

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
```

```
    cos_log_set_output(NULL);

    test_put_cors();

    cos_http_io_deinitialize();

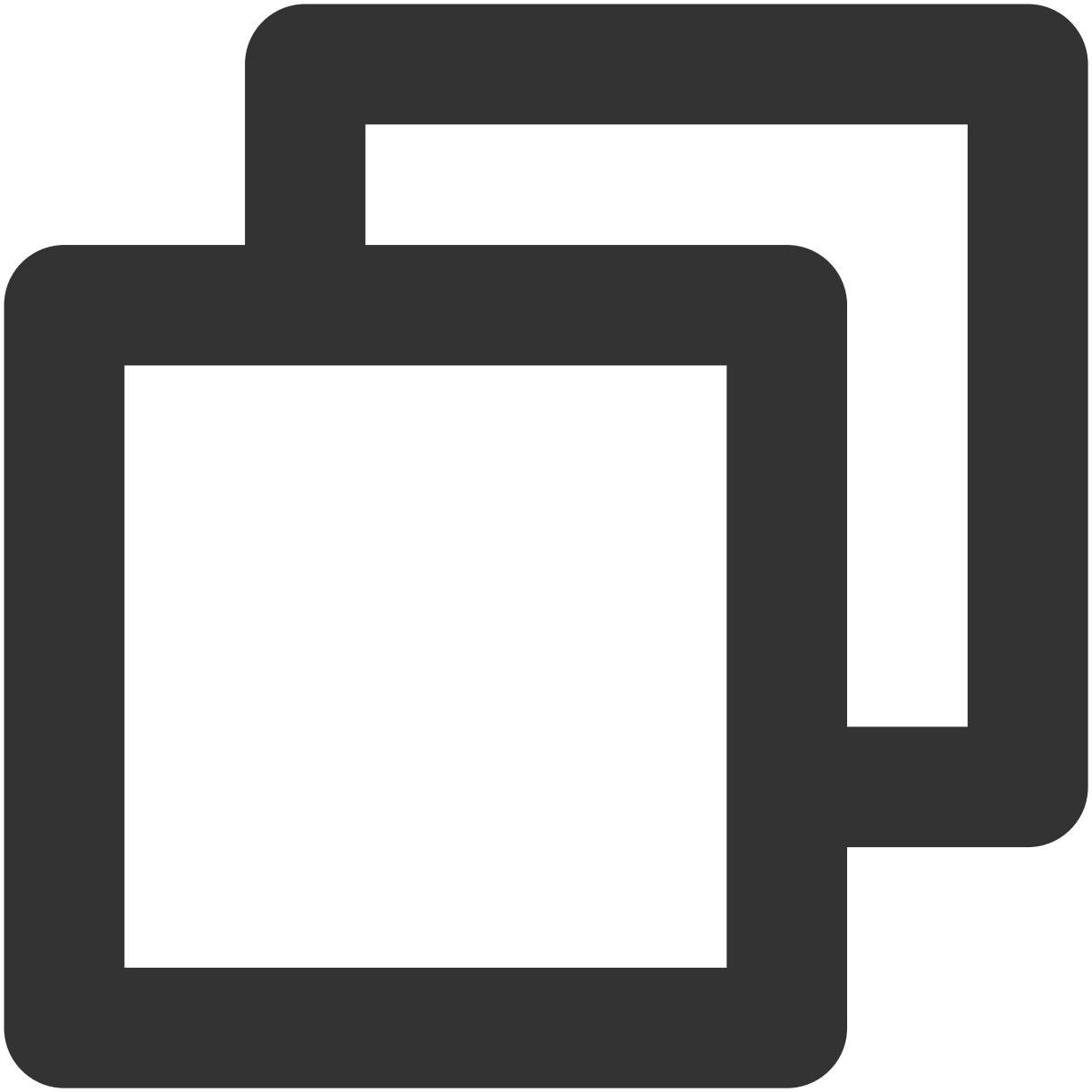
    return 0;
}
```

查询跨域配置

功能说明

查询存储桶的跨域访问配置信息。

方法原型



```
cos_status_t *cos_get_bucket_cors(const cos_request_options_t *options,
                                  const cos_string_t *bucket,
                                  cos_list_t *cors_rule_list,
                                  cos_table_t **resp_headers);
```

参数说明

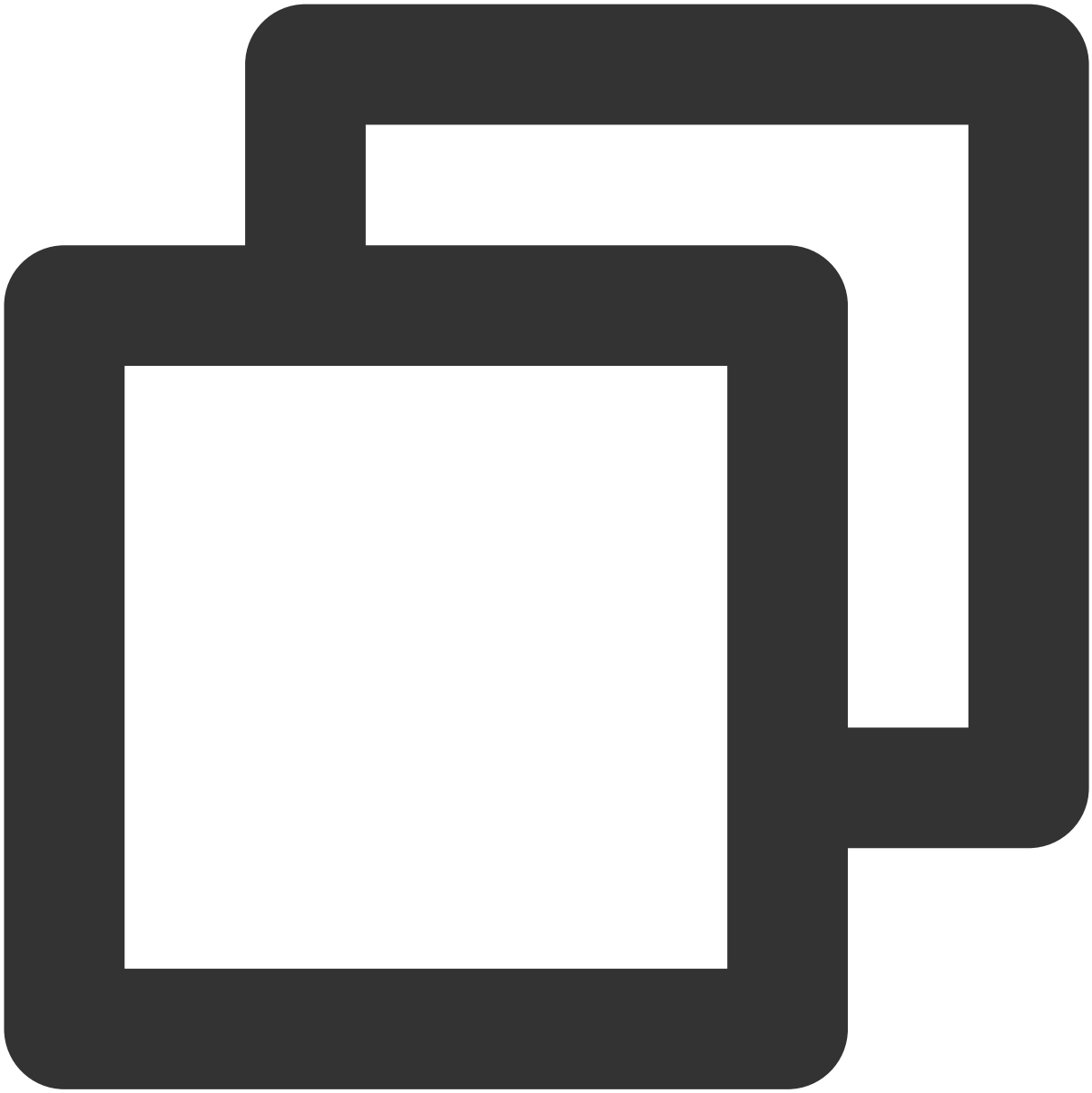
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
cors_rule_list	存储桶跨域配置信息	Struct
id	配置规则 ID	String
allowed_origin	允许的访问来源，支持通配符 `*`	String
allowed_method	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	String
allowed_header	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 `*`	String
expose_header	设置浏览器可以接收到的来自服务器端的自定义头部信息	String
max_age_seconds	设置 OPTIONS 请求得到结果的有效期	Int
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_get_cors()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    cos_list_t rule_list;
    cos_list_init(&rule_list);
    cos_cors_rule_content_t *rule_content = NULL;

    rule_content = cos_create_cors_rule_content(p);
```

```
    cos_str_set(&rule_content->id, "testrule1");
    cos_str_set(&rule_content->allowed_origin, "http://www.test1.com");
    cos_str_set(&rule_content->allowed_method, "GET");
    cos_str_set(&rule_content->allowed_header, "*");
    cos_str_set(&rule_content->expose_header, "xxx");
    rule_content->max_age_seconds = 3600;
    cos_list_add_tail(&rule_content->node, &rule_list);

    rule_content = cos_create_cors_rule_content(p);
    cos_str_set(&rule_content->id, "testrule2");
    cos_str_set(&rule_content->allowed_origin, "http://www.test2.com");
    cos_str_set(&rule_content->allowed_method, "GET");
    cos_str_set(&rule_content->allowed_header, "*");
    cos_str_set(&rule_content->expose_header, "yyy");
    rule_content->max_age_seconds = 7200;
    cos_list_add_tail(&rule_content->node, &rule_list);

    rule_content = cos_create_cors_rule_content(p);
    cos_str_set(&rule_content->id, "testrule3");
    cos_str_set(&rule_content->allowed_origin, "http://www.test3.com");
    cos_str_set(&rule_content->allowed_method, "GET");
    cos_str_set(&rule_content->allowed_header, "*");
    cos_str_set(&rule_content->expose_header, "zzz");
    rule_content->max_age_seconds = 60;
    cos_list_add_tail(&rule_content->node, &rule_list);

    //get cors
    cos_list_t rule_list_ret;
    cos_list_init(&rule_list_ret);
    s = cos_get_bucket_cors(options, &bucket, &rule_list_ret, &resp_headers);
    log_status(s);
    cos_cors_rule_content_t *content = NULL;
    cos_list_for_each_entry(cos_cors_rule_content_t, content, &rule_list_ret, node)
        printf("cors id:%s, allowed_origin:%s, allowed_method:%s, allowed_header:%s
                content->id.data, content->allowed_origin.data, content->allowed_me
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
```

```
        exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_get_cors();

    cos_http_io_deinitialize();

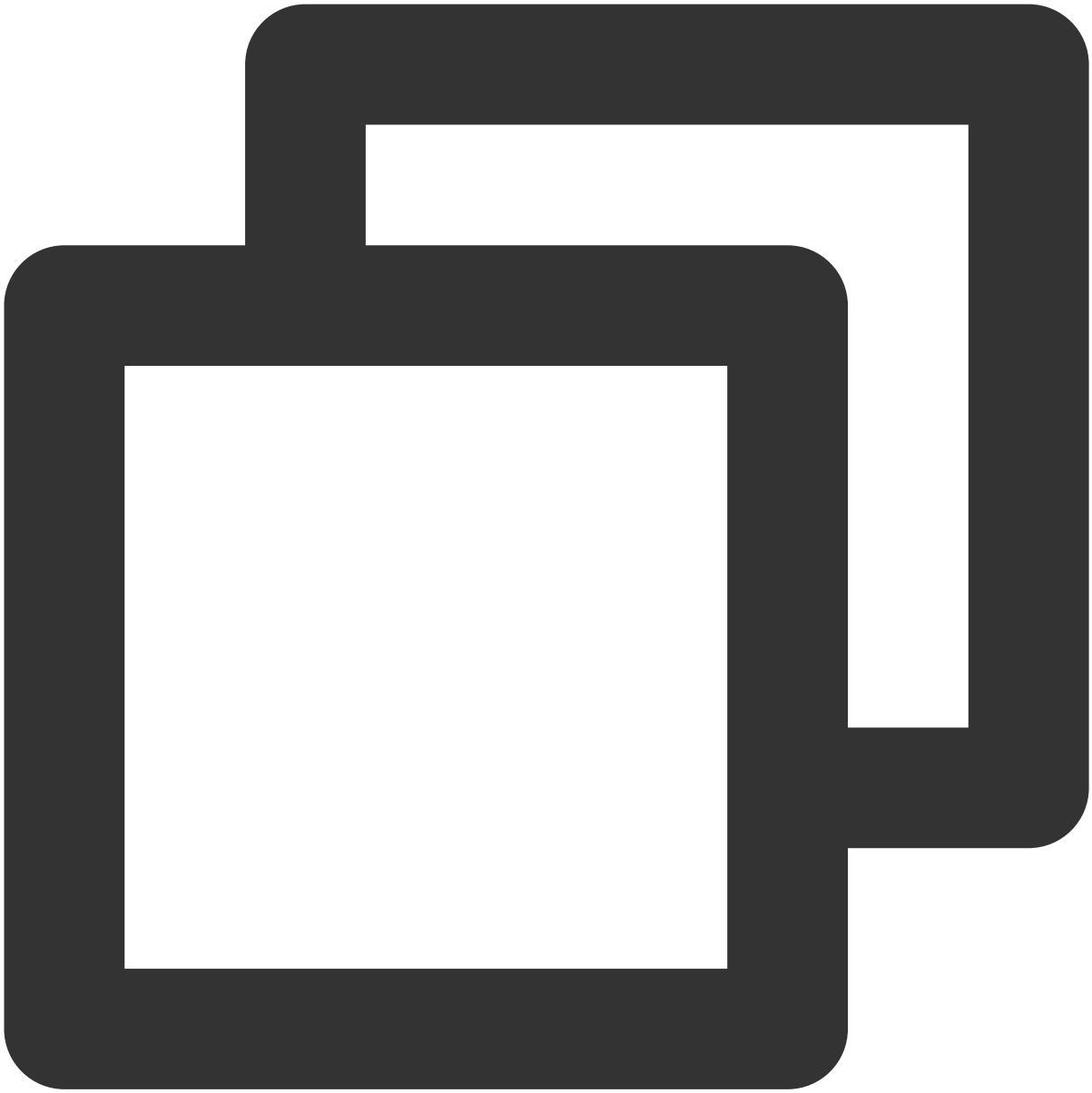
    return 0;
}
```

删除跨域配置

功能说明

删除存储桶的跨域访问配置信息。

方法原型



```
cos_status_t *cos_delete_bucket_cors(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     cos_table_t **resp_headers);
```

参数说明

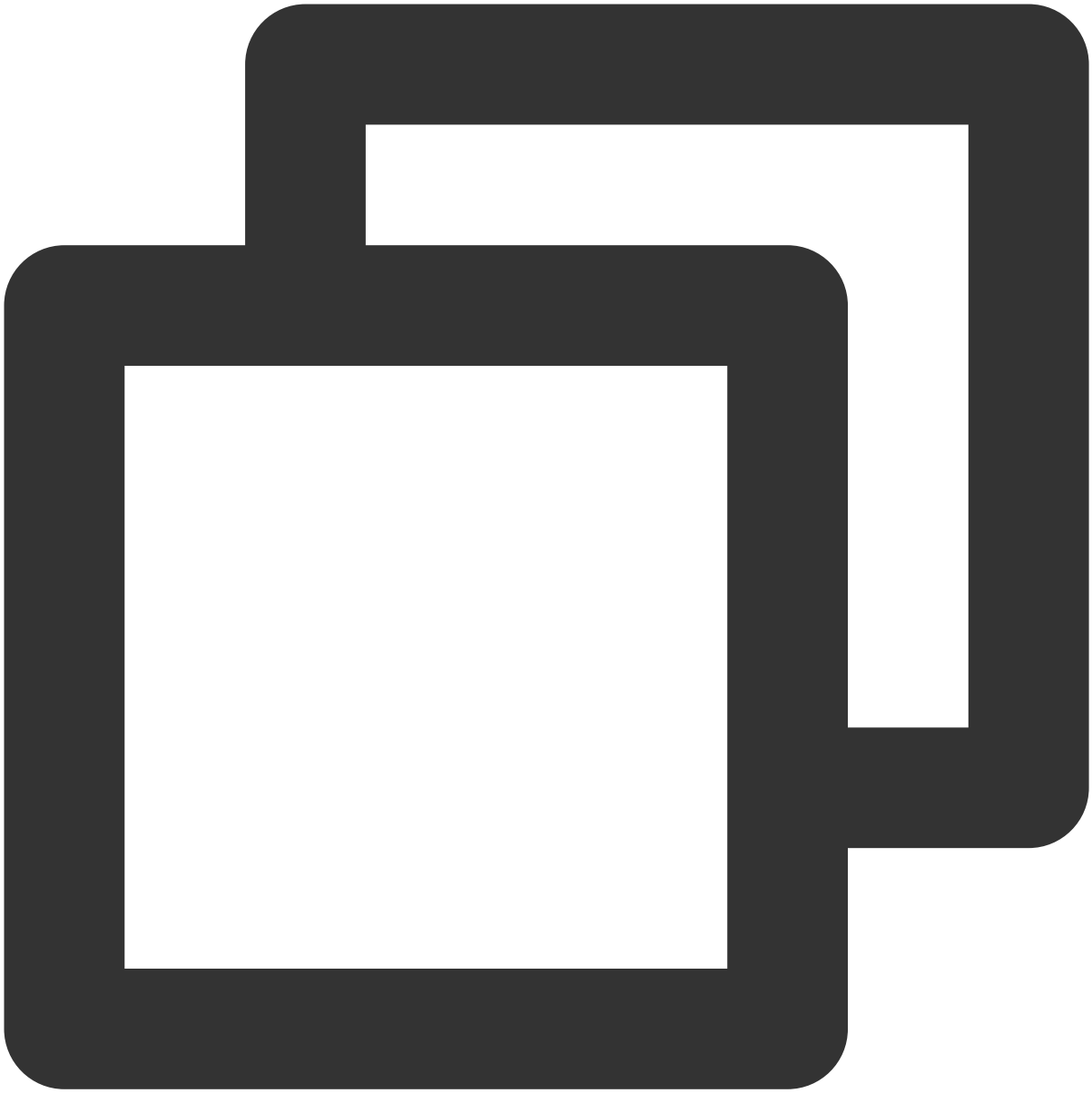
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_cors()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    cos_list_t rule_list;
    cos_list_init(&rule_list);
    cos_cors_rule_content_t *rule_content = NULL;

    rule_content = cos_create_cors_rule_content(p);
```

```
    cos_str_set(&rule_content->id, "testrule1");
    cos_str_set(&rule_content->allowed_origin, "http://www.test1.com");
    cos_str_set(&rule_content->allowed_method, "GET");
    cos_str_set(&rule_content->allowed_header, "*");
    cos_str_set(&rule_content->expose_header, "xxx");
    rule_content->max_age_seconds = 3600;
    cos_list_add_tail(&rule_content->node, &rule_list);

    rule_content = cos_create_cors_rule_content(p);
    cos_str_set(&rule_content->id, "testrule2");
    cos_str_set(&rule_content->allowed_origin, "http://www.test2.com");
    cos_str_set(&rule_content->allowed_method, "GET");
    cos_str_set(&rule_content->allowed_header, "*");
    cos_str_set(&rule_content->expose_header, "yyy");
    rule_content->max_age_seconds = 7200;
    cos_list_add_tail(&rule_content->node, &rule_list);

    rule_content = cos_create_cors_rule_content(p);
    cos_str_set(&rule_content->id, "testrule3");
    cos_str_set(&rule_content->allowed_origin, "http://www.test3.com");
    cos_str_set(&rule_content->allowed_method, "GET");
    cos_str_set(&rule_content->allowed_header, "*");
    cos_str_set(&rule_content->expose_header, "zzz");
    rule_content->max_age_seconds = 60;
    cos_list_add_tail(&rule_content->node, &rule_list);

    //delete cors
    cos_delete_bucket_cors(options, &bucket, &resp_headers);
    log_status(s);

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
        exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
```

```
    cos_log_set_output(NULL);

    test_delete_cors();

    cos_http_io_deinitialize();

    return 0;
}
```

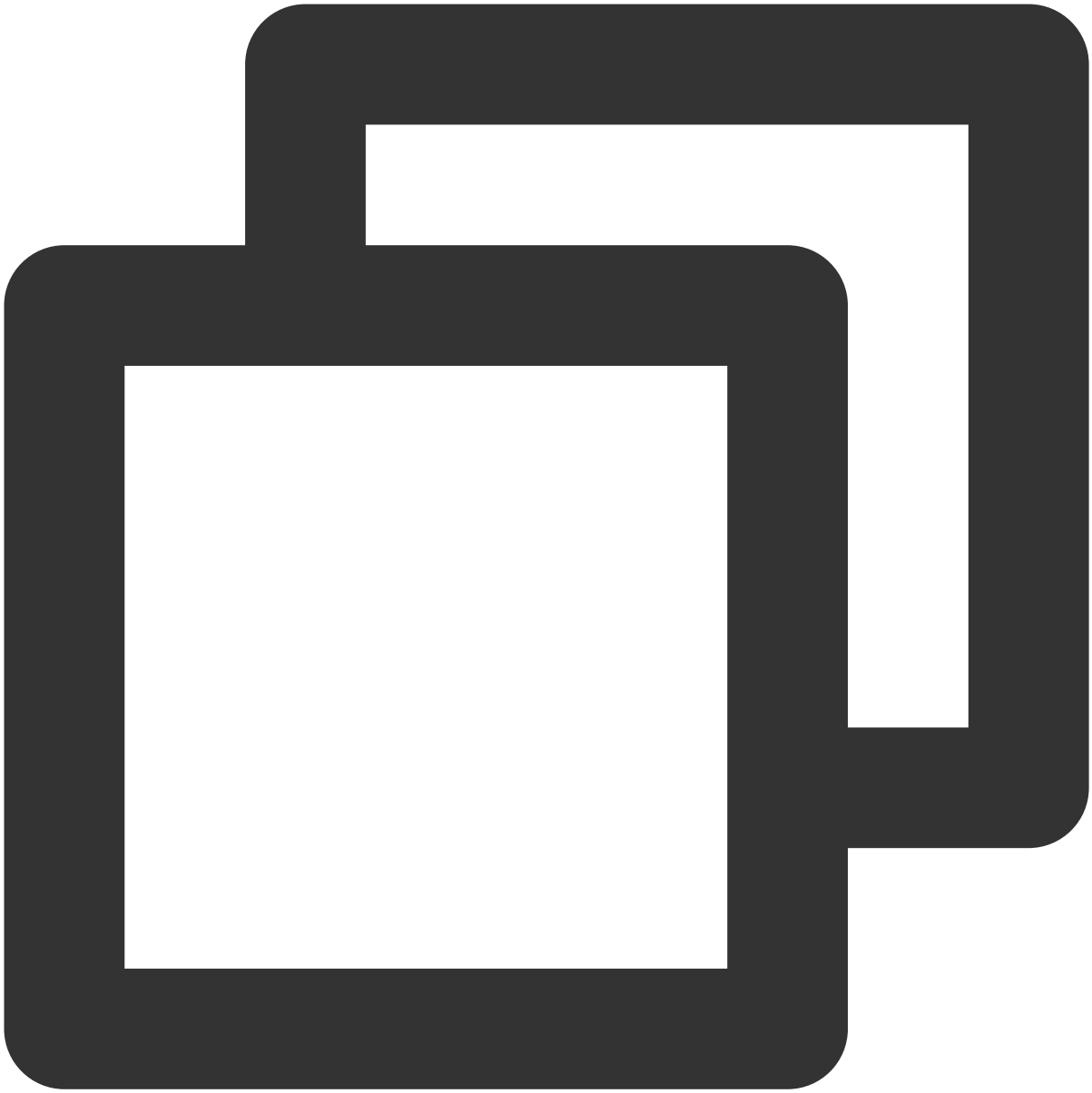
生命周期

设置生命周期

功能说明

设置存储桶生命周期管理的配置。

方法原型



```
cos_status_t *cos_put_bucket_lifecycle(const cos_request_options_t *options,
                                       const cos_string_t *bucket,
                                       cos_list_t *lifecycle_rule_list,
                                       cos_table_t **resp_headers);
```

参数说明

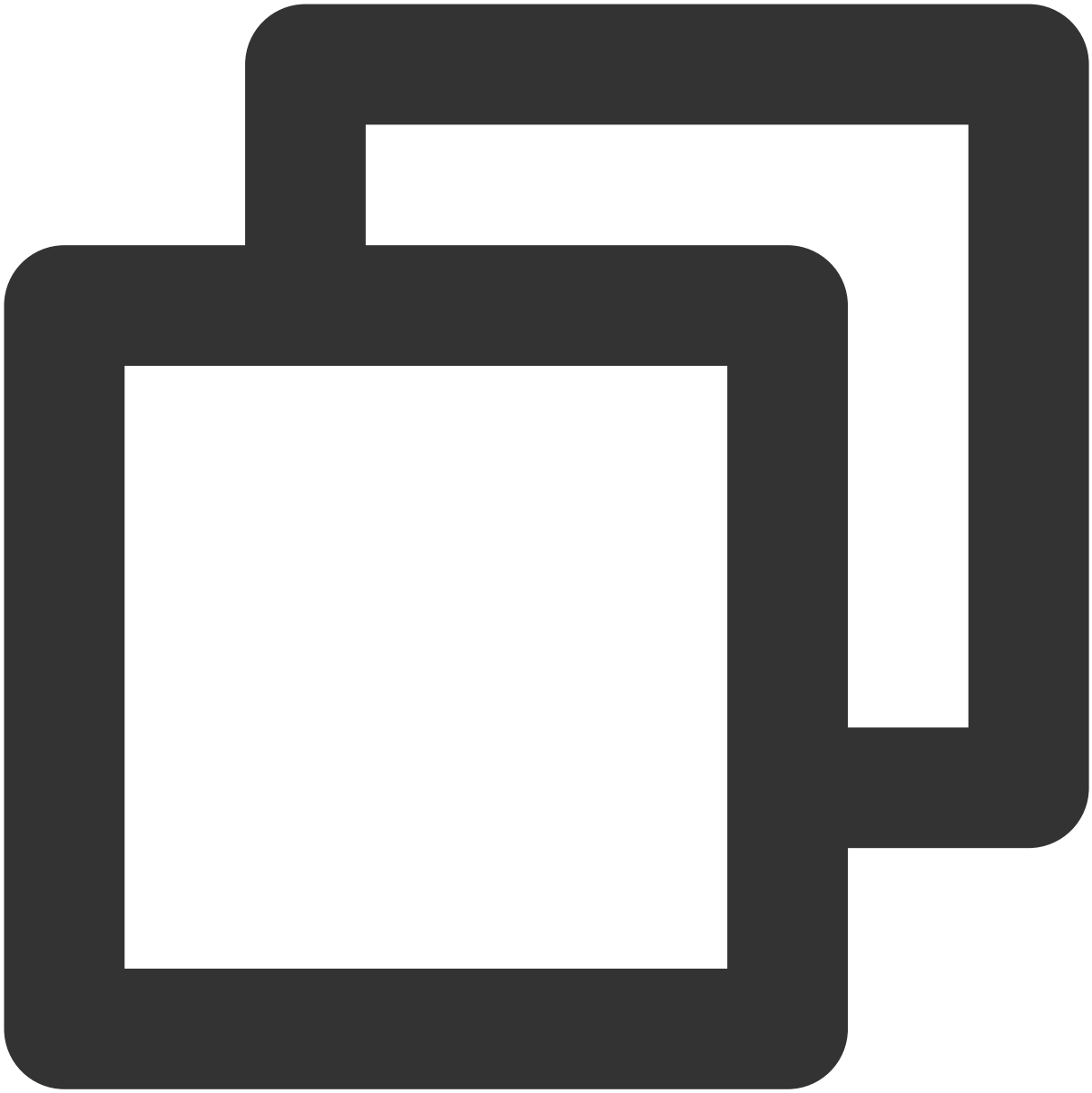
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
lifecycle_rule_list	生命周期规则信息	Struct
id	生命周期规则 ID	String
prefix	指定规则所适用的前缀	String
status	指明规则是否启用，枚举值：Enabled，Disabled	String
expire	规则过期属性	Struct
days	指明多少天后执行删除操作或转换操作；指明分块上传开始后多少天内必须完成上传	Int
date	指明在何时执行删除操作或转换操作	String
transition	规则转换属性，对象何时转换被转换为 Standard_IA、Archive	Struct
storage_class	指定 Object 转储到的目标存储类型，枚举值：Standard_IA、Archive	String
abort	设置允许分块上传保持运行的最长时间	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_bucket_lifecycle()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

    //初始化请求选项
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //设置生命周期信息
    cos_list_t rule_list;
    cos_list_init(&rule_list);
    cos_lifecycle_rule_content_t *rule_content = NULL;
```

```
    rule_content = cos_create_lifecycle_rule_content(p);
    cos_str_set(&rule_content->id, "testrule1");
    cos_str_set(&rule_content->prefix, "abc/");
    cos_str_set(&rule_content->status, "Enabled");
    rule_content->expire.days = 365;
    cos_list_add_tail(&rule_content->node, &rule_list);

    rule_content = cos_create_lifecycle_rule_content(p);
    cos_str_set(&rule_content->id, "testrule2");
    cos_str_set(&rule_content->prefix, "efg/");
    cos_str_set(&rule_content->status, "Disabled");
    cos_str_set(&rule_content->transition.storage_class, "Standard_IA");
    rule_content->transition.days = 999;
    cos_list_add_tail(&rule_content->node, &rule_list);

    rule_content = cos_create_lifecycle_rule_content(p);
    cos_str_set(&rule_content->id, "testrule3");
    cos_str_set(&rule_content->prefix, "xxx/");
    cos_str_set(&rule_content->status, "Enabled");
    rule_content->abort.days = 1;
    cos_list_add_tail(&rule_content->node, &rule_list);

    //设置生命周期
    s = cos_put_bucket_lifecycle(options, &bucket, &rule_list, &resp_headers);
    log_status(s);

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_bucket_lifecycle();
```

```
    cos_http_io_deinitialize();

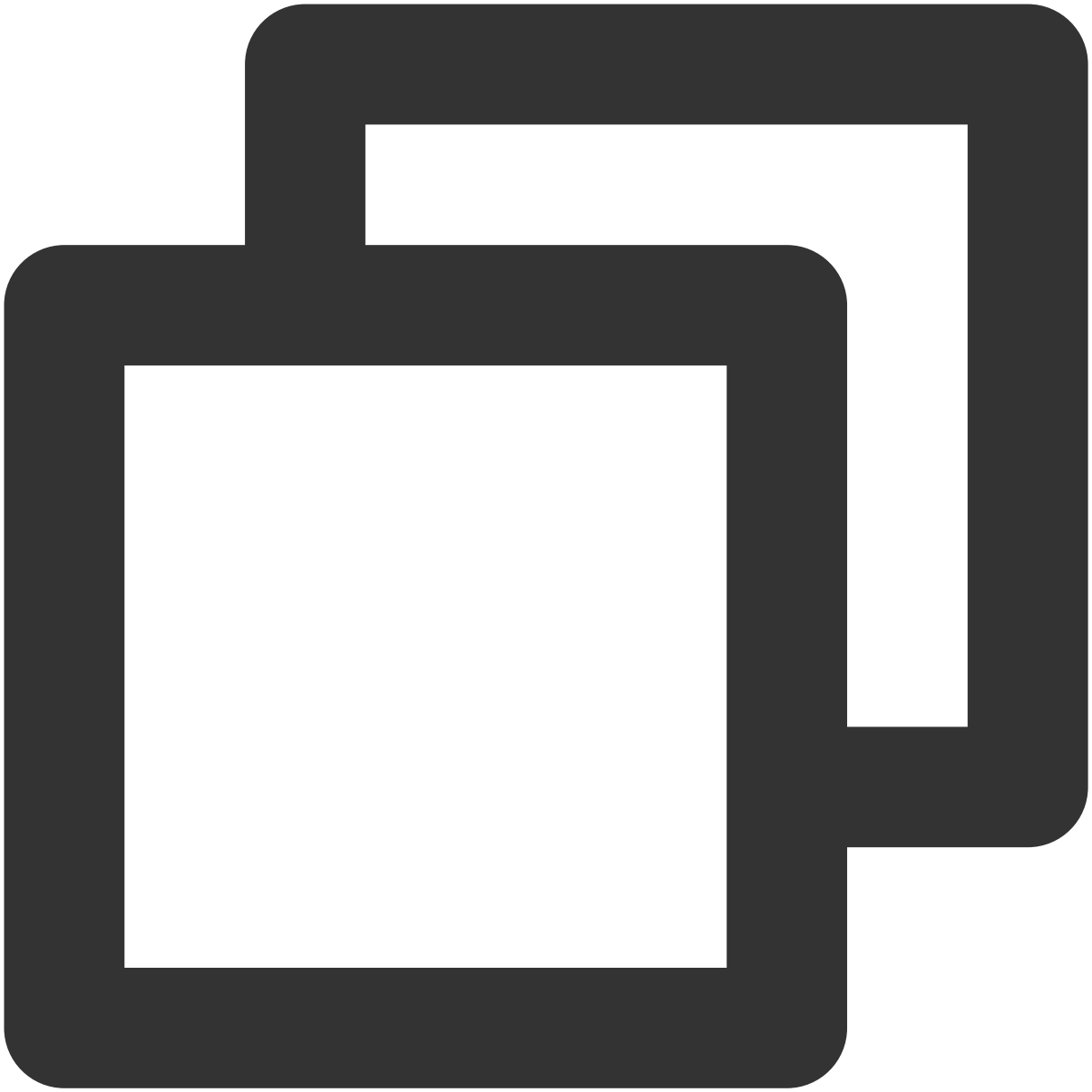
    return 0;
}
```

查询生命周期

功能说明

查询存储桶生命周期管理的配置。

方法原型



```
cos_status_t *cos_get_bucket_lifecycle(const cos_request_options_t *options,
                                       const cos_string_t *bucket,
                                       cos_list_t *lifecycle_rule_list,
                                       cos_table_t **resp_headers);
```

参数说明

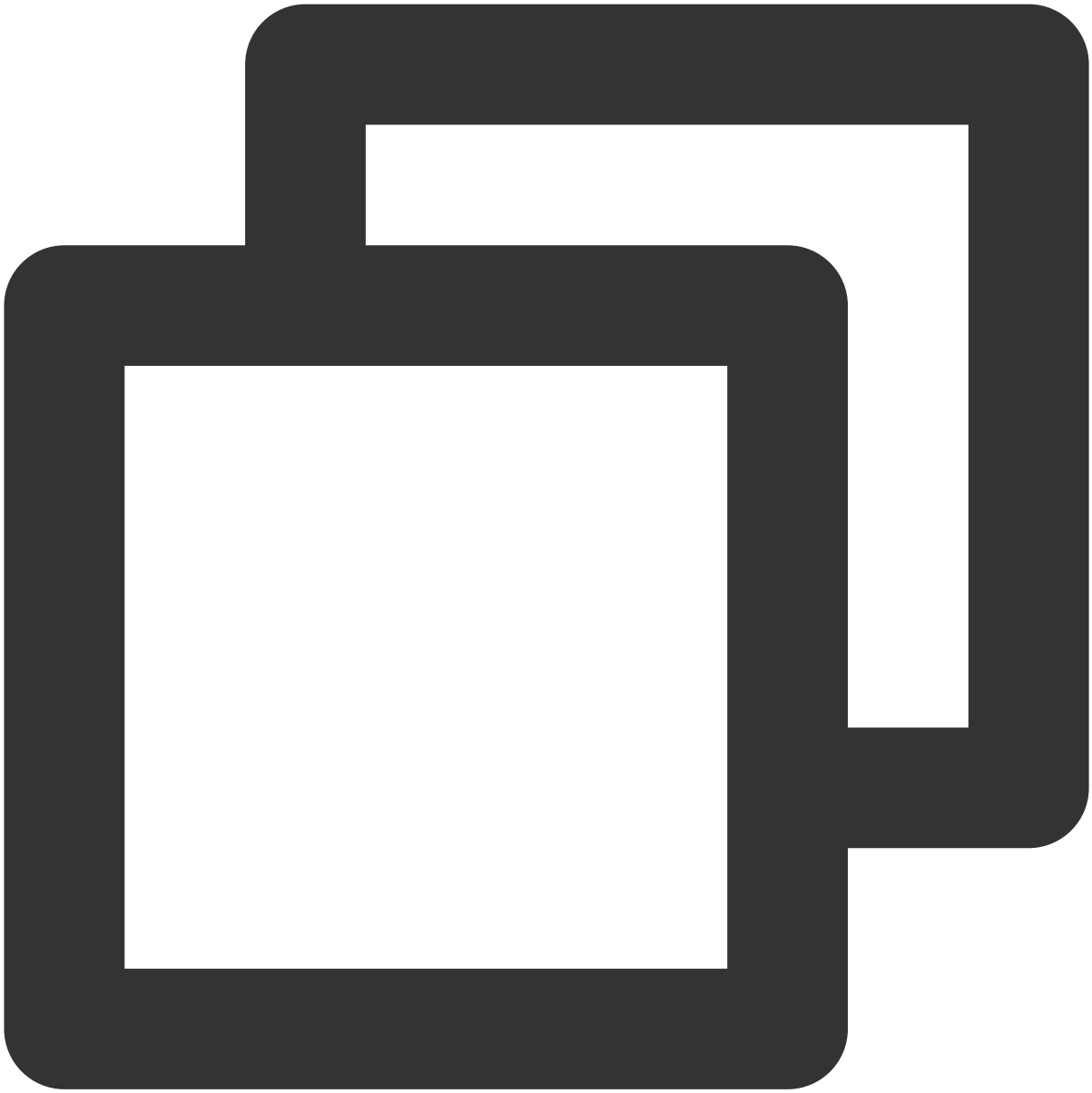
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
lifecycle_rule_list	生命周期规则信息	Struct
id	生命周期规则 ID	String
prefix	规则所适用的前缀	String
status	规则是否启用，枚举值：Enabled，Disabled	String
expire	规则过期属性	Struct
days	指明多少天后执行删除操作	Int
date	指明在何时执行删除操作	String
transition	规则转换属性，对象何时转换被转换为 Standard_IA、Archive	Struct
days	指明多少天后执行转换操作	Int
date	指明在何时执行转换操作	String
storage_class	指定 Object 转储到的目标存储类型，枚举值：Standard_IA、Archive	String
abort	设置允许分块上传保持运行的最长时间	Struct
days	指明分块上传开始后多少天内必须完成上传	Int
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_get_bucket_lifecycle()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

    //初始化请求选项
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //查询生命周期
    cos_list_t rule_list_ret;
    cos_list_init(&rule_list_ret);
    s = cos_get_bucket_lifecycle(options, &bucket, &rule_list_ret, &resp_headers);
```

```
    if (cos_status_is_ok(s)) {
            printf("get bucket lifecycle succeeded\\n");
    } else {
            printf("get bucket lifecycle failed\\n");
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_get_bucket_lifecycle();

    cos_http_io_deinitialize();

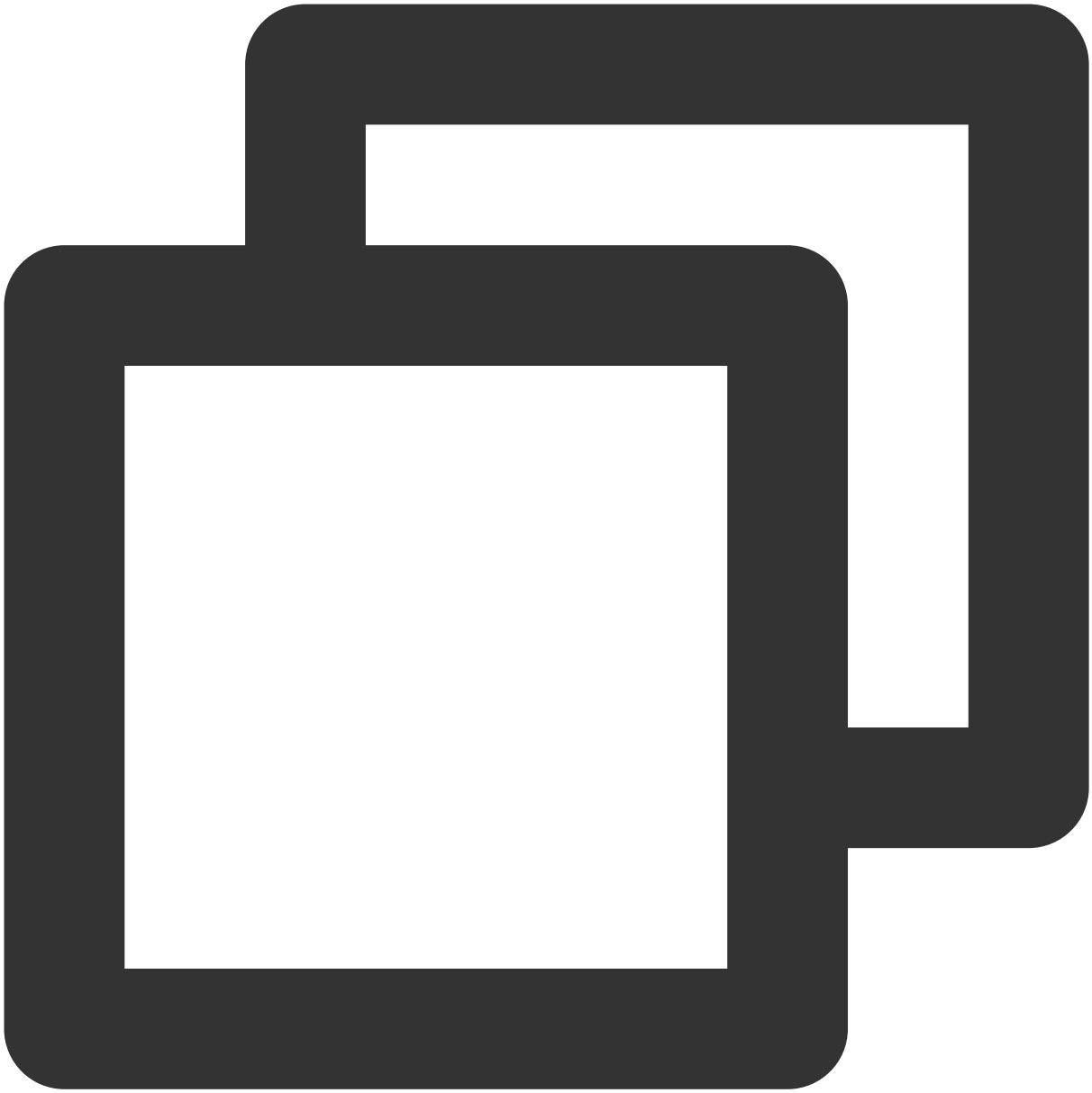
    return 0;
}
```

删除生命周期

功能说明

删除生命周期接口用来删除存储桶的生命周期规则。

方法原型



```
cos_status_t *cos_delete_bucket_lifecycle(const cos_request_options_t *options,
                                          const cos_string_t *bucket,
                                          cos_table_t **resp_headers);
```

参数说明

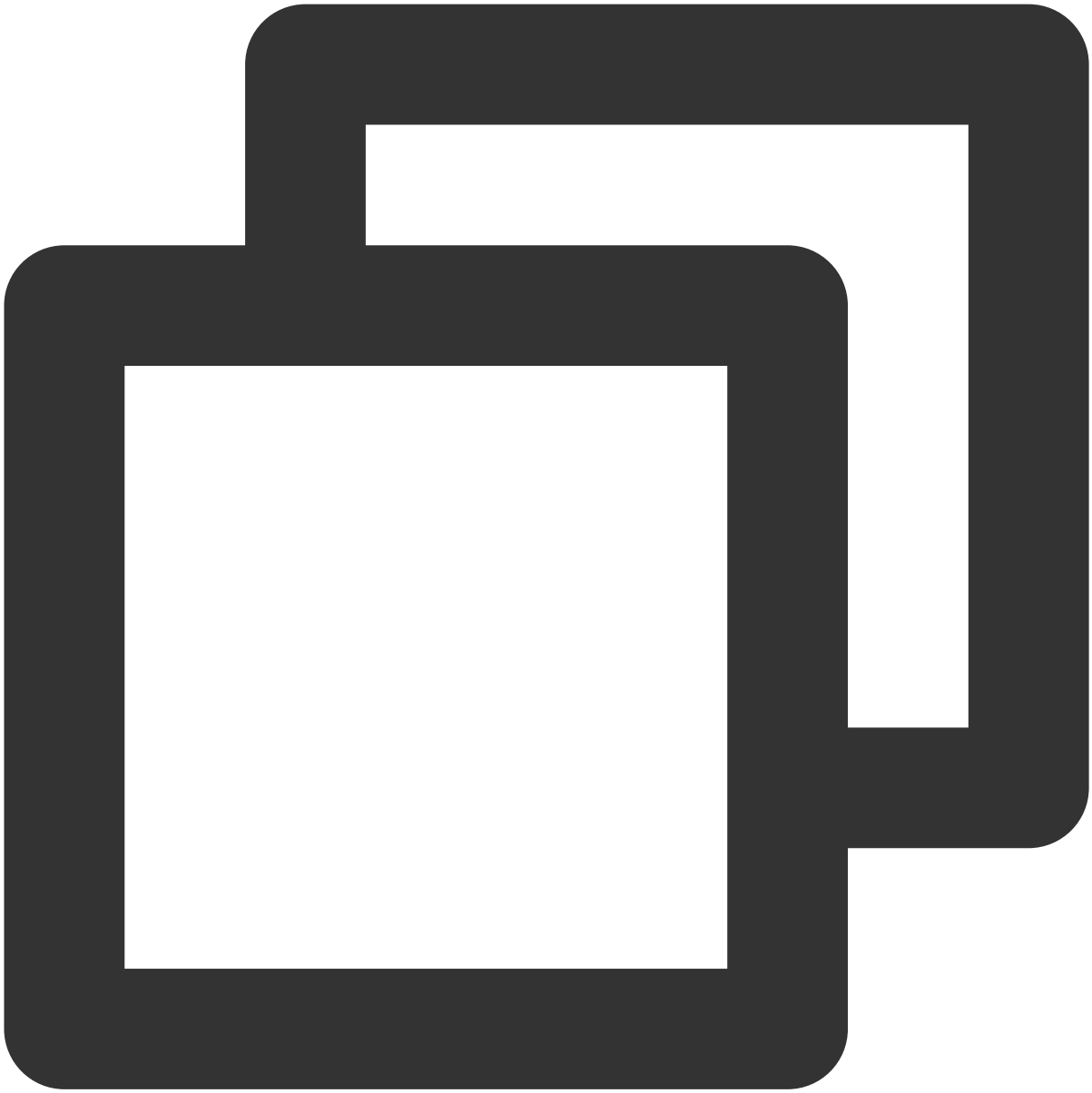
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
resp_headers	返回 HTTP 响应消息的头	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_bucket_lifecycle()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

    //初始化请求选项
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //删除生命周期
    s = cos_delete_bucket_lifecycle(options, &bucket, &resp_headers);
    if (cos_status_is_ok(s)) {
            printf("delete bucket lifecycle succeeded\\n");
```

```
    } else {
            printf("delete bucket lifecycle failed\\n");
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_delete_bucket_lifecycle();

    cos_http_io_deinitialize();

    return 0;
}
```

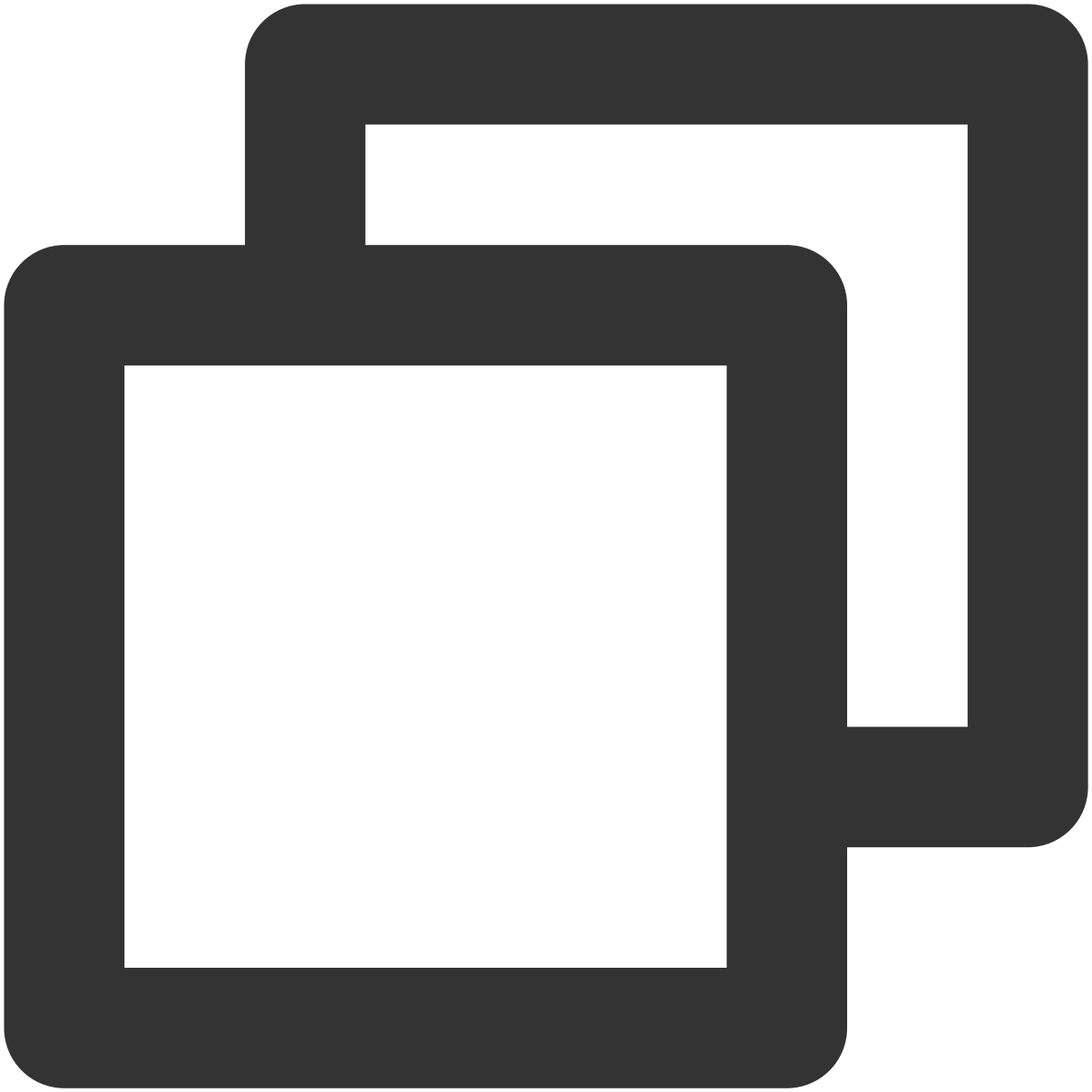
版本控制

设置版本控制

功能说明

设置指定存储桶的版本控制功能。

方法原型



```
cos_status_t *cos_put_bucket_versioning(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        cos_versioning_content_t *versioning,
                                        cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct

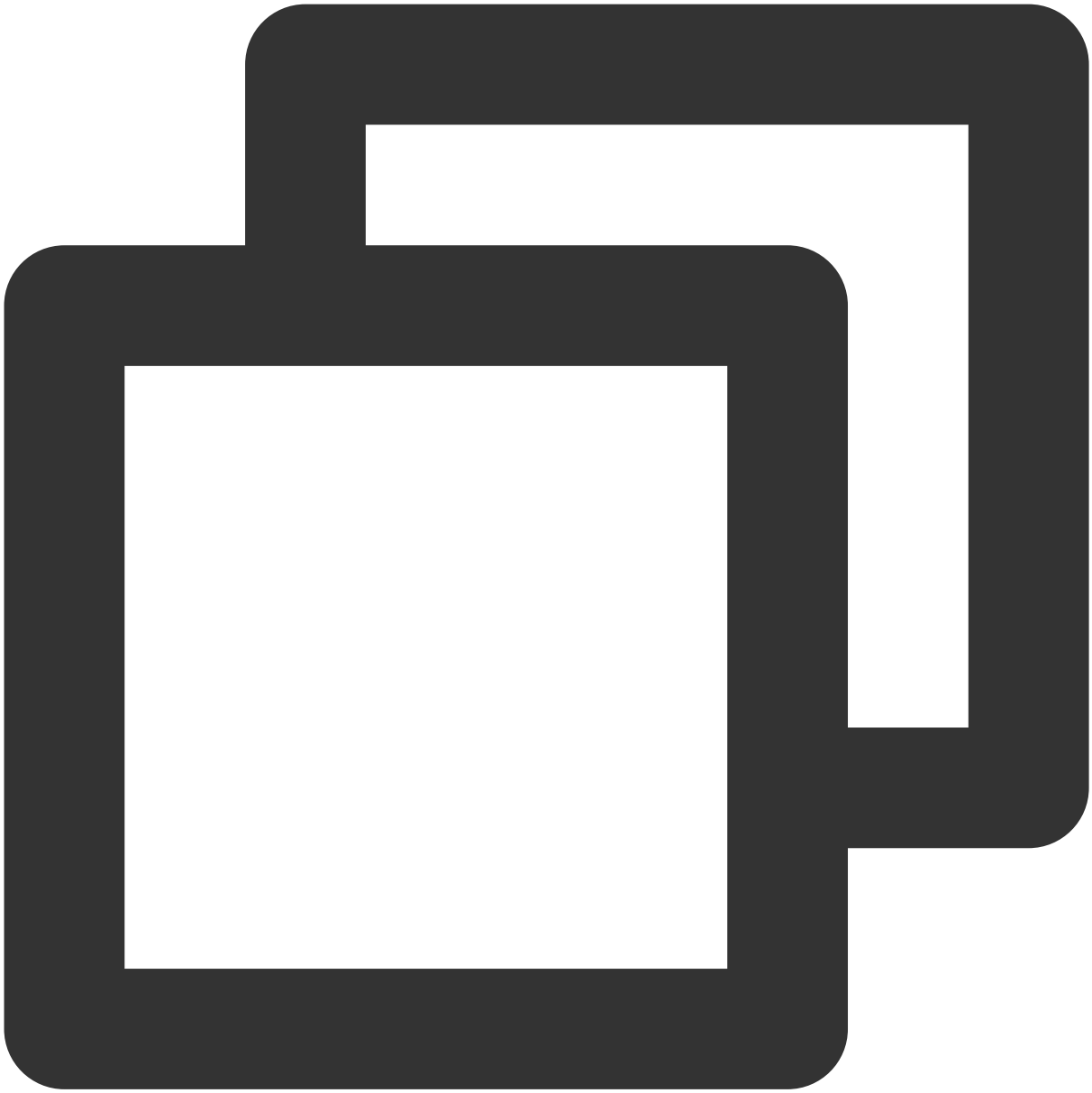
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
versioning	版本控制请求操作参数	Struct
status	版本是否开启，枚举值：Suspended，Enabled	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

开启版本控制



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_versioning()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

   //初始化请求选项
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //put bucket versioning
    cos_versioning_content_t *versioning = NULL;
    versioning = cos_create_versioning_content(p);
    cos_str_set(&versioning->status, "Enabled");
```

```
    s = cos_put_bucket_versioning(options, &bucket, versioning, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("put bucket versioning succeeded\\n");
    } else {
        printf("put bucket versioning failed\\n");
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

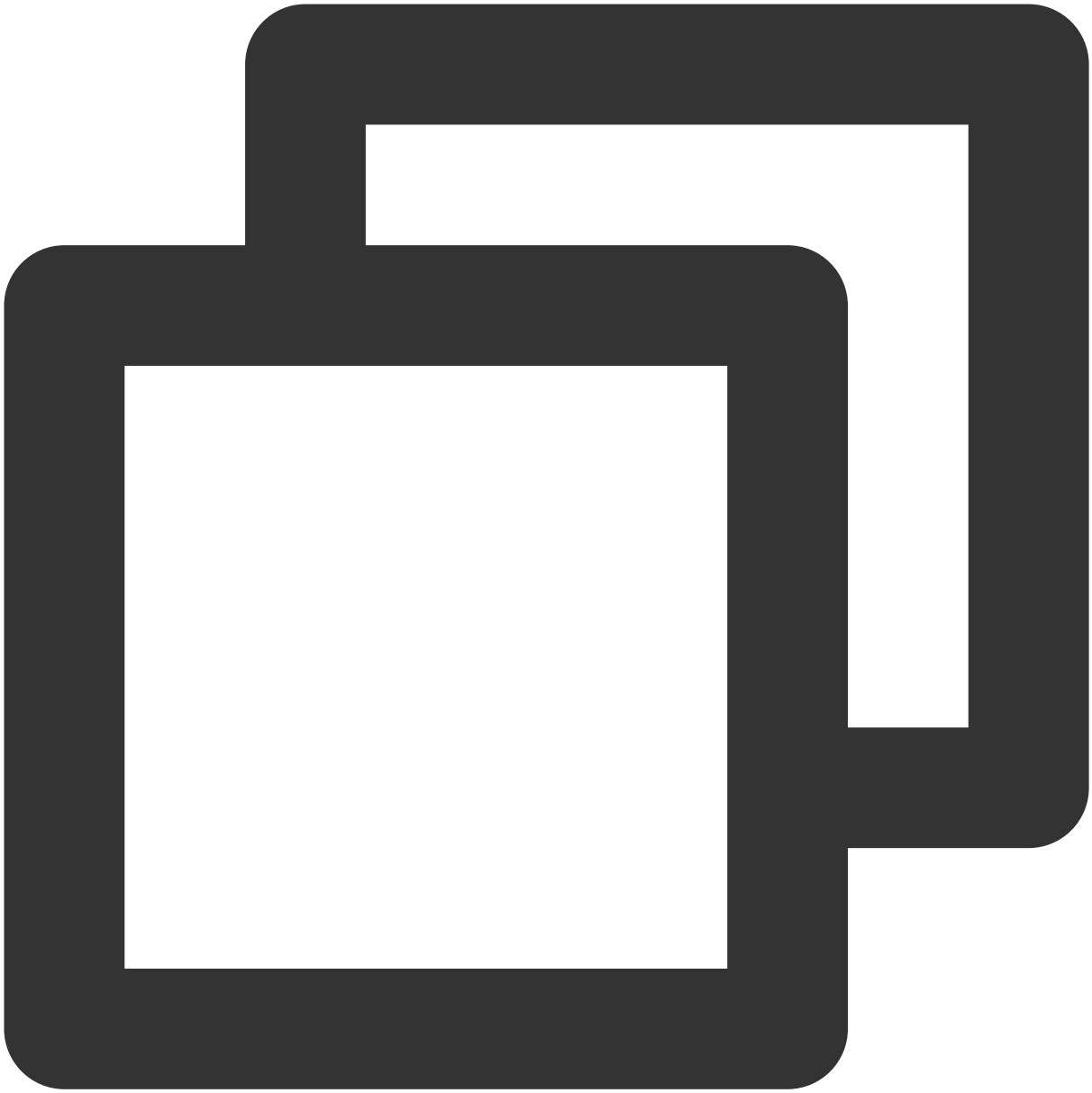
    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_versioning();

    cos_http_io_deinitialize();

    return 0;
}
```

暂停版本控制



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_versioning()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

   //初始化请求选项
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //put bucket versioning
    cos_versioning_content_t *versioning = NULL;
    versioning = cos_create_versioning_content(p);
    cos_str_set(&versioning->status, "Suspended");
```

```
    s = cos_put_bucket_versioning(options, &bucket, versioning, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("put bucket versioning succeeded\\n");
    } else {
        printf("put bucket versioning failed\\n");
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_versioning();

    cos_http_io_deinitialize();

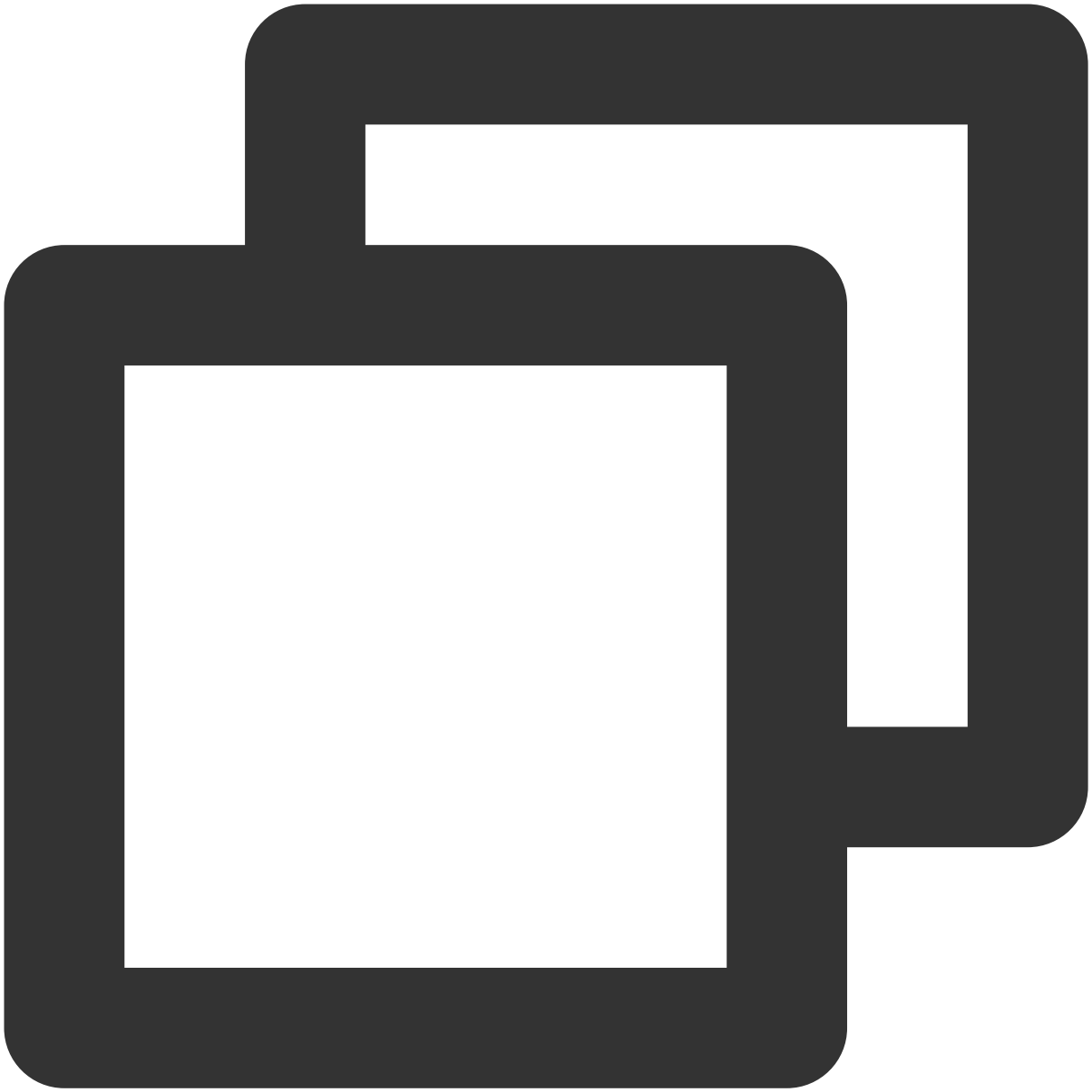
    return 0;
}
```

查询版本控制

功能说明

查询指定存储桶的版本控制信息。

方法原型



```
cos_status_t *cos_get_bucket_versioning(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        cos_versioning_content_t *versioning,
                                        cos_table_t **resp_headers);
```

参数说明

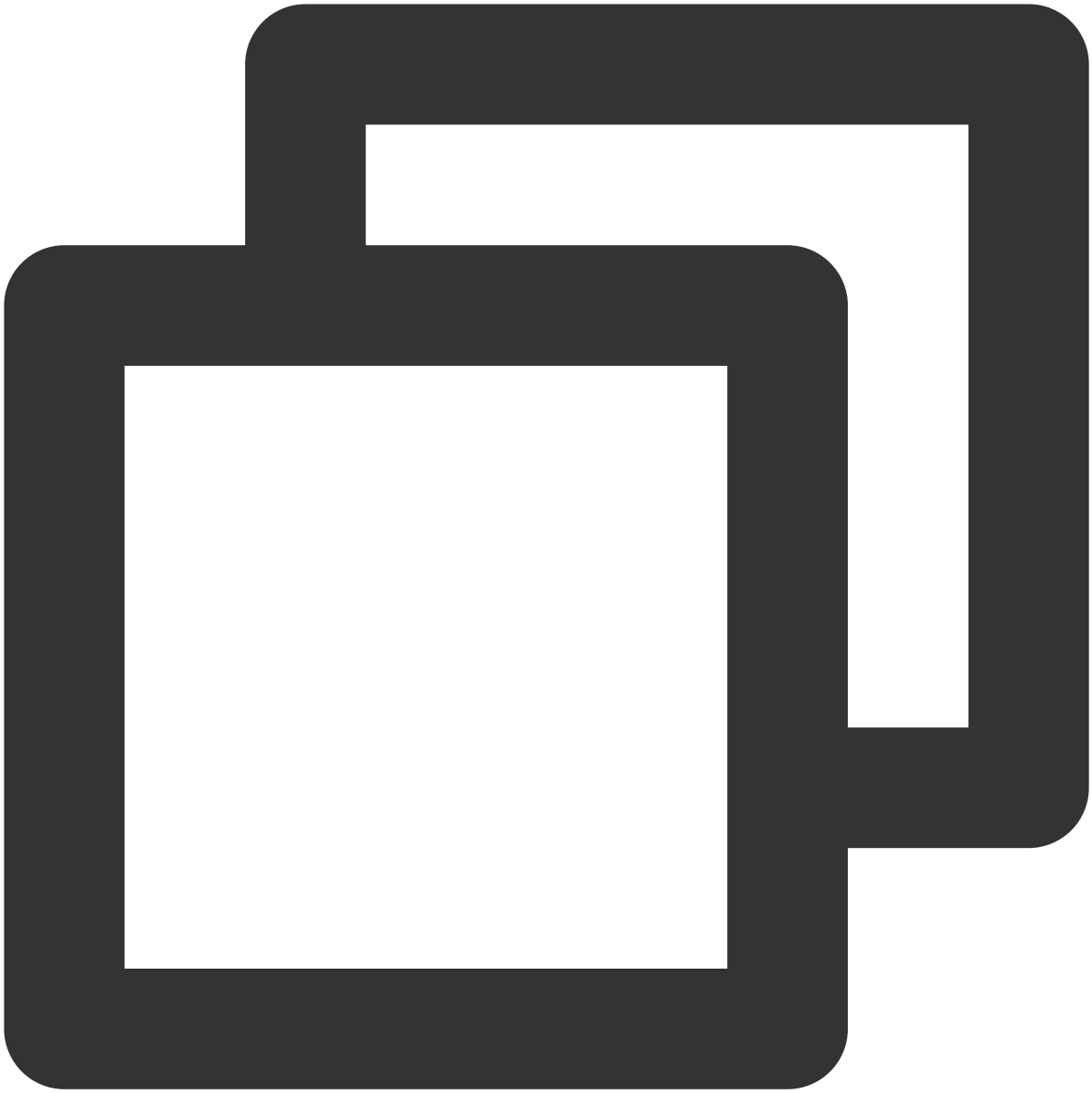
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
versioning	版本控制请求操作参数	Struct
status	版本是否开启，枚举值：Suspended，Enabled	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_get_versioning()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

   //初始化请求选项
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //get bucket versioning
    cos_versioning_content_t *versioning = NULL;
    versioning = cos_create_versioning_content(p);
    s = cos_get_bucket_versioning(options, &bucket, versioning, &resp_headers);
```

```
    if (cos_status_is_ok(s)) {
        printf("put bucket versioning succeeded\\n");
        printf("bucket versioning status: %s\\n", versioning->status.data);
    } else {
        printf("put bucket versioning failed\\n");
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_get_versioning();

    cos_http_io_deinitialize();

    return 0;
}
```

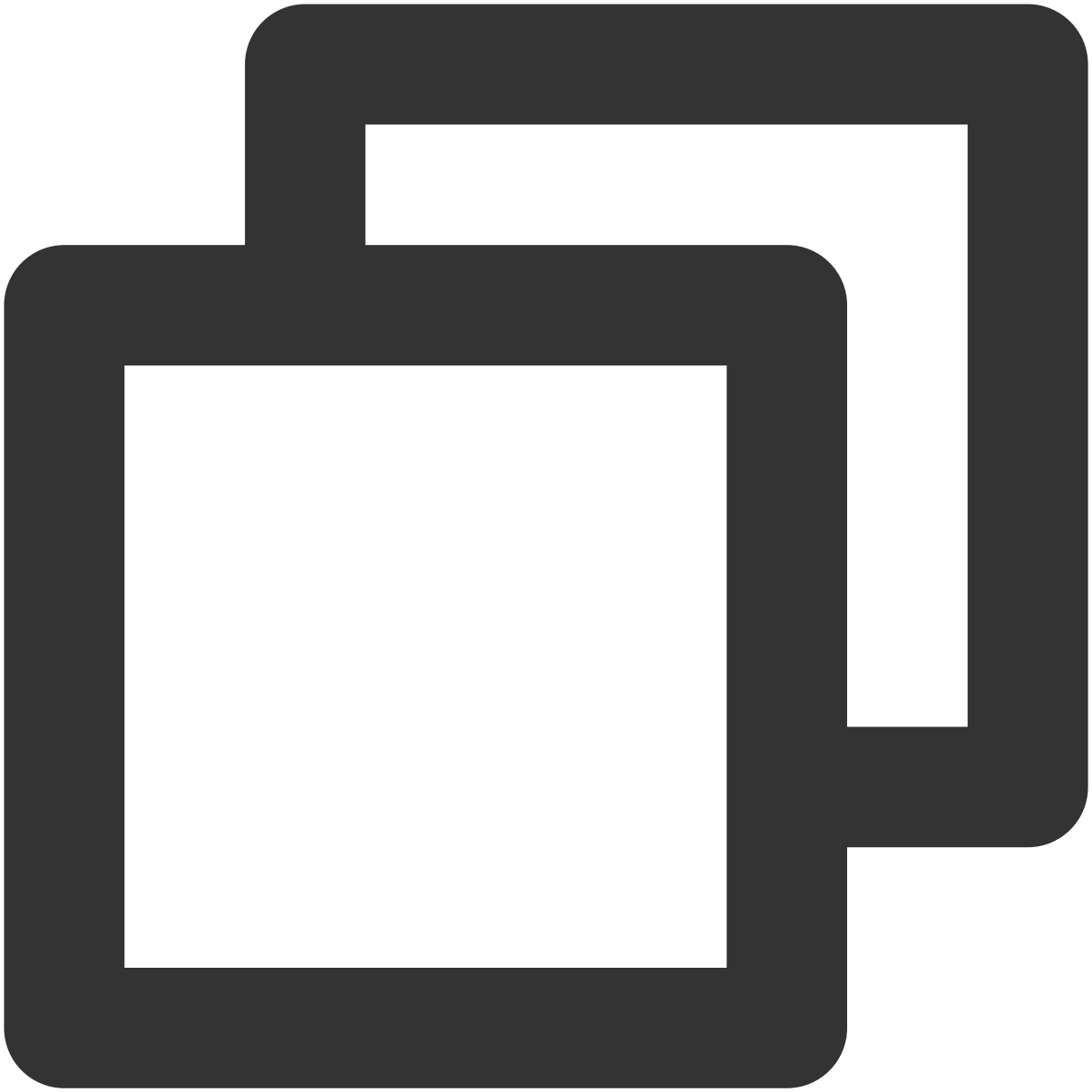
跨地域复制

设置跨地域复制

功能说明

设置指定存储桶的跨地域复制规则。

方法原型



```
cos_status_t *cos_put_bucket_replication(const cos_request_options_t *options,
                                         const cos_string_t *bucket,
                                         cos_replication_params_t *replication_para
                                         cos_table_t **resp_headers);
```

参数说明

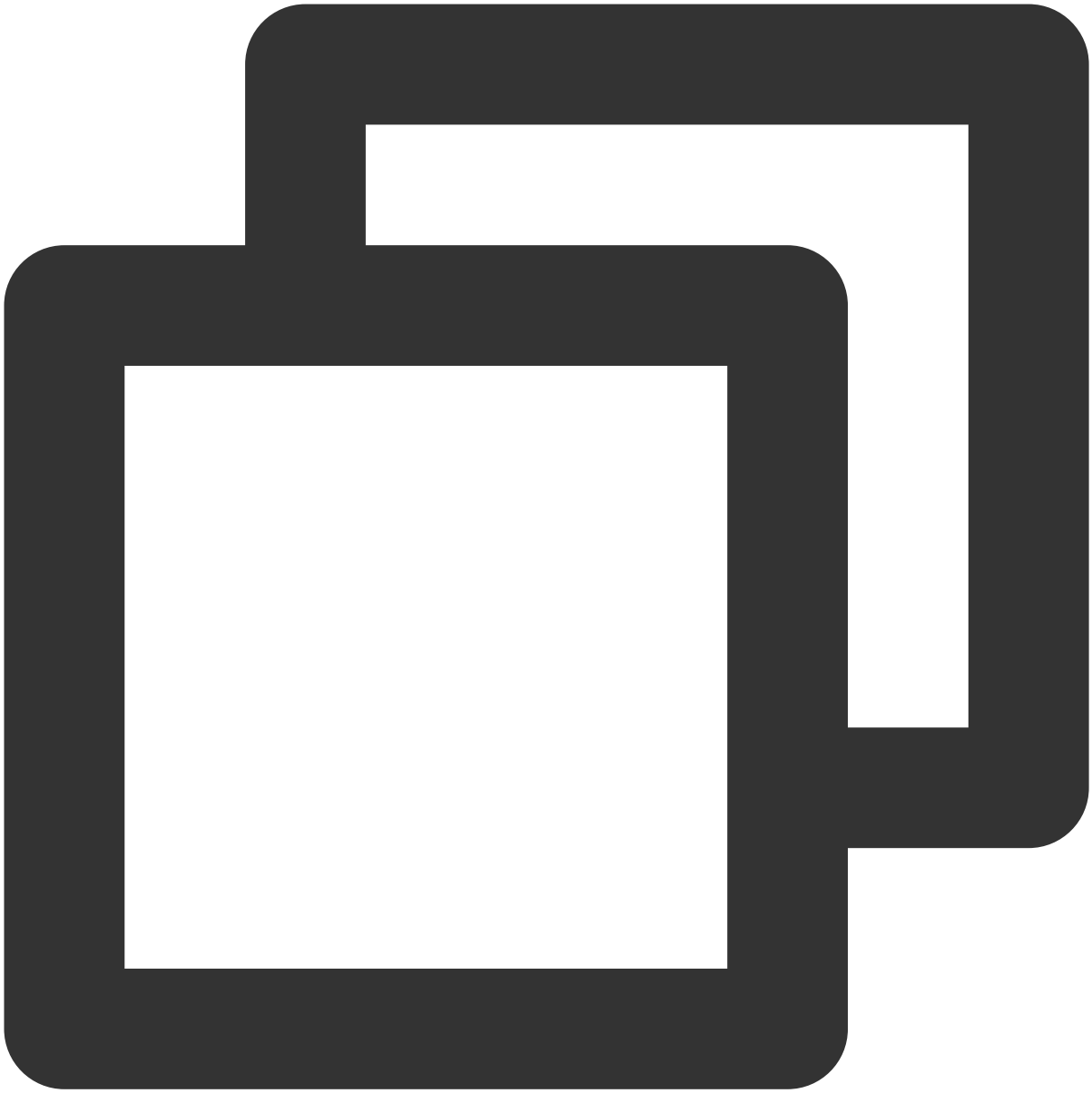
参数名称	参数描述	类型
options	COS 请求选项。	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
replication_param	跨地域复制请求操作参数	Struct
role	操作者账户信息	String
rule_list	跨地域复制规则配置信息	Struct
id	用来标注具体规则的名称	String
status	标识规则是否生效，枚举值：Enabled，Disabled	String
prefix	匹配前缀。不可重叠，重叠返回错误	String
dst_bucket	目的存储桶标识，格式为：资源标识符 `qcs::cos:[region]::[bucketname-AppId]`	String
storage_class	存储类别，枚举值：Standard，Standard_IA 默认值原存储桶级别	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_replication()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_request_options_t *dst_options = NULL;
    cos_string_t bucket;
    cos_string_t dst_bucket;
    cos_table_t *resp_headers = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&dst_bucket, "replicationtest");

    dst_options = cos_request_options_create(p);
    init_test_request_options(dst_options, is_cname);
```

```
    cos_str_set(&dst_options->config->endpoint, "cn-east.myqcloud.com");

    cos_replication_params_t *replication_param = NULL;
    replication_param = cos_create_replication_params(p);
    cos_str_set(&replication_param->role, "qcs::cam::uin/100000616666:uin/100000616

    cos_replication_rule_content_t *rule = NULL;
    rule = cos_create_replication_rule_content(p);
    cos_str_set(&rule->id, "Rule_01");
    cos_str_set(&rule->status, "Enabled");
    cos_str_set(&rule->prefix, "test1");
    cos_str_set(&rule->dst_bucket, "qcs:id/0:cos:cn-east:appid/1253686666:replicati
    cos_list_add_tail(&rule->node, &replication_param->rule_list);

    rule = cos_create_replication_rule_content(p);
    cos_str_set(&rule->id, "Rule_02");
    cos_str_set(&rule->status, "Disabled");
    cos_str_set(&rule->prefix, "test2");
    cos_str_set(&rule->storage_class, "Standard_IA");
    cos_str_set(&rule->dst_bucket, "qcs:id/0:cos:cn-east:appid/1253686666:replicati
    cos_list_add_tail(&rule->node, &replication_param->rule_list);

    rule = cos_create_replication_rule_content(p);
    cos_str_set(&rule->id, "Rule_03");
    cos_str_set(&rule->status, "Enabled");
    cos_str_set(&rule->prefix, "test3");
    cos_str_set(&rule->storage_class, "Standard_IA");
    cos_str_set(&rule->dst_bucket, "qcs:id/0:cos:cn-east:appid/1253686666:replicati
    cos_list_add_tail(&rule->node, &replication_param->rule_list);

    //put bucket replication
    s = cos_put_bucket_replication(options, &bucket, replication_param, &resp_heade
    log_status(s);

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }
```

```
    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_replication();

    cos_http_io_deinitialize();

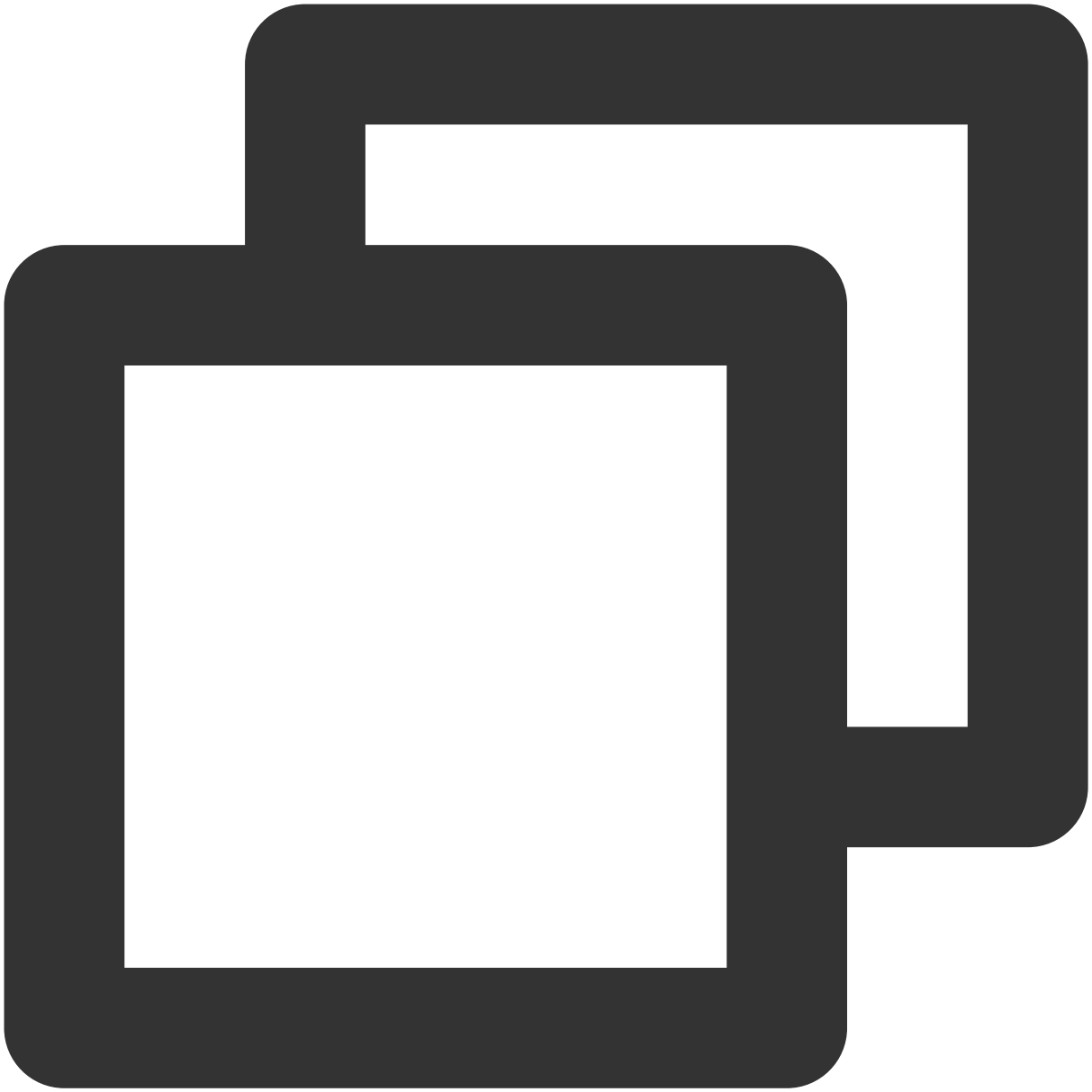
    return 0;
}
```

查询跨地域复制

功能说明

查询指定存储桶的跨地域复制规则。

方法原型



```
cos_status_t *cos_get_bucket_replication(const cos_request_options_t *options,
                                         const cos_string_t *bucket,
                                         cos_replication_params_t *replication_para
                                         cos_table_t **resp_headers);
```

参数说明

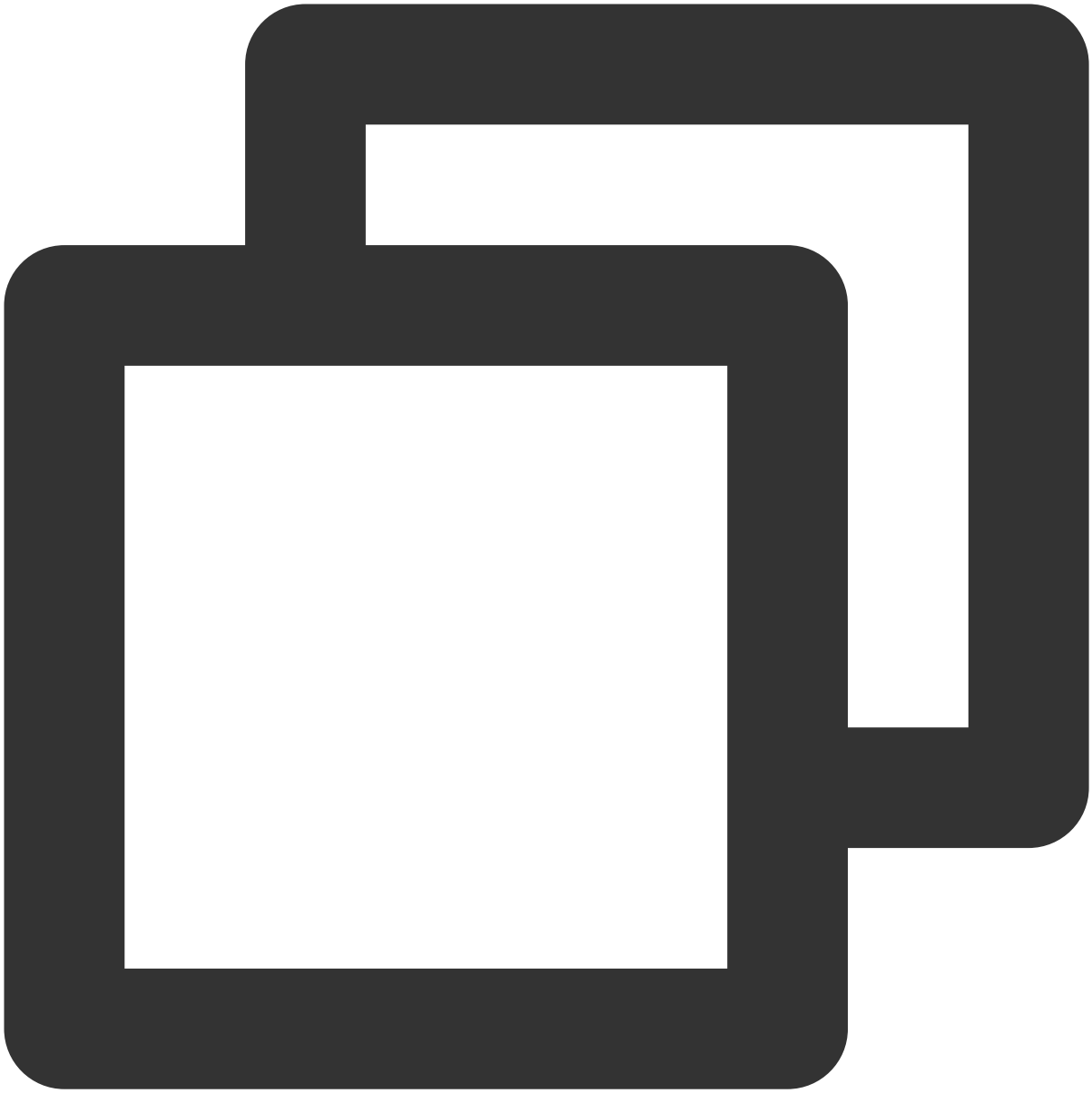
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
replication_param	跨地域复制请求操作参数	Struct
role	操作者账户信息	String
rule_list	跨地域复制规则配置信息	Struct
id	用来标注具体规则的名称	String
status	标识规则是否生效，枚举值：Enabled，Disabled	String
prefix	匹配前缀。不可重叠，重叠返回错误	String
dst_bucket	目的存储桶标识，格式为：资源标识符 `qcs::cos:[region]::[bucketname-AppId]`	String
storage_class	存储类别，枚举值：Standard，Standard_IA 默认值原存储桶级别	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_get_replication()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

    //初始化请求选项
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //查询跨地域复制规则
    cos_replication_params_t *replication_param2 = NULL;
    replication_param2 = cos_create_replication_params(p);
    s = cos_get_bucket_replication(options, &bucket, replication_param2, &resp_head
```

```
    if (cos_status_is_ok(s)) {
        printf("get bucket replication succeeded\\n");
        printf("ReplicationConfiguration role: %s\\n", replication_param2->role.dat
        cos_replication_rule_content_t *content = NULL;
        cos_list_for_each_entry(cos_replication_rule_content_t, content, &replicati
            printf("ReplicationConfiguration rule, id:%s, status:%s, prefix:%s, dst
                           content->id.data, content->status.data, content->prefix
        }
    } else {
        printf("get bucket replication failed\\n");
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_get_replication();

    cos_http_io_deinitialize();

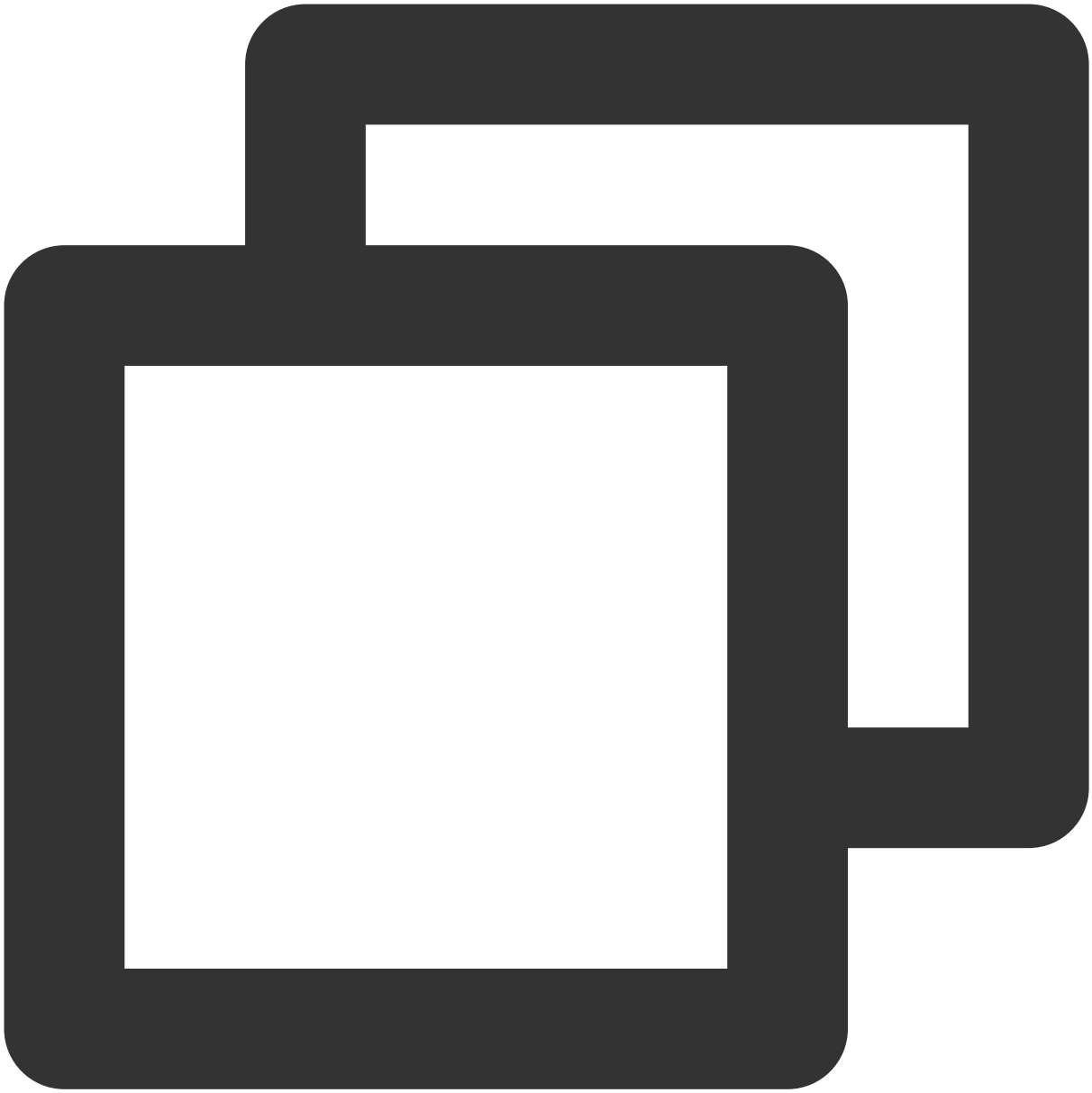
    return 0;
}
```

删除跨地域复制

功能说明

删除指定存储桶的跨地域复制规则。

方法原型



```
cos_status_t *cos_delete_bucket_replication(const cos_request_options_t *options,
                                            const cos_string_t *bucket,
                                            cos_table_t **resp_headers);
```

参数说明

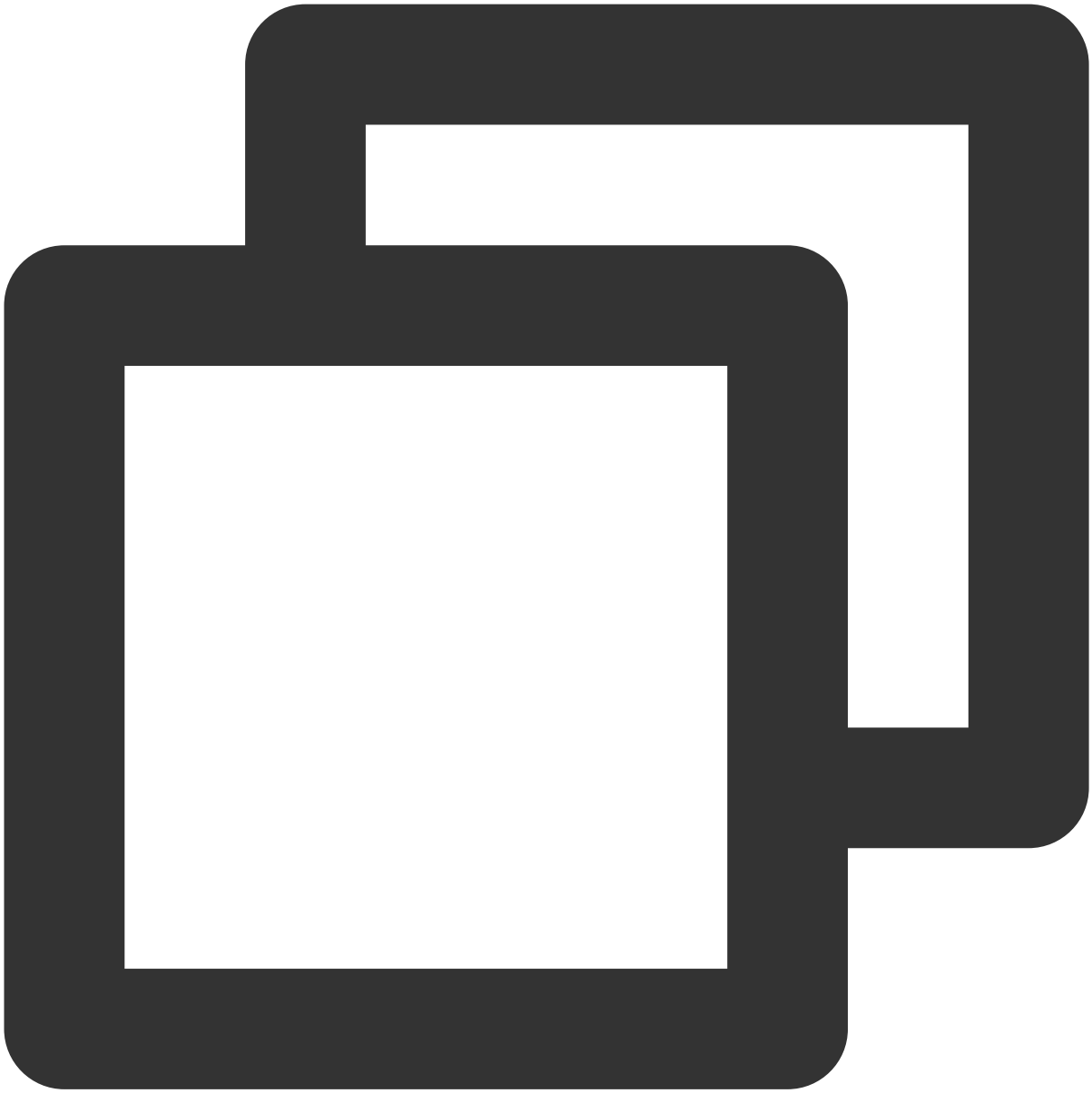
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_replication()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

    //初始化请求选项
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //删除跨地域复制规则
    s = cos_delete_bucket_replication(options, &bucket, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("delete bucket replication succeeded\\n");
```

```
    } else {
        printf("delete bucket replication failed\\n");
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_delete_replication();

    cos_http_io_deinitialize();

    return 0;
```

数据管理
日志管理

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于日志管理的 API 概览以及 C SDK 示例代码。

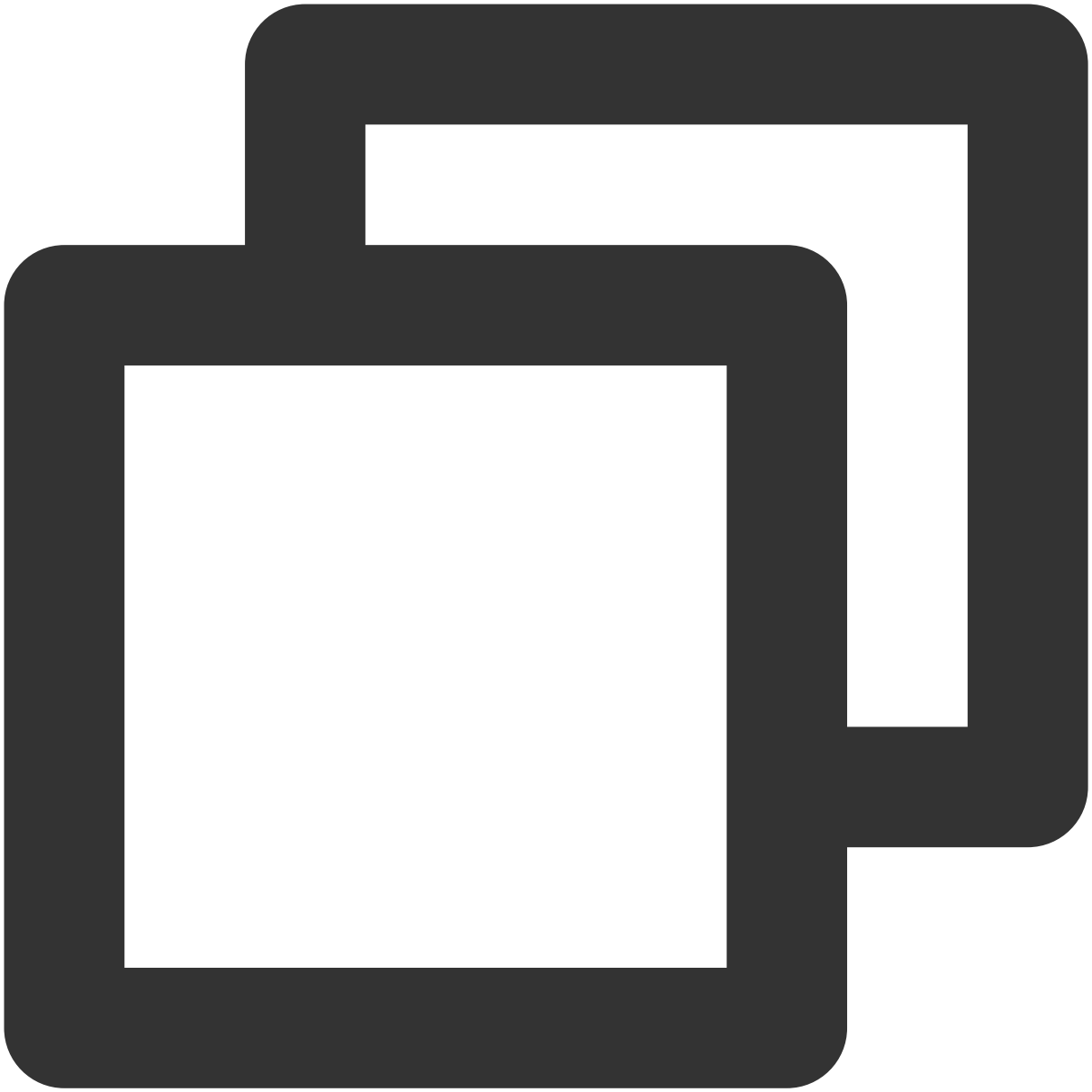
API	操作名	操作描述
PUT Bucket logging	设置日志管理	为源存储桶开启日志记录
GET Bucket logging	查询日志管理	查询源存储桶的日志配置信息

设置日志管理

功能说明

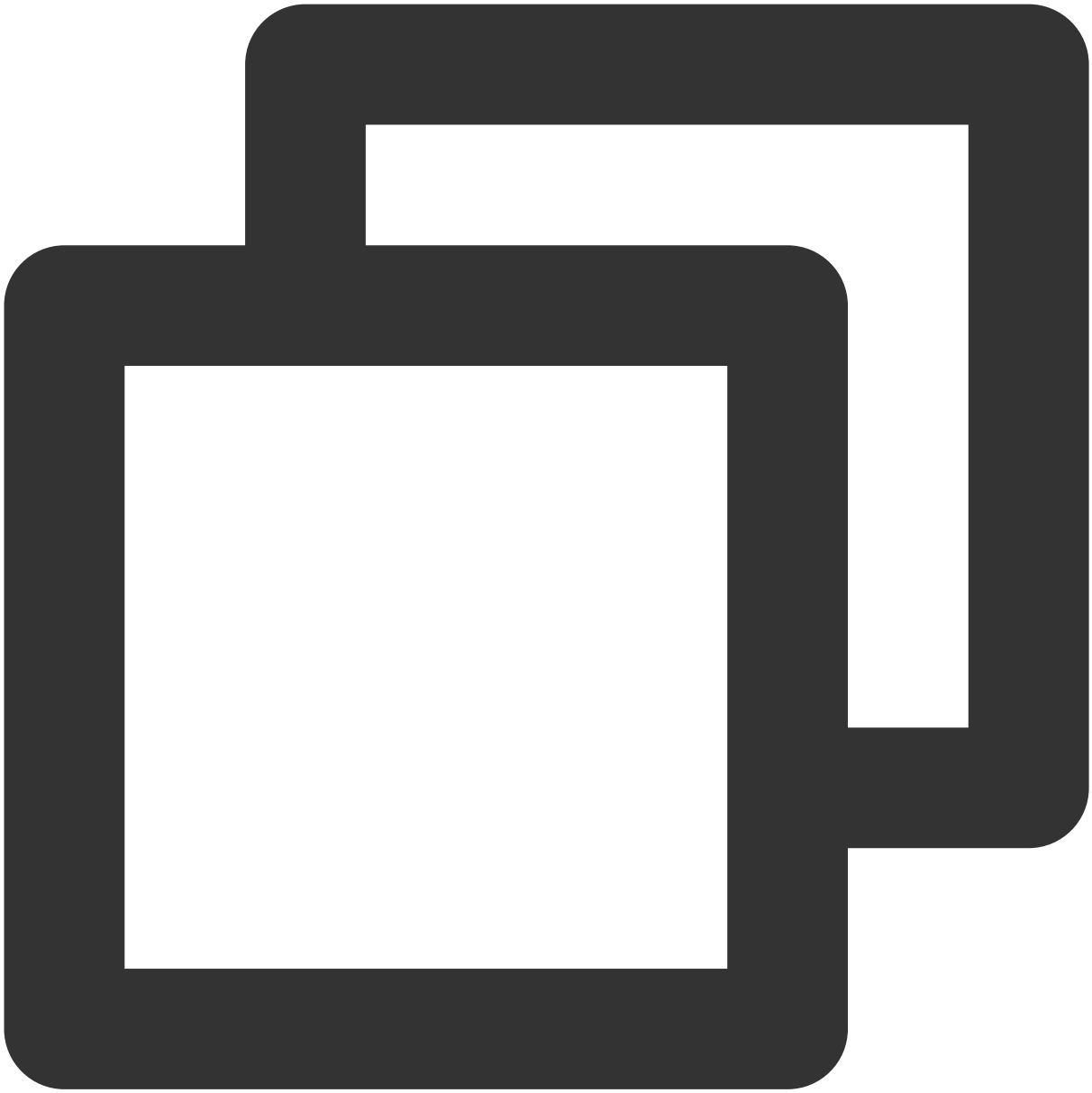
PUT Bucket logging 用于为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中。

方法原型



```
cos_status_t *cos_put_bucket_logging(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     cos_logging_params_t *logging_params,
                                     cos_table_t **resp_headers);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_logging()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_logging_params_t  *params = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //创建logging参数
```

```
    params = cos_create_logging_params(options->pool);
    cos_str_set(&params->target_bucket, TEST_BUCKET_NAME);
    cos_str_set(&params->target_prefix, "logging/");

    status = cos_put_bucket_logging(options, &bucket, params, &resp_headers);
    log_status(status);

    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_logging();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	开启日志功能的源存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
logging_params	存储桶日志配置信息	Struct
target_bucket	存放日志的目标存储桶，可以是同一个存储桶（但不推荐），或同一账户下、同一地域的存储桶	String
target_prefix	日志存放在目标存储桶的指定路径	String

版权所有：腾讯云计算（北京）有限责任公司

resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

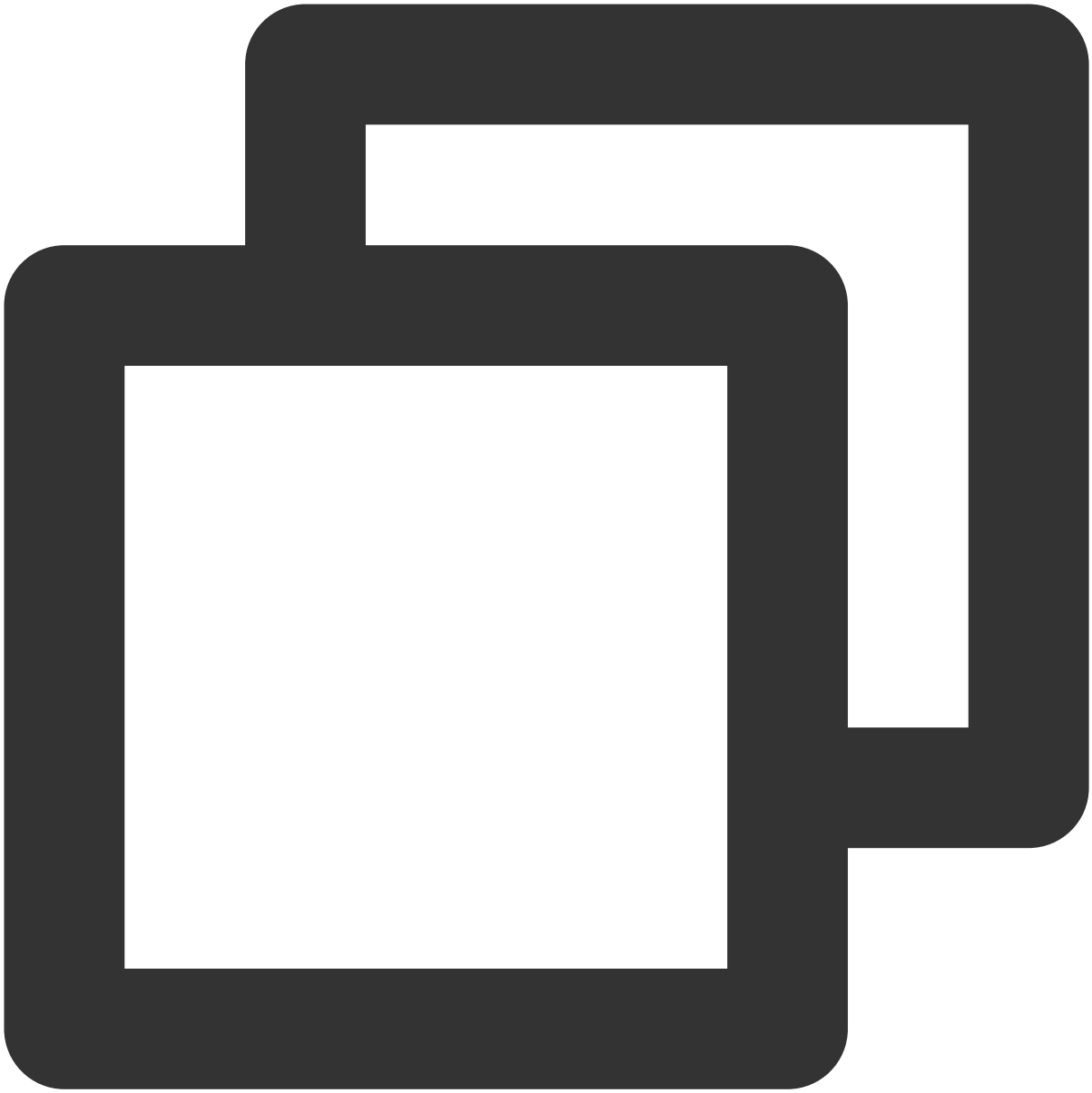
返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

查询日志管理

功能说明

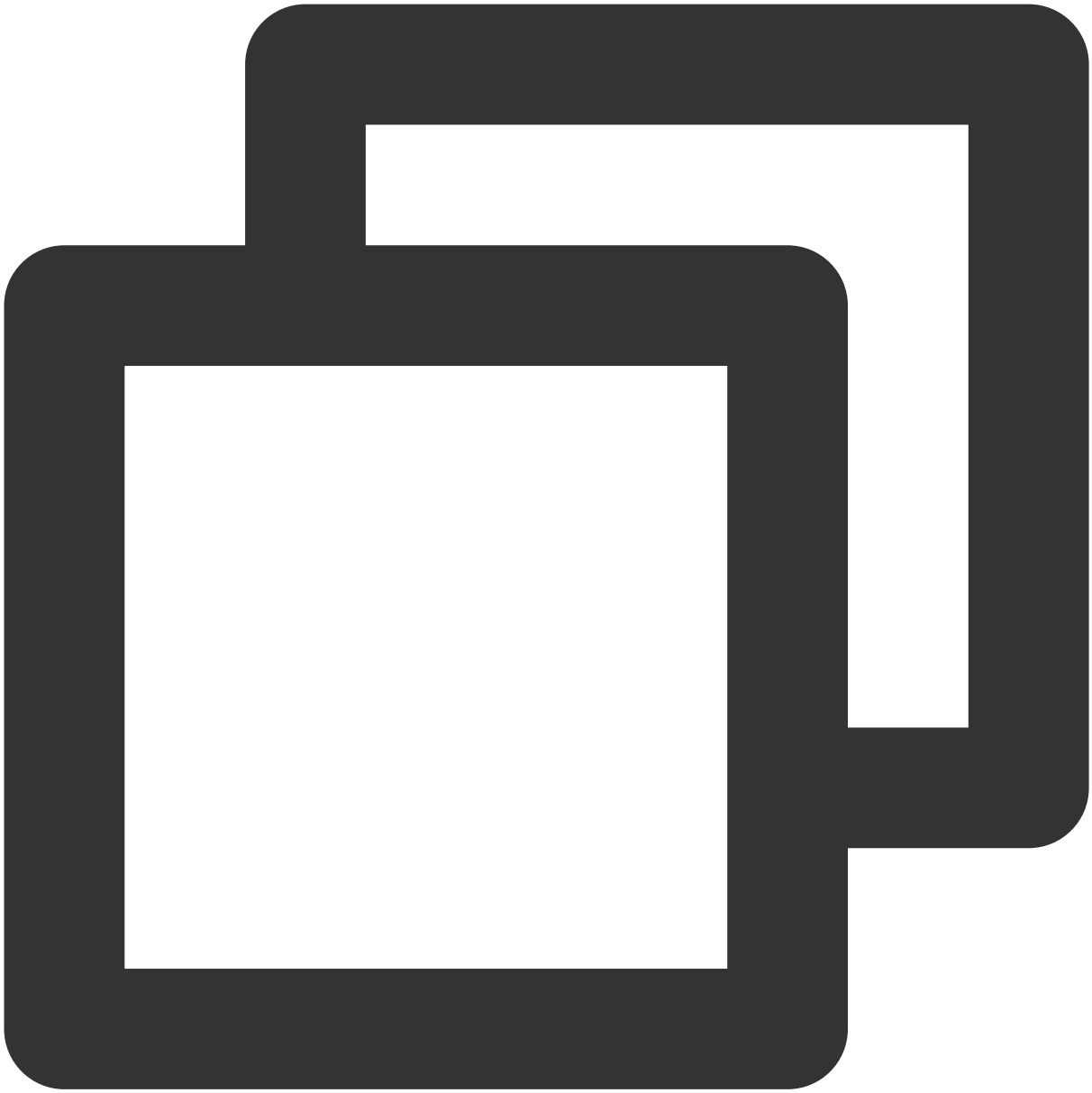
GET Bucket logging 用于查询指定存储桶的日志配置信息。

方法原型



```
cos_status_t *cos_get_bucket_logging(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     cos_logging_params_t *logging_params,
                                     cos_table_t **resp_headers);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_get_logging()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_logging_params_t  *result = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    result = cos_create_logging_params(options->pool);
```

```
    status = cos_get_bucket_logging(options, &bucket, result, &resp_headers);
    log_status(status);
    if (!cos_status_is_ok(status)) {
        cos_pool_destroy(pool);
        return;
    }

    //查看结果
    char *line = NULL;
    line = apr_psprintf(options->pool, "%.*s\\n", result->target_bucket.len, result
    printf("target bucket: %s", line);
    line = apr_psprintf(options->pool, "%.*s\\n", result->target_prefix.len, result
    printf("target prefix: %s", line);

    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_get_logging();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	存放日志的目标存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String

logging_params	存储桶日志配置信息	Struct
target_bucket	存放日志的目标存储桶，可以是同一个存储桶（但不推荐），或同一账户下、同一地域的存储桶	String
target_prefix	日志存放在目标存储桶的指定路径	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

存储桶标签

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket tagging	设置存储桶标签	为已存在的存储桶设置标签
GET Bucket tagging	查询存储桶标签	查询指定存储桶下已有的存储桶标签
DELETE Bucket tagging	删除存储桶标签	删除指定的存储桶标签

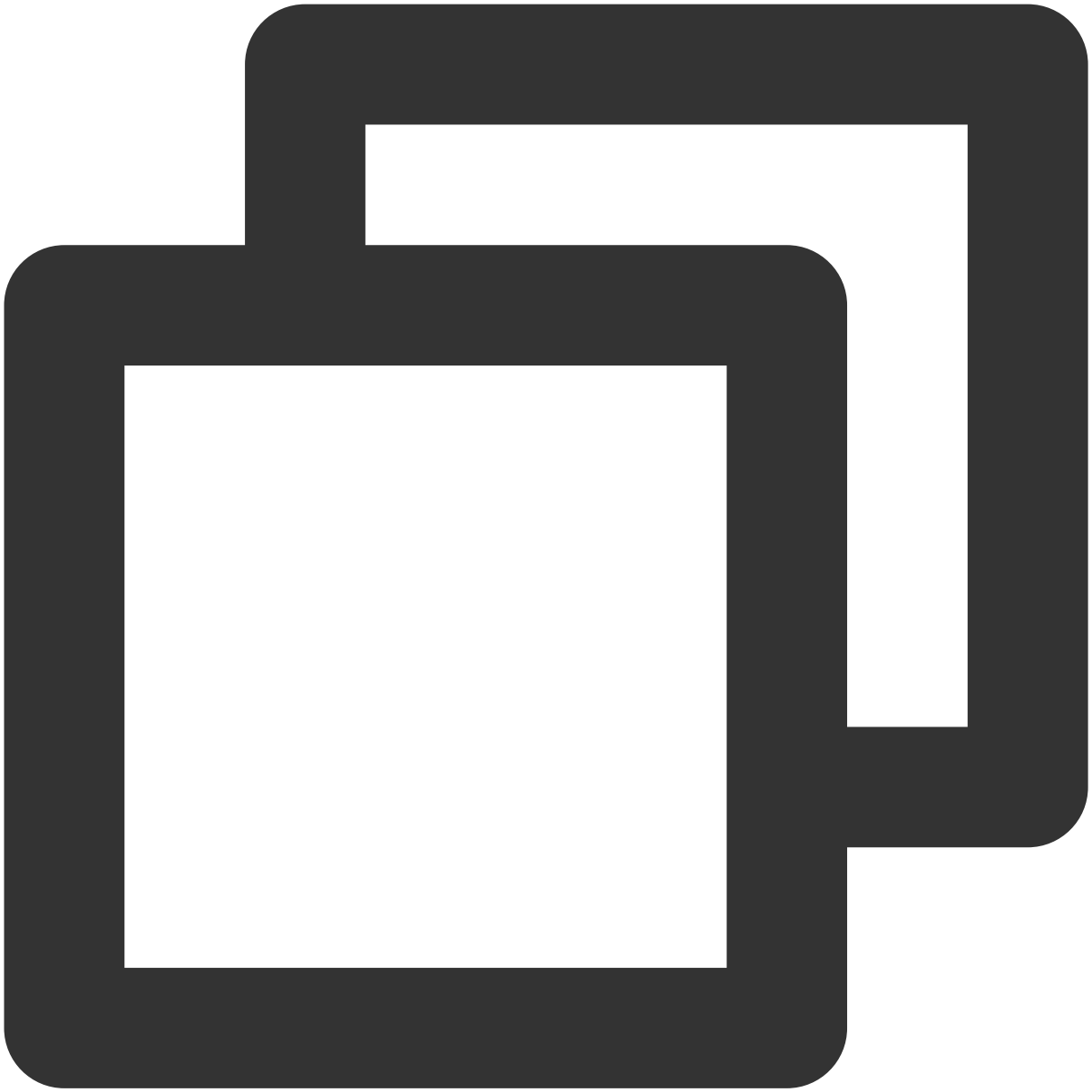
设置存储桶标签

功能说明

PUT Bucket tagging 用于为已存在的存储桶设置标签。

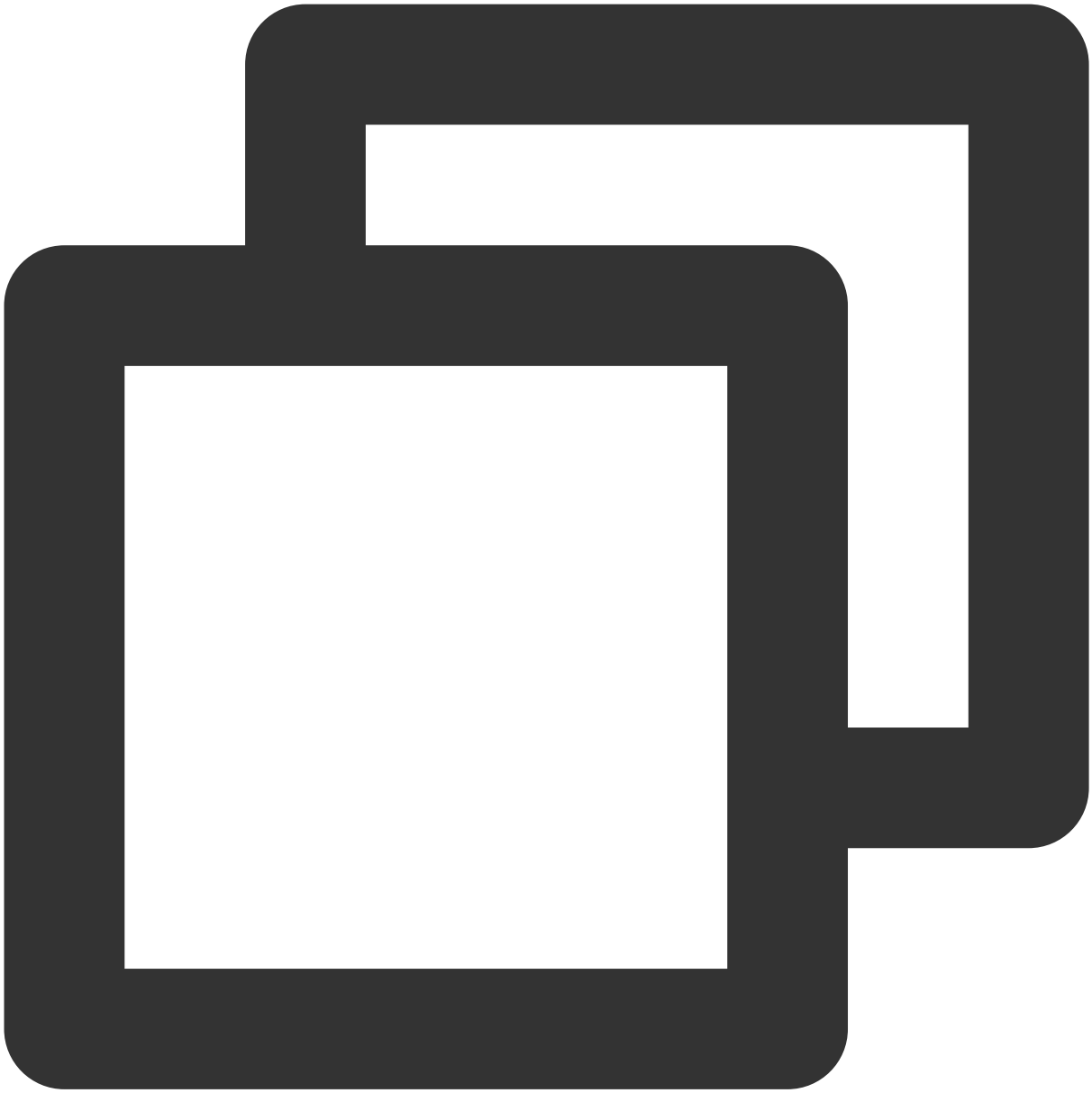
方法原型

版权所有：腾讯云计算（北京）有限责任公司



```
cos_status_t *cos_put_bucket_tagging(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     cos_tagging_params_t *tagging_params,
                                     cos_table_t **resp_headers);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_bucket_tagging()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;
    cos_tagging_params_t *params = NULL;
    cos_tagging_tag_t *tag = NULL;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
```

```
    // put tagging
    params = cos_create_tagging_params(pool);
    tag = cos_create_tagging_tag(pool);
    cos_str_set(&tag->key, "age");
    cos_str_set(&tag->value, "18");
    cos_list_add_tail(&tag->node, &params->node);

    tag = cos_create_tagging_tag(pool);
    cos_str_set(&tag->key, "name");
    cos_str_set(&tag->value, "xiaoming");
    cos_list_add_tail(&tag->node, &params->node);

    status = cos_put_bucket_tagging(options, &bucket, params, &resp_headers);
    log_status(status);

    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_bucket_tagging();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct

bucket	设置标签的存储桶，格式为 BucketName-APPID ， 详情请参见 命名规范	String
tagging_params	存储桶标签配置信息	Struct
key	标签的 Key，长度不超过128字节, 支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String
value	标签的 Value，长度不超过256字节, 支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

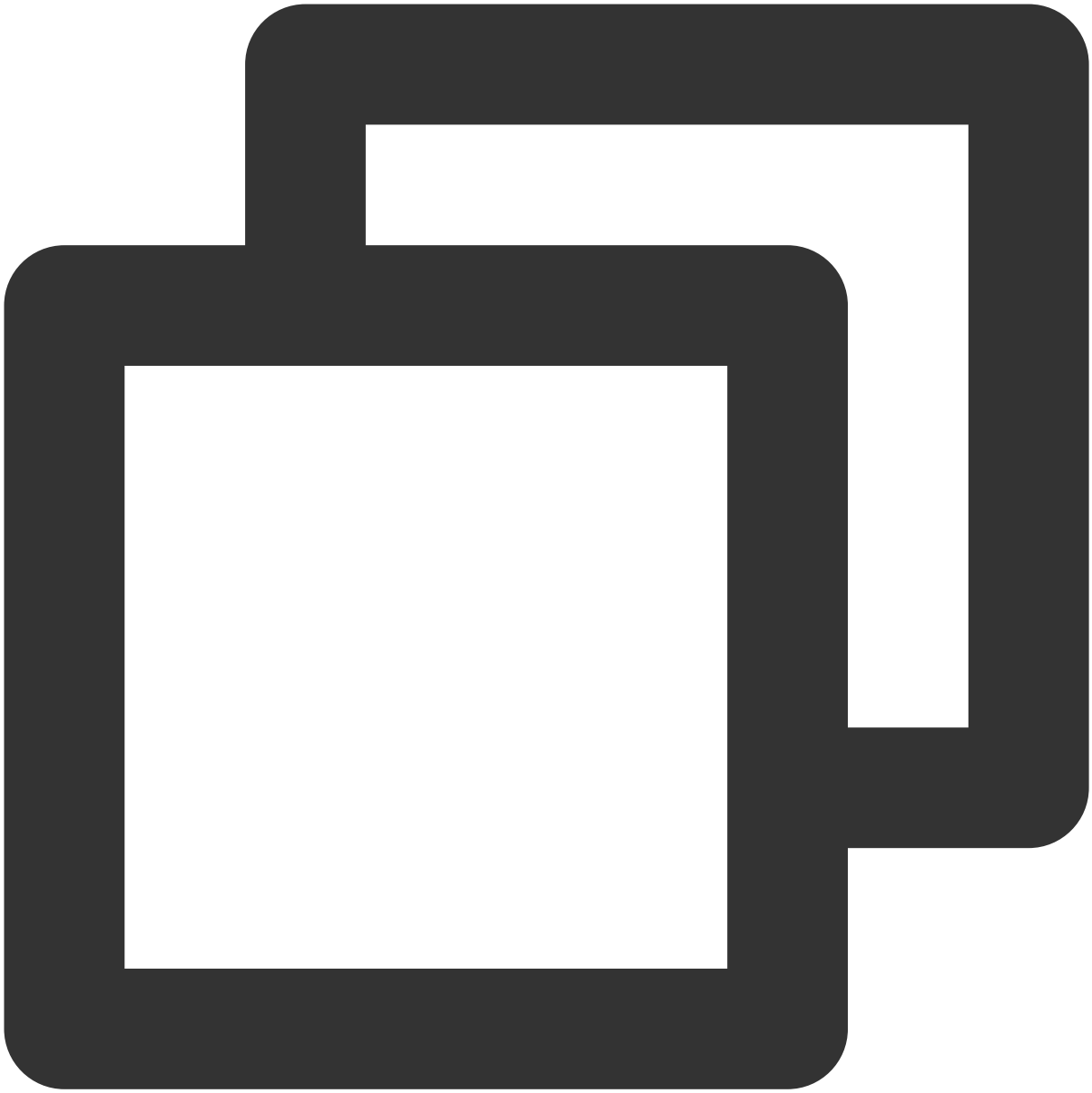
返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

查询存储桶标签

功能说明

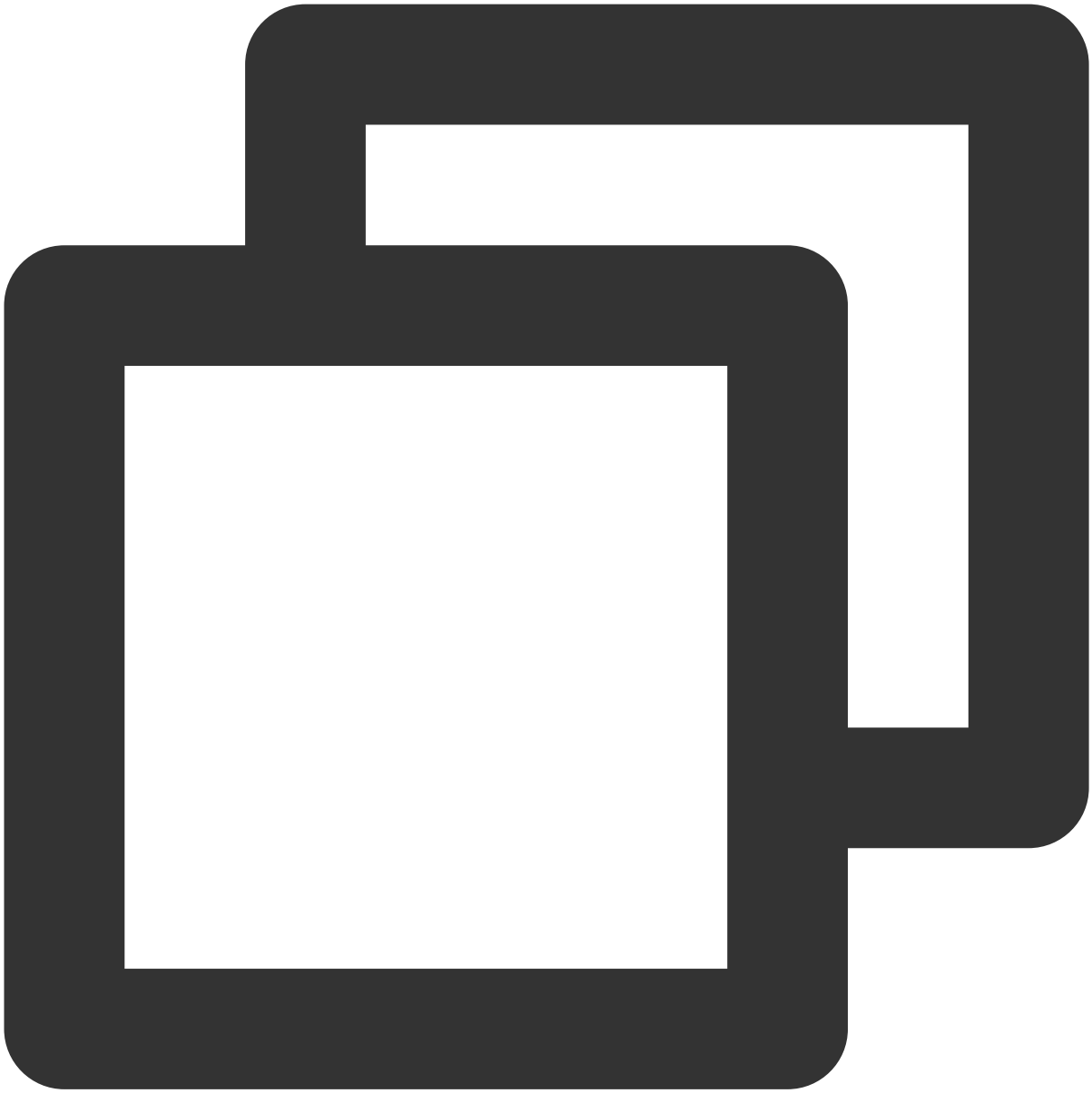
GET Bucket tagging 用于查询指定存储桶下已有的存储桶标签。

方法原型



```
cos_status_t *cos_get_bucket_tagging(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     cos_tagging_params_t *tagging_params,
                                     cos_table_t **resp_headers);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_get_bucket_tagging()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;
    cos_tagging_params_t *result = NULL;
    cos_tagging_tag_t *tag = NULL;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
```

```
    // get tagging
    result = cos_create_tagging_params(pool);
    status = cos_get_bucket_tagging(options, &bucket, result, &resp_headers);
    log_status(status);

    tag = NULL;
    cos_list_for_each_entry(cos_tagging_tag_t, tag, &result->node, node) {
        printf("taging key: %s\\n", tag->key.data);
        printf("taging value: %s\\n", tag->value.data);

    }

    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_get_bucket_tagging();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	查询标签的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
tagging_params	存储桶标签配置信息	Struct

版权所有：腾讯云计算（北京）有限责任公司

key	标签的 Key，长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String
value	标签的 Value，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

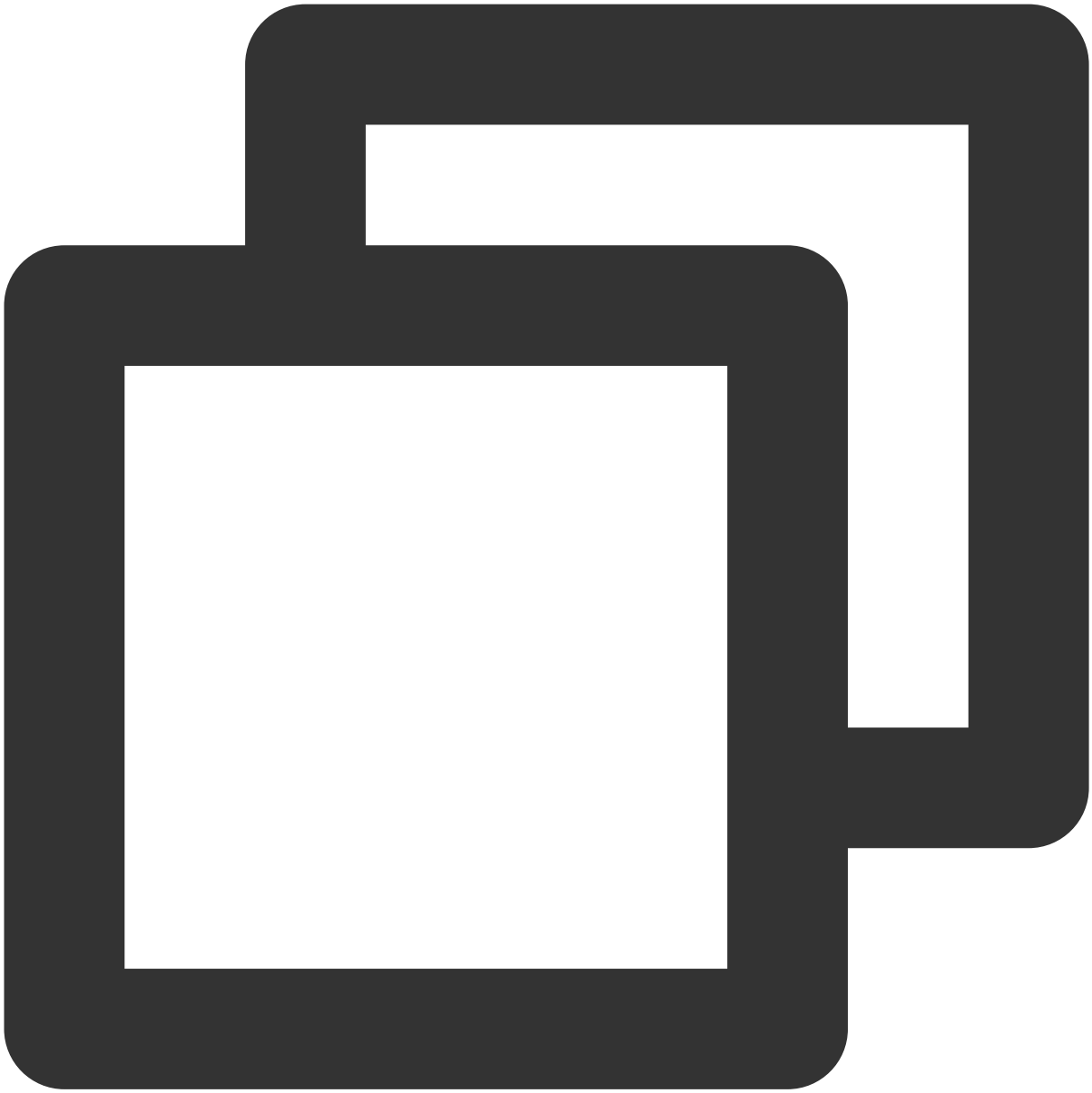
返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

删除存储桶标签

功能说明

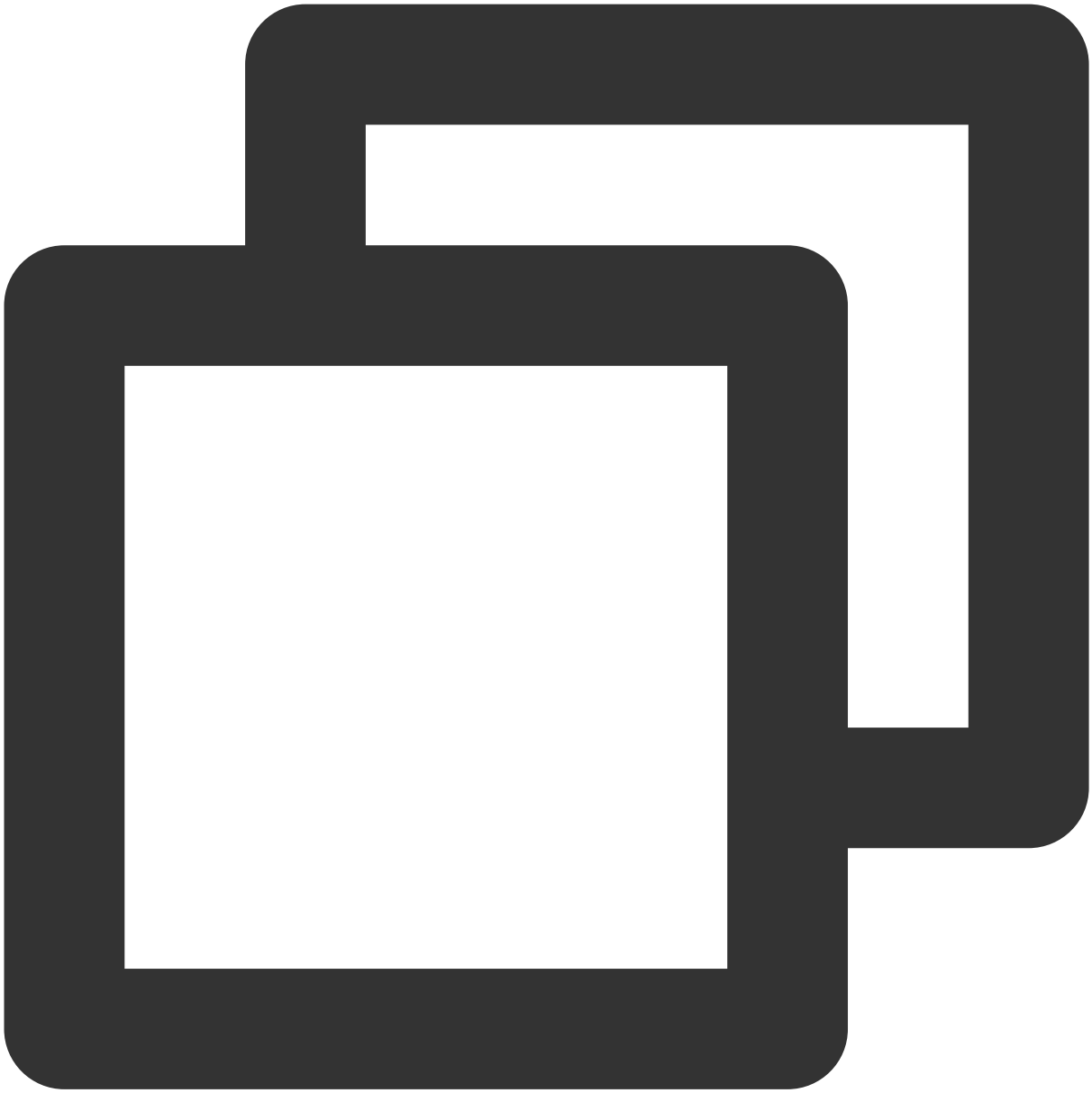
DELETE Bucket tagging 用于删除指定存储桶下已有的存储桶标签。

方法原型



```
cos_status_t *cos_delete_bucket_tagging(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        cos_table_t **resp_headers);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_bucket_tagging()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    // delete tagging
    status = cos_delete_bucket_tagging(options, &bucket, &resp_headers);
```

```
    log_status(status);

    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_delete_bucket_tagging();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	被删除标签的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String

req_id	请求消息 ID	String

静态网站

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket website	设置静态网站	设置存储桶的静态网站配置
GET Bucket website	查询静态网站配置	查询存储桶的静态网站配置
DELETE Bucket website	删除静态网站配置	删除存储桶的静态网站配置

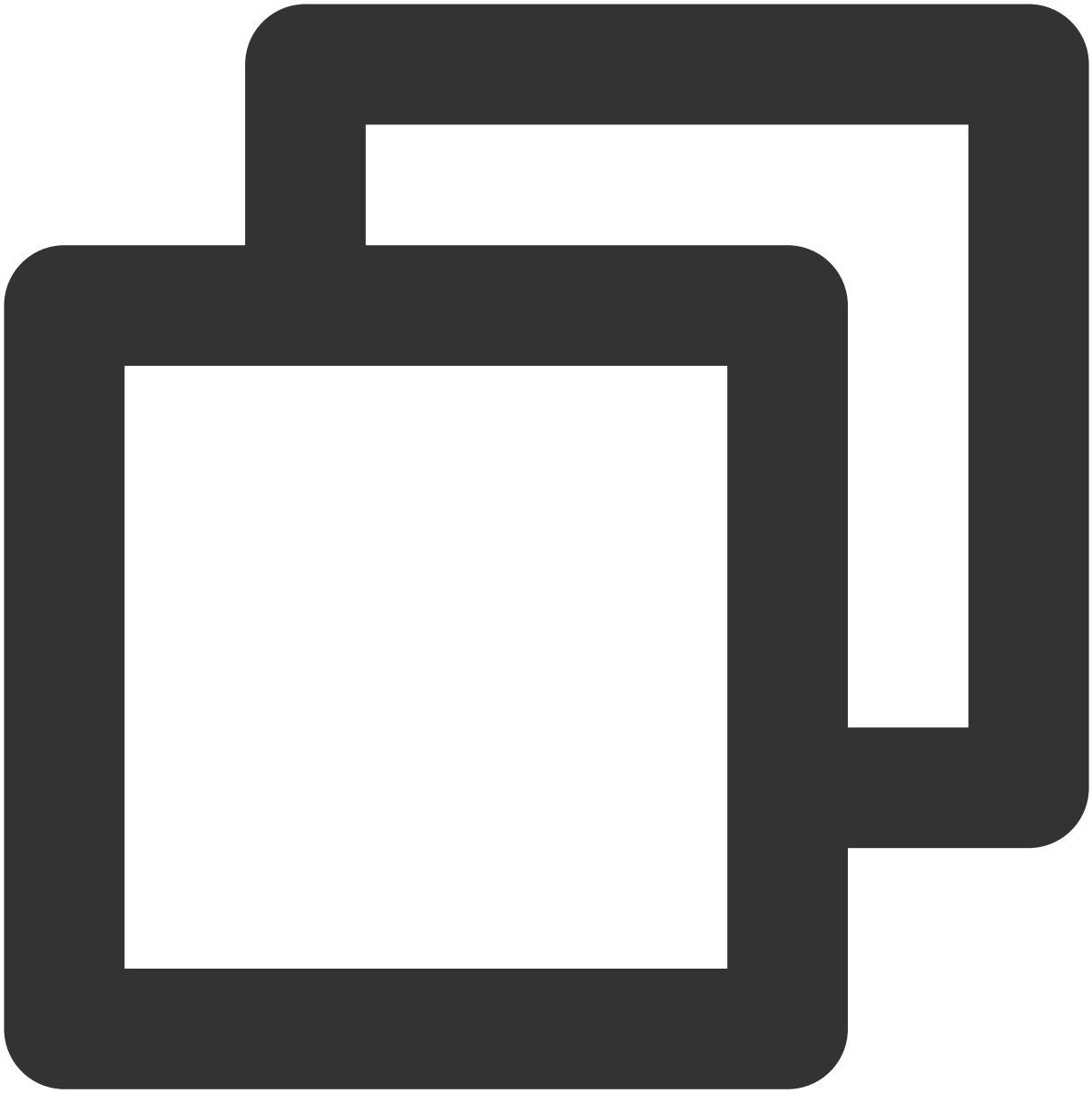
设置静态网站

功能说明

PUT Bucket website 用于为存储桶配置静态网站。

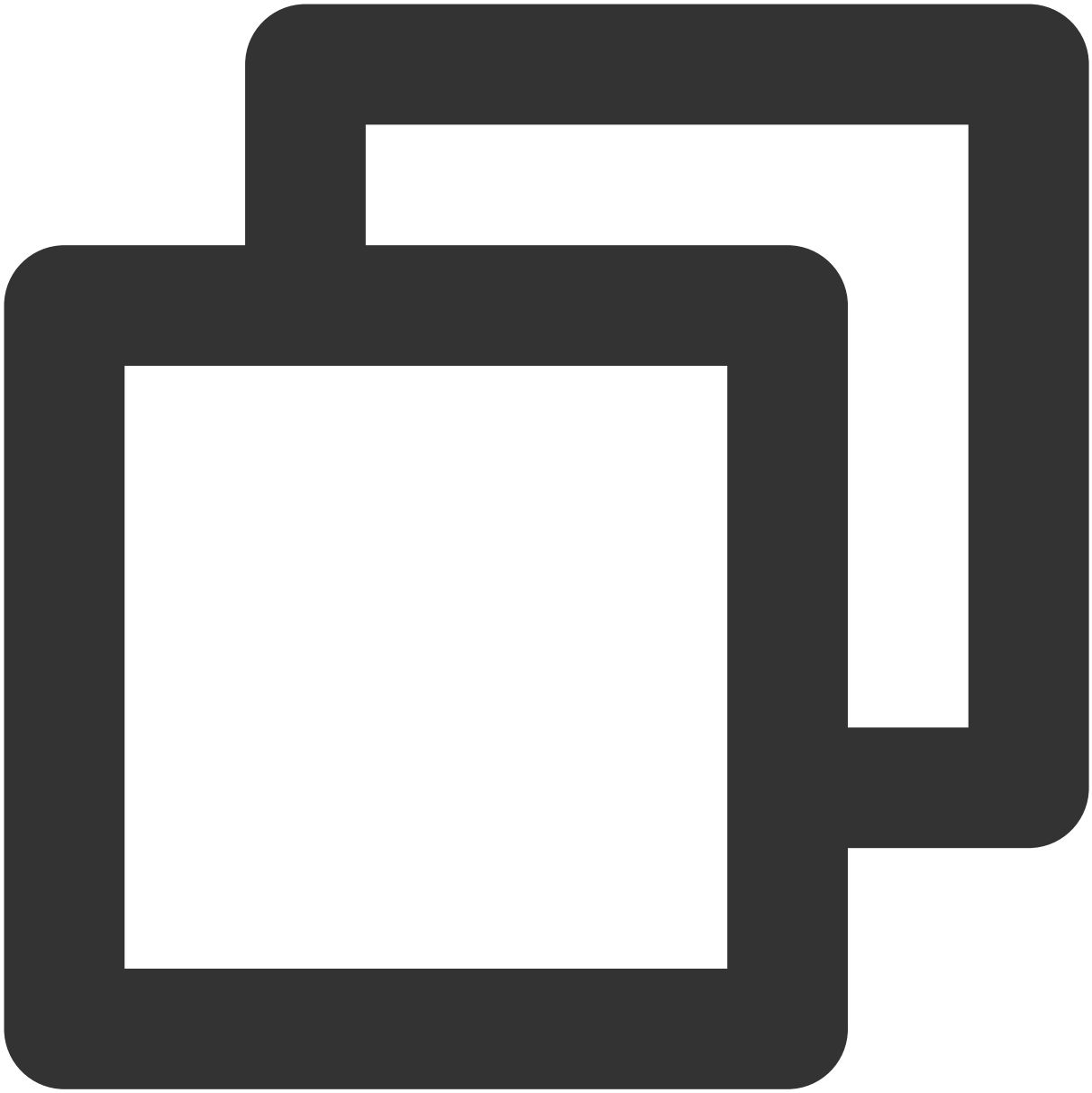
方法原型

版权所有：腾讯云计算（北京）有限责任公司



```
cos_status_t *cos_put_bucket_website(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     cos_website_params_t *website_params,
                                     cos_table_t **resp_header);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_website()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_website_params_t  *website_params = NULL;
    cos_website_rule_content_t *website_content = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
```

```
    //创建website参数
    website_params = cos_create_website_params(options->pool);
    cos_str_set(&website_params->index, "index.html");
    cos_str_set(&website_params->redirect_protocol, "https");
    cos_str_set(&website_params->error_document, "Error.html");

    website_content = cos_create_website_rule_content(options->pool);
    cos_str_set(&website_content->condition_errcode, "404");
    cos_str_set(&website_content->redirect_protocol, "https");
    cos_str_set(&website_content->redirect_replace_key, "404.html");
    cos_list_add_tail(&website_content->node, &website_params->rule_list);

    website_content = cos_create_website_rule_content(options->pool);
    cos_str_set(&website_content->condition_prefix, "docs/");
    cos_str_set(&website_content->redirect_protocol, "https");
    cos_str_set(&website_content->redirect_replace_key_prefix, "documents/");
    cos_list_add_tail(&website_content->node, &website_params->rule_list);

    website_content = cos_create_website_rule_content(options->pool);
    cos_str_set(&website_content->condition_prefix, "img/");
    cos_str_set(&website_content->redirect_protocol, "https");
    cos_str_set(&website_content->redirect_replace_key, "demo.jpg");
    cos_list_add_tail(&website_content->node, &website_params->rule_list);

    status = cos_put_bucket_website(options, &bucket, website_params, &resp_headers
    log_status(status);

    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_website();
```

```
    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	设置静态网站的存储桶，格式为 BucketName-APPID ， 详情请参见命名规范	String
website_params	存储桶静态网站配置信息	Struct
index	指定索引文档	String
redirect_protocol	指定全站重定向的协议，只能设置为 https	String
error_document	指定通用错误返回	String
rule_list	设置重定向规则，最多设置100条 RoutingRule	list
condition_errcode	指定重定向错误码，只支持配置4XX返回码，优先级高于 error_document	String
condition_prefix	指定前缀重定向的路径，替换指定的 folder/	String
redirect_protocol	指定重定向规定的协议，只能设置为 https	String
redirect_replace_key	替换整个 Key 为指定的内容	String
redirect_replace_key_prefix	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String

版权所有：腾讯云计算（北京）有限责任公司

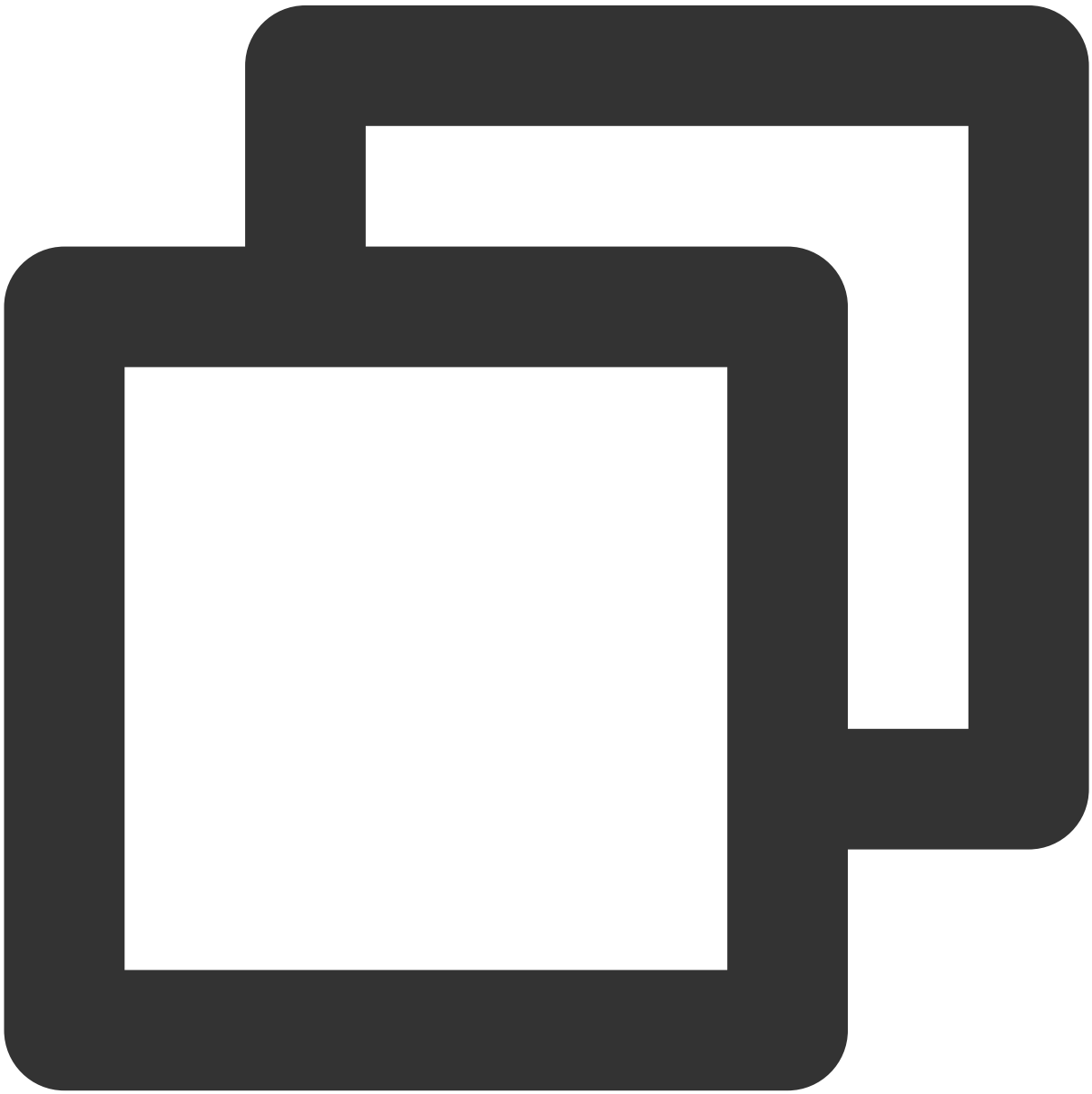
req_id	请求消息 ID	String

查询静态网站配置

功能说明

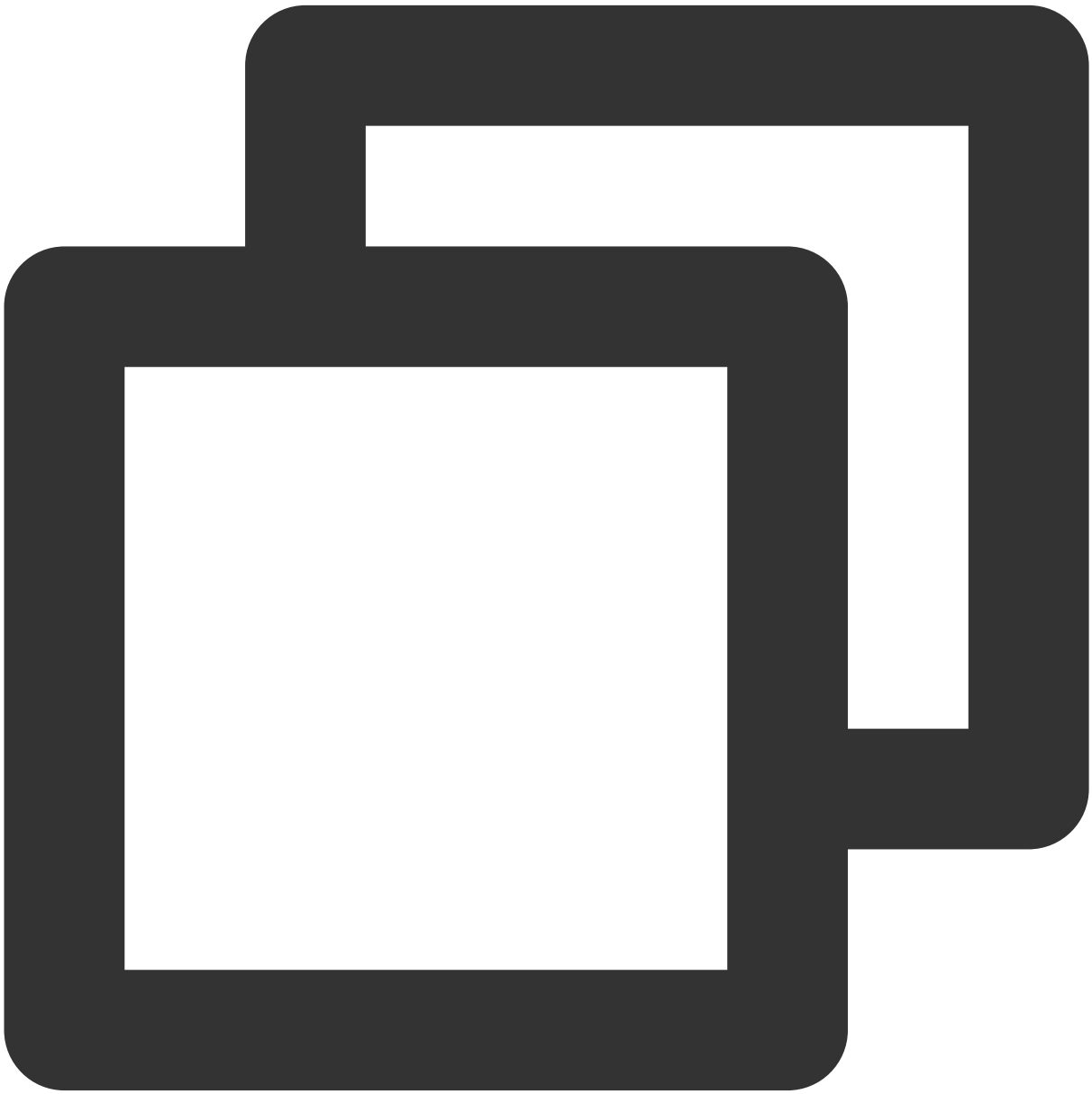
GET Bucket website 用于查询与存储桶关联的静态网站配置信息。

方法原型



```
cos_status_t *cos_get_bucket_website(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     cos_website_params_t *website_params,
                                     cos_table_t **resp_header);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
```

```
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_get_website()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_website_params_t  *website_result = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;

    //创建内存池
```

```
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    website_result = cos_create_website_params(options->pool);
    status = cos_get_bucket_website(options, &bucket, website_result, &resp_headers
    log_status(status);
    if (!cos_status_is_ok(status)) {
        cos_pool_destroy(pool);
        return;
    }

    //查看结果
    cos_website_rule_content_t *content = NULL;
    char *line = NULL;
    line = apr_psprintf(options->pool, "%.*s\\n", website_result->index.len, websit
    printf("index: %s", line);
    line = apr_psprintf(options->pool, "%.*s\\n", website_result->redirect_protocol
    printf("redirect protocol: %s", line);
    line = apr_psprintf(options->pool, "%.*s\\n", website_result->error_document.le
    printf("error document: %s", line);
    cos_list_for_each_entry(cos_website_rule_content_t, content, &website_result->r
        line = apr_psprintf(options->pool, "%.*s\\t%.*s\\t%.*s\\t%.*s\\t%.*s\\n", c
        printf("%s", line);
    }

    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);
```

```
    test_get_website();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	查询静态网站配置的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
website_params	存储桶静态网站配置信息	Struct
index	指定索引文档	String
redirect_protocol	指定全站重定向的协议，只能设置为 https	String
error_document	指定通用错误返回	String
rule_list	设置重定向规则，最多设置100条 RoutingRule	list
condition_errcode	指定重定向错误码，只支持配置4XX返回码，优先级高于 error_document	String
condition_prefix	指定前缀重定向的路径，替换指定的 folder/	String
redirect_protocol	指定重定向规定的协议，只能设置为 https	String
redirect_replace_key	替换整个 Key 为指定的内容	String
redirect_replace_key_prefix	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String

版权所有：腾讯云计算（北京）有限责任公司

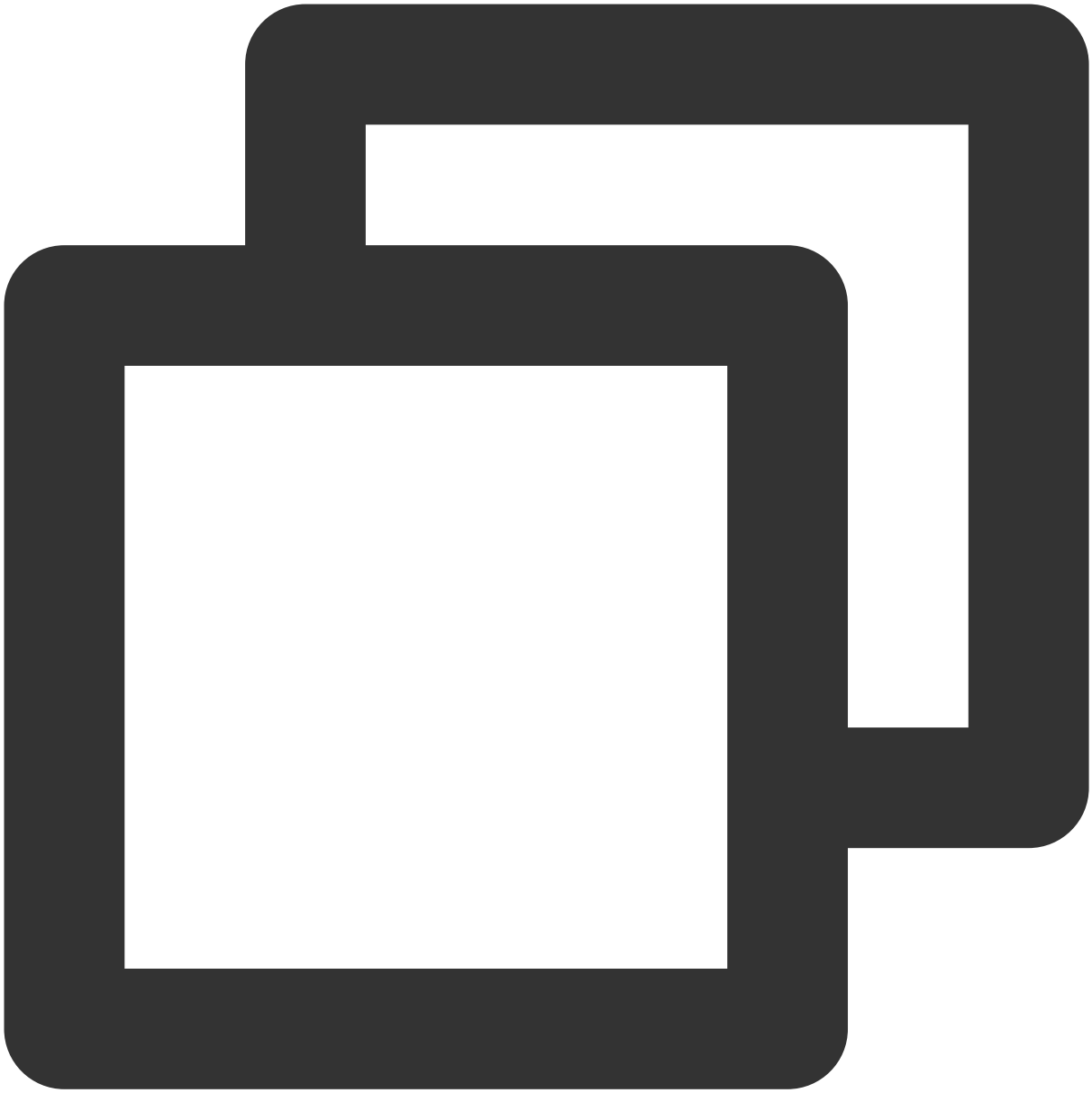
error_msg	错误码描述	String
req_id	请求消息 ID	String

删除静态网站配置

功能说明

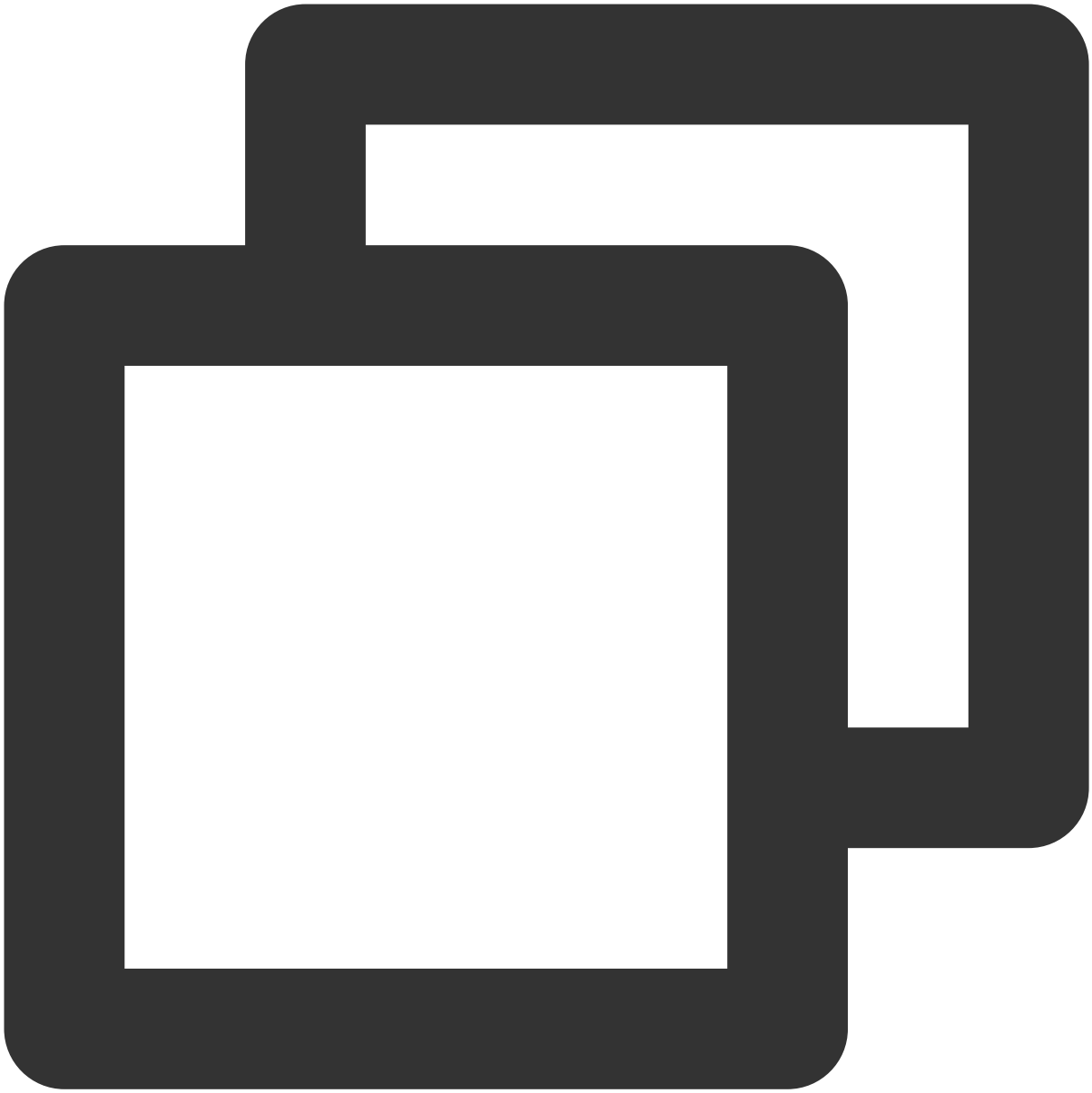
DELETE Bucket website 用于删除存储桶中的静态网站配置。

方法原型



```
cos_status_t *cos_delete_bucket_website(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        cos_table_t **resp_header);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_website()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    status = cos_delete_bucket_website(options, &bucket, &resp_headers);
    log_status(status);
```

```
    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_delete_website();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	被删除静态网站配置的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String

req_id	请求消息 ID	String

清单

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于清单的 API 概览以及 SDK 示例代码。

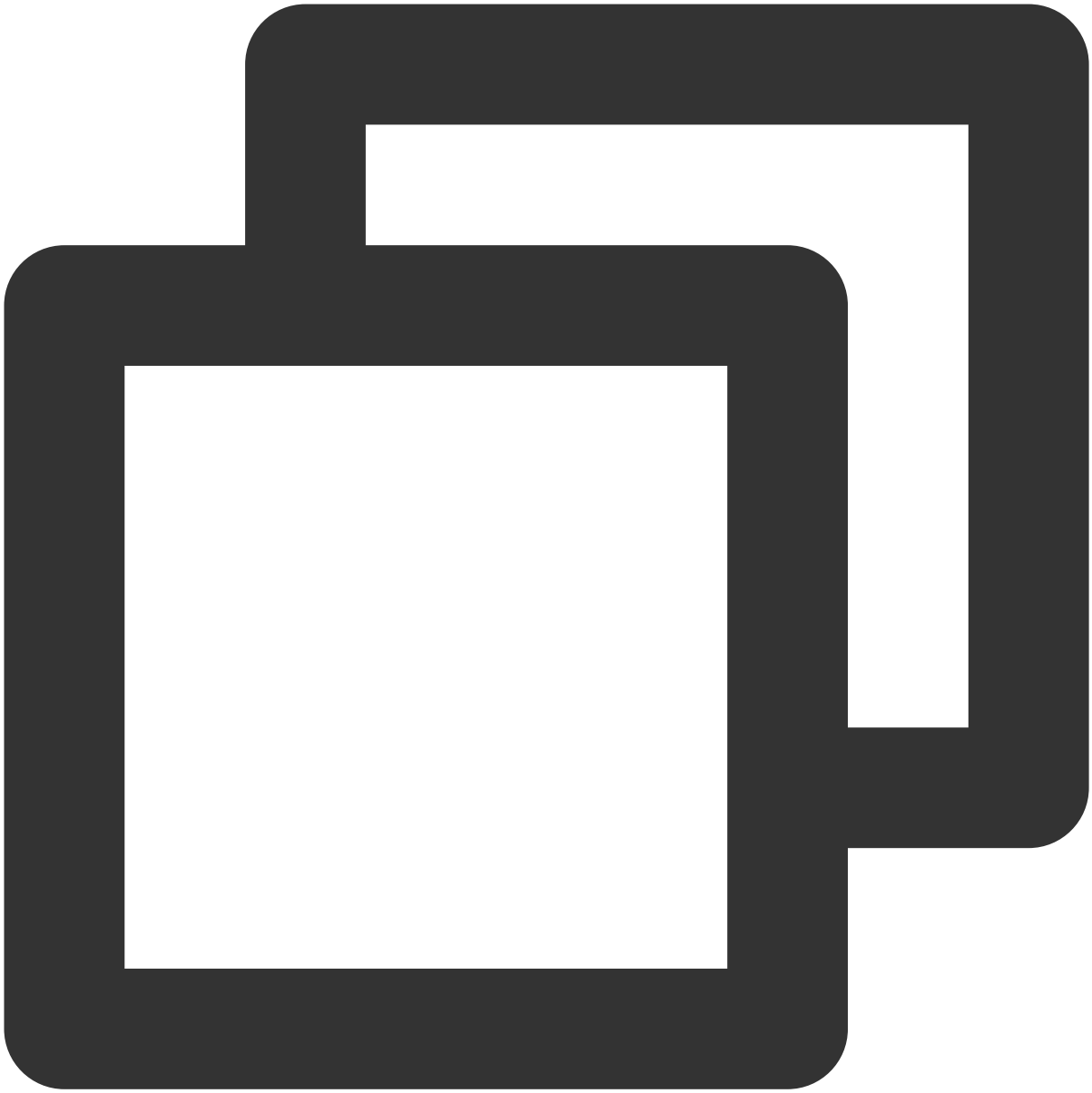
API	操作名	操作描述
PUT Bucket inventory	设置清单任务	设置存储桶的清单任务
GET Bucket inventory	查询清单任务	查询存储桶的清单任务
List Bucket Inventory Configurations	查询所有清单	查询存储桶的所有清单任务
DELETE Bucket inventory	删除清单任务	删除存储桶的清单任务

设置清单任务

功能说明

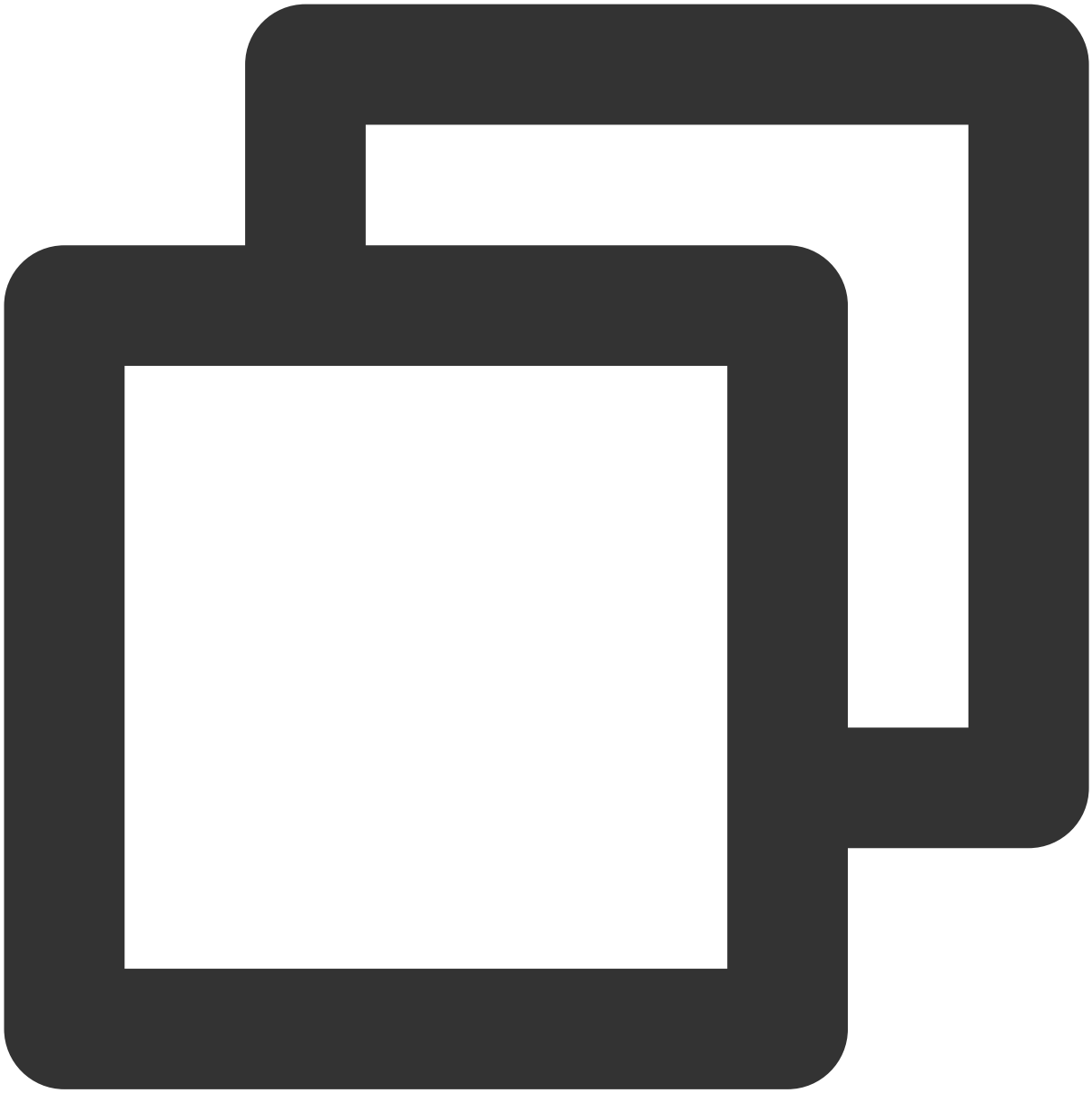
PUT Bucket inventory 用于在存储桶中创建清单任务。

方法原型



```
cos_status_t *cos_put_bucket_inventory(const cos_request_options_t *options,
                                       const cos_string_t *bucket,
                                       cos_inventory_params_t *inventory_params,
                                       cos_table_t **resp_headers
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
// 地域信息，枚举值可参见 https://cloud.tencent.com/document/product/436/6224 文档，例如
static char TEST_REGION[] = "ap-guangzhou";    //region in endpoint
// 对象拥有者，例如用户UIN：100000000001
static char TEST_UIN[] = "<Uin>";    //your uin
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_bucket_inventory()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    int inum = 3, i, len;
    char buf[inum][32];
    char dest_bucket[128];
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
```

```
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    // put bucket inventory
    len = snprintf(dest_bucket, 128, "qcs::cos:%s::%s", TEST_REGION, TEST_BUCKET_NA
    dest_bucket[len] = 0;
    // 设置多个清单
    for (i = 0; i < inum; i++) {
        cos_inventory_params_t *params = cos_create_inventory_params(pool);
        cos_inventory_optional_t *optional;
        len = snprintf(buf[i], 32, "id%d", i);
        buf[i][len] = 0;
        cos_str_set(&params->id, buf[i]);
        cos_str_set(&params->is_enabled, "true");
        cos_str_set(&params->frequency, "Daily");
        cos_str_set(&params->filter_prefix, "myPrefix");
        cos_str_set(&params->included_object_versions, "All");
        cos_str_set(&params->destination.format, "CSV");
        cos_str_set(&params->destination.account_id, TEST_UIN);
        cos_str_set(&params->destination.bucket, dest_bucket);
        cos_str_set(&params->destination.prefix, "invent");
        params->destination.encryption = 1;
        optional = cos_create_inventory_optional(pool);
        cos_str_set(&optional->field, "Size");
        cos_list_add_tail(&optional->node, &params->fields);
        optional = cos_create_inventory_optional(pool);
        cos_str_set(&optional->field, "LastModifiedDate");
        cos_list_add_tail(&optional->node, &params->fields);
        optional = cos_create_inventory_optional(pool);
        cos_str_set(&optional->field, "ETag");
        cos_list_add_tail(&optional->node, &params->fields);
        optional = cos_create_inventory_optional(pool);
        cos_str_set(&optional->field, "StorageClass");
        cos_list_add_tail(&optional->node, &params->fields);
        optional = cos_create_inventory_optional(pool);
        cos_str_set(&optional->field, "ReplicationStatus");
        cos_list_add_tail(&optional->node, &params->fields);

        // 设置存储桶清单
        status = cos_put_bucket_inventory(options, &bucket, params, &resp_headers);
        log_status(status);
    }

    // 销毁内存池
    cos_pool_destroy(pool);
}
```

```
int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_bucket_inventory();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	设置清单任务的存储桶，格式为 BucketName-APPID ，详情请参见命名规范	String
inventory_params	存储桶清单配置信息	Struct
node	用于list inventory接口链接清单配置	List
id	清单的名称，与请求参数中的 id 对应	String
is_enabled	清单是否启用的标识： 如果设置为 true，清单功能将生效 如果设置为 false，将不生成任何清单	Struct
frequency	清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly	String
filter_prefix	需要分析的对象的前缀	String

included_object_versions	是否在清单中包含对象版本： 如果设置为 All，清单中将会包含所有对象版本，并在清单中增加 VersionId，IsLatest，DeleteMarker 这几个字段 如果设置为 Current，则清单中不包含对象版本信息	String
destination	描述存放清单结果的信息	Struct
format	清单分析结果的文件形式，可选项为 CSV 格式	String
account_id	存储桶的所有者 ID，例如100000000001	String
bucket	清单分析结果的存储桶名	String
prefix	清单分析结果的前缀	String
encryption	为清单结果提供服务端加密的选项	Int
fields	设置清单结果中应包含的分析项目	Struct
field	清单结果中可选包含的分析项目名称，可选字段包括：Size，LastModifiedDate，StorageClass，ETag，IsMultipartUploaded，ReplicationStatus	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

错误码说明

该请求可能会发生的一些常见的特殊错误如下：

错误码	描述	状态码
InvalidArgument	不合法的参数值	HTTP 400 Bad Request
TooManyConfigurations	清单数量已经达到1000条的上限	HTTP 400 Bad Request
AccessDenied	未授权的访问。您可能不具备访问该存储桶的权限	HTTP 403 Forbidden

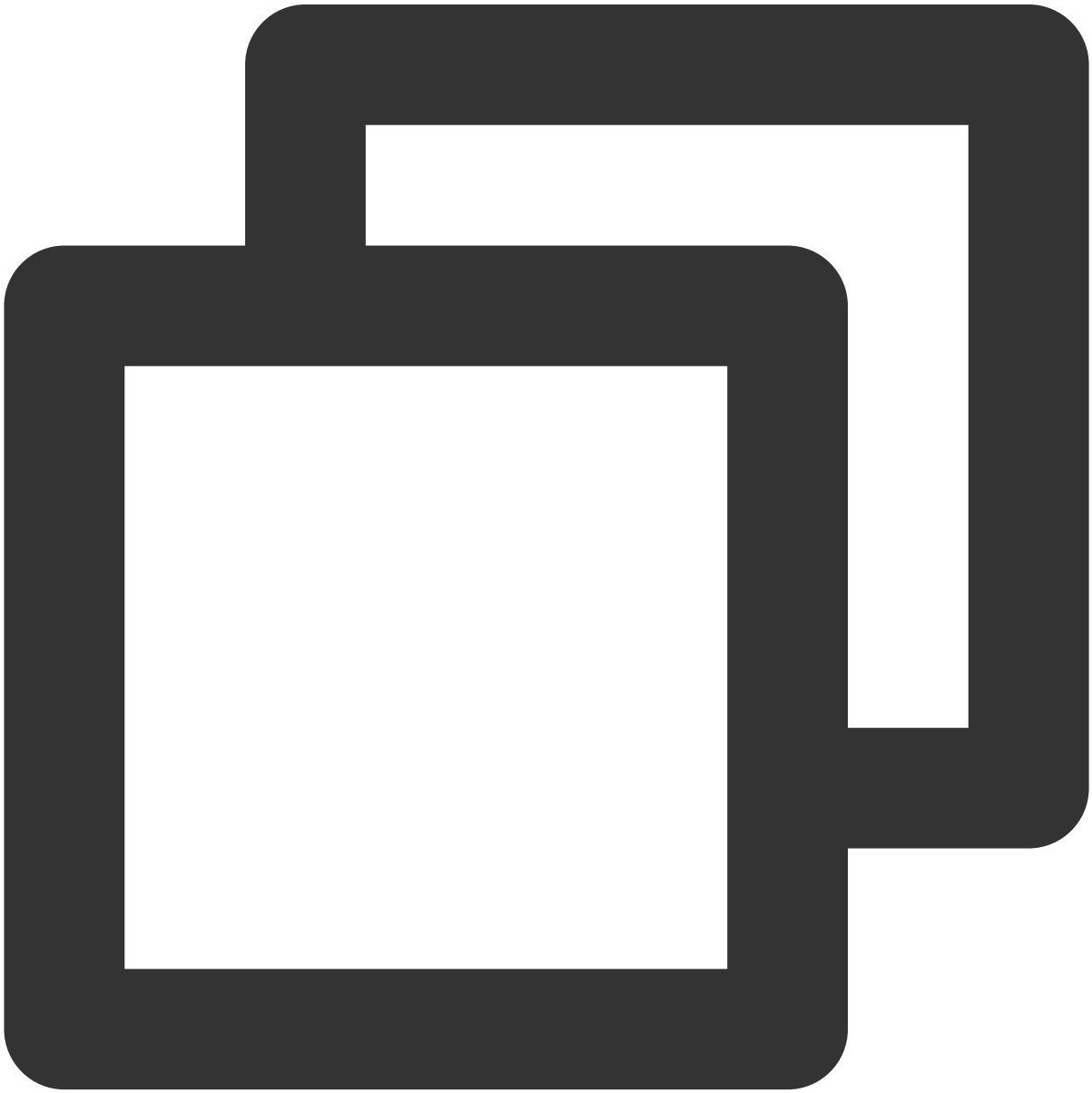
版权所有：腾讯云计算（北京）有限责任公司

查询清单任务

功能说明

GET Bucket inventory 用于查询存储桶中用户的清单任务信息。

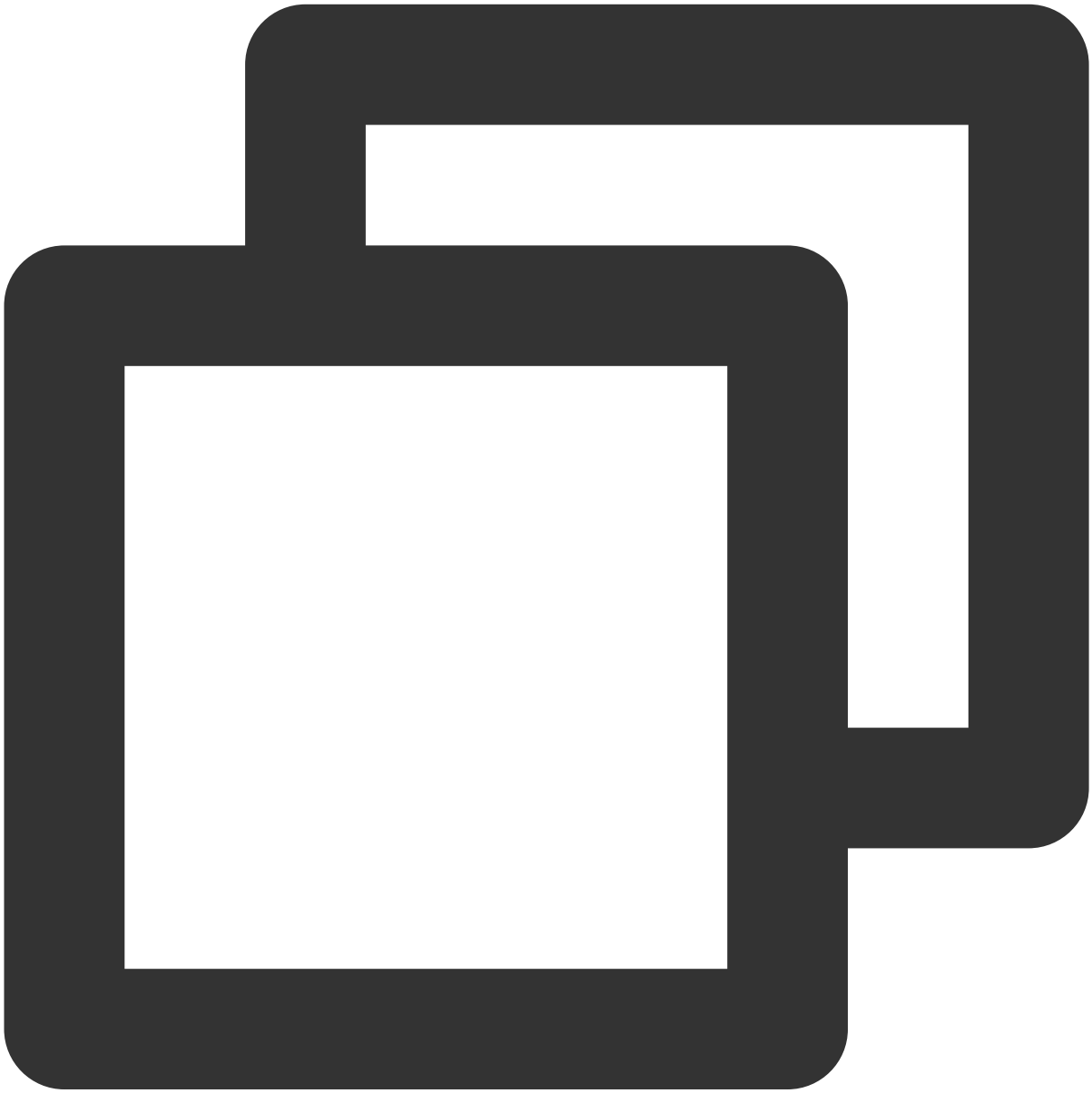
方法原型



```
cos_status_t *cos_get_bucket_inventory(const cos_request_options_t *options,
                                       const cos_string_t *bucket,
                                       cos_inventory_params_t *inventory_params,
```

```
                                        cos_table_t **resp_headers);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
```

```
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_get_bucket_inventory()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    int inum = 3;
    char buf[inum][32];
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;
    cos_inventory_params_t *get_params = NULL;
    cos_inventory_optional_t *optional = NULL;

    //创建内存池
    cos_pool_create(&pool, NULL);
```

```
    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    // get inventory
    get_params = cos_create_inventory_params(pool);
    cos_str_set(&get_params->id, buf[inum/2]);
    status = cos_get_bucket_inventory(options, &bucket, get_params, &resp_headers);
    log_status(status);

    printf("id: %s\\nis_enabled: %s\\nfrequency: %s\\nfilter_prefix: %s\\nincluded_
            get_params->id.data, get_params->is_enabled.data, get_params->frequency
    printf("destination:\\n");
    printf("\\tencryption: %d\\n", get_params->destination.encryption);
    printf("\\tformat: %s\\n", get_params->destination.format.data);
    printf("\\taccount_id: %s\\n", get_params->destination.account_id.data);
    printf("\\tbucket: %s\\n", get_params->destination.bucket.data);
    printf("\\tprefix: %s\\n", get_params->destination.prefix.data);
    cos_list_for_each_entry(cos_inventory_optional_t, optional, &get_params->fields
        printf("field: %s\\n", optional->field.data);
    }

    // 销毁内存池
    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_get_bucket_inventory();

    cos_http_io_deinitialize();
```

```
    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	查询清单任务的存储桶，格式为 BucketName-APPID ，详情请参见命名规范	String
inventory_params	用于 list inventory 接口链接清单配置	Struct
node	链接清单配置	List
id	清单的名称，与请求参数中的 id 对应	String
is_enabled	清单是否启用的标识： 如果设置为 true，清单功能将生效 如果设置为 false，将不生成任何清单	Struct
frequency	清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly	String
filter_prefix	需要分析的对象的前缀	String
included_object_versions	是否在清单中包含对象版本： 如果设置为 All，清单中将会包含所有对象版本，并在清单中增加 VersionId，IsLatest，DeleteMarker 这几个字段 如果设置为 Current，则清单中不包含对象版本信息	String
destination	描述存放清单结果的信息	Struct
format	清单分析结果的文件形式，可选项为 CSV 格式	String
account_id	存储桶的所有者 ID，例如100000000001	String
bucket	清单分析结果的存储桶名	String
prefix	清单分析结果的前缀	String
encryption	为清单结果提供服务端加密的选项	Int
fields	设置清单结果中应包含的分析项目	Struct
field	清单结果中可选包含的分析项目名称，可选字段包括：Size，LastModifiedDate，StorageClass，ETag，IsMultipartUploaded，ReplicationStatus	String

resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

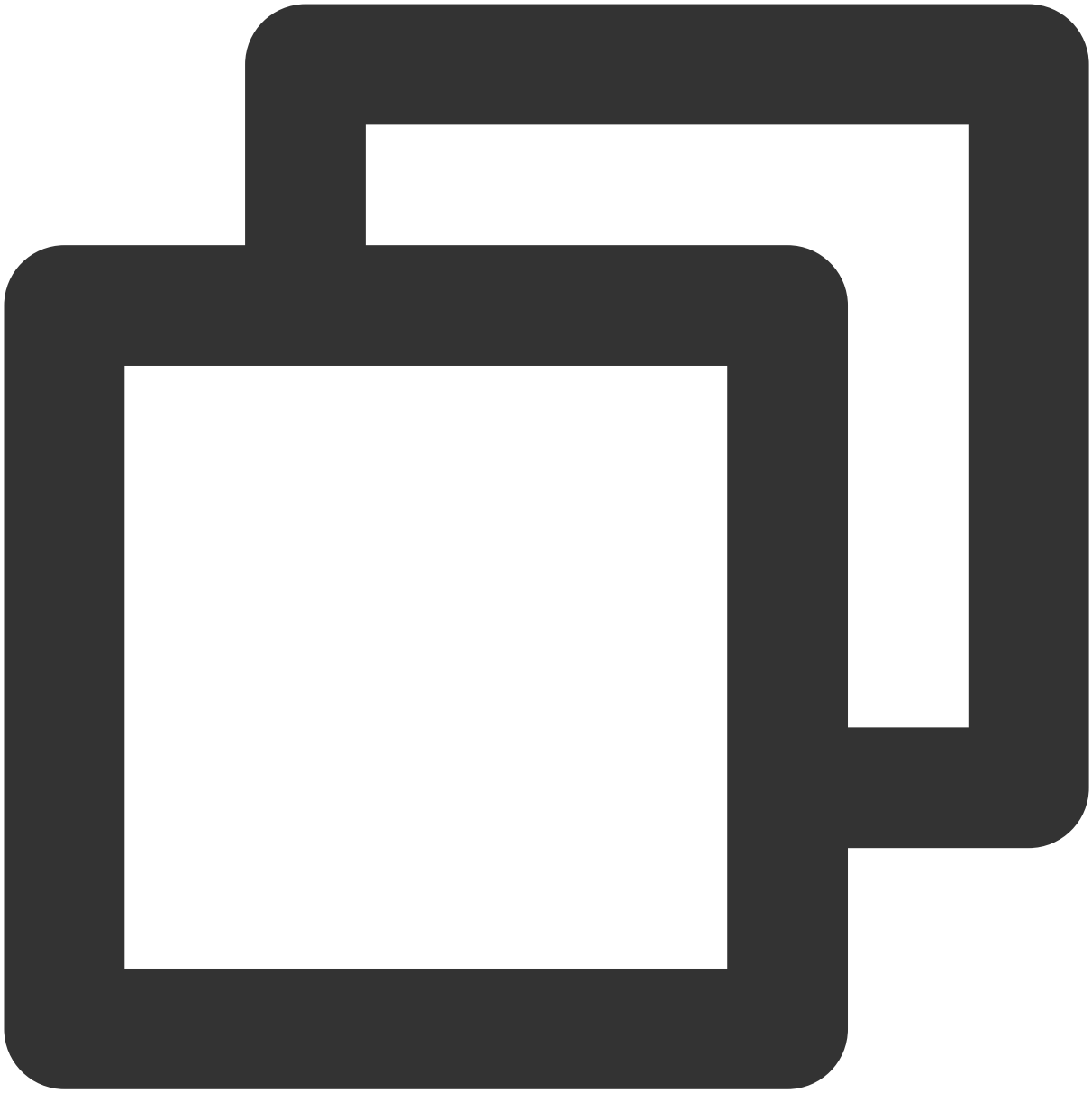
返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

查询所有清单

功能说明

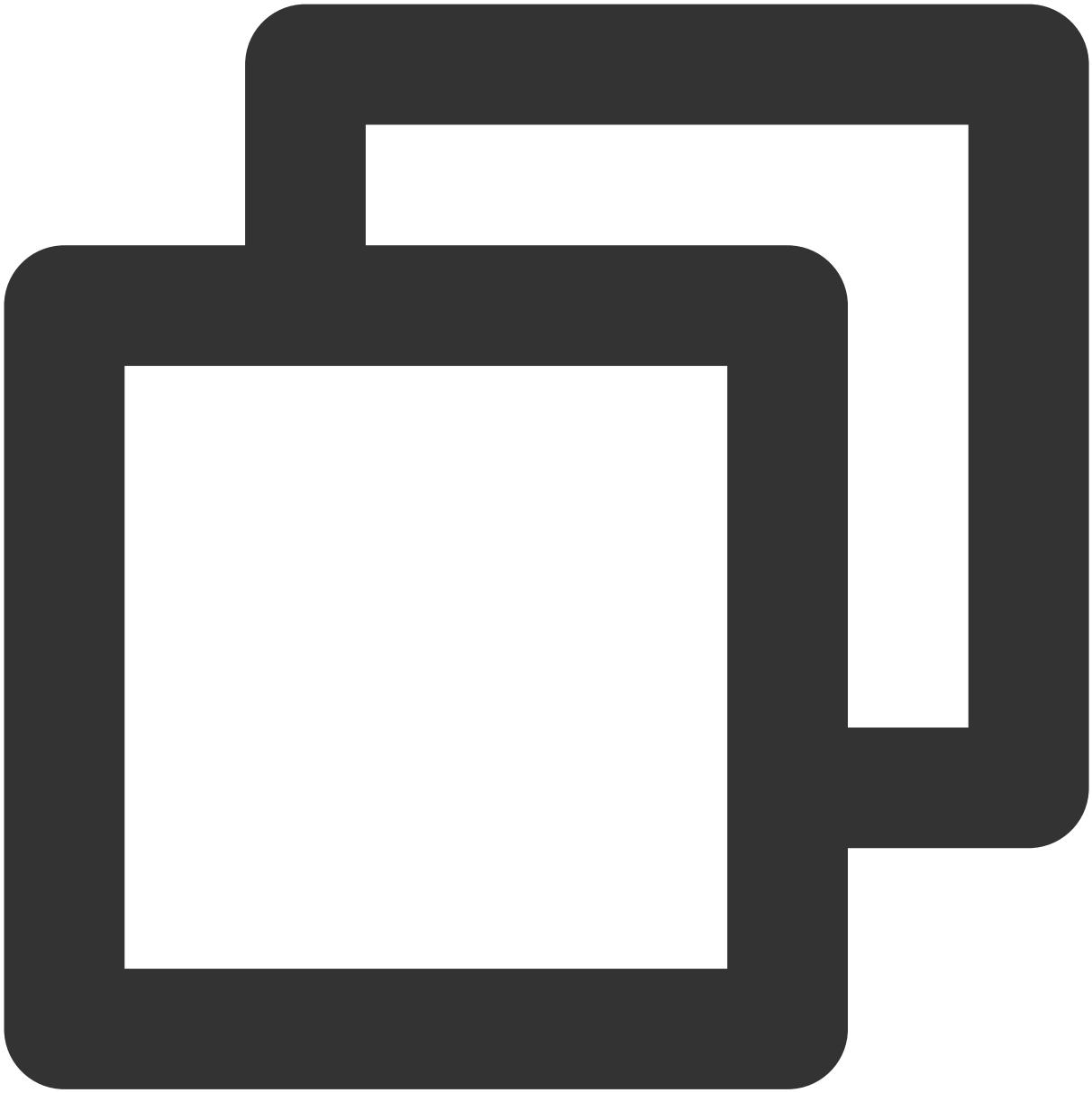
List Bucket Inventory Configurations 用于请求返回一个存储桶中的所有清单任务。每一个存储桶中最多配置1000条清单任务。

方法原型



```
cos_status_t *cos_list_bucket_inventory(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        cos_list_inventory_params_t *inventory_params
                                        cos_table_t **resp_headers);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_list_bucket_inventory()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;
    cos_inventory_params_t *get_params = NULL;
    cos_inventory_optional_t *optional = NULL;
    cos_list_inventory_params_t *list_params = NULL;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
```

```
    // list inventory
    list_params = cos_create_list_inventory_params(pool);
    status = cos_list_bucket_inventory(options, &bucket, list_params, &resp_headers
    log_status(status);

    // 查看结果
    get_params = NULL;
    cos_list_for_each_entry(cos_inventory_params_t, get_params, &list_params->inven
        printf("id: %s\\nis_enabled: %s\\nfrequency: %s\\nfilter_prefix: %s\\ninclu
        printf("destination:\\n");
        printf("\\tencryption: %d\\n", get_params->destination.encryption);
        printf("\\tformat: %s\\n", get_params->destination.format.data);
        printf("\\taccount_id: %s\\n", get_params->destination.account_id.data);
        printf("\\tbucket: %s\\n", get_params->destination.bucket.data);
        printf("\\tprefix: %s\\n", get_params->destination.prefix.data);
        cos_list_for_each_entry(cos_inventory_optional_t, optional, &get_params->fi
            printf("field: %s\\n", optional->field.data);
        }
    }

    // 销毁内存池
    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_list_bucket_inventory();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	存放清单的目标存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
inventory_params	存储桶所有清单配置信息	Struct
inventorys	链接 cos_inventory_params_t 的 node 成员	List
is_truncated	是否已列出所有清单任务信息的标识。如果已经展示完则为 false，否则为 true	Struct
continuation_token	当页清单列表的标识，可理解为页数。该标识与请求中的 continuation-token 参数对应	String
next_continuation_token	下一页清单列表的标识。如果该参数中有值，则可将该值作为 continuation-token 参数并发起 GET 请求以获取下一页清单任务信息	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

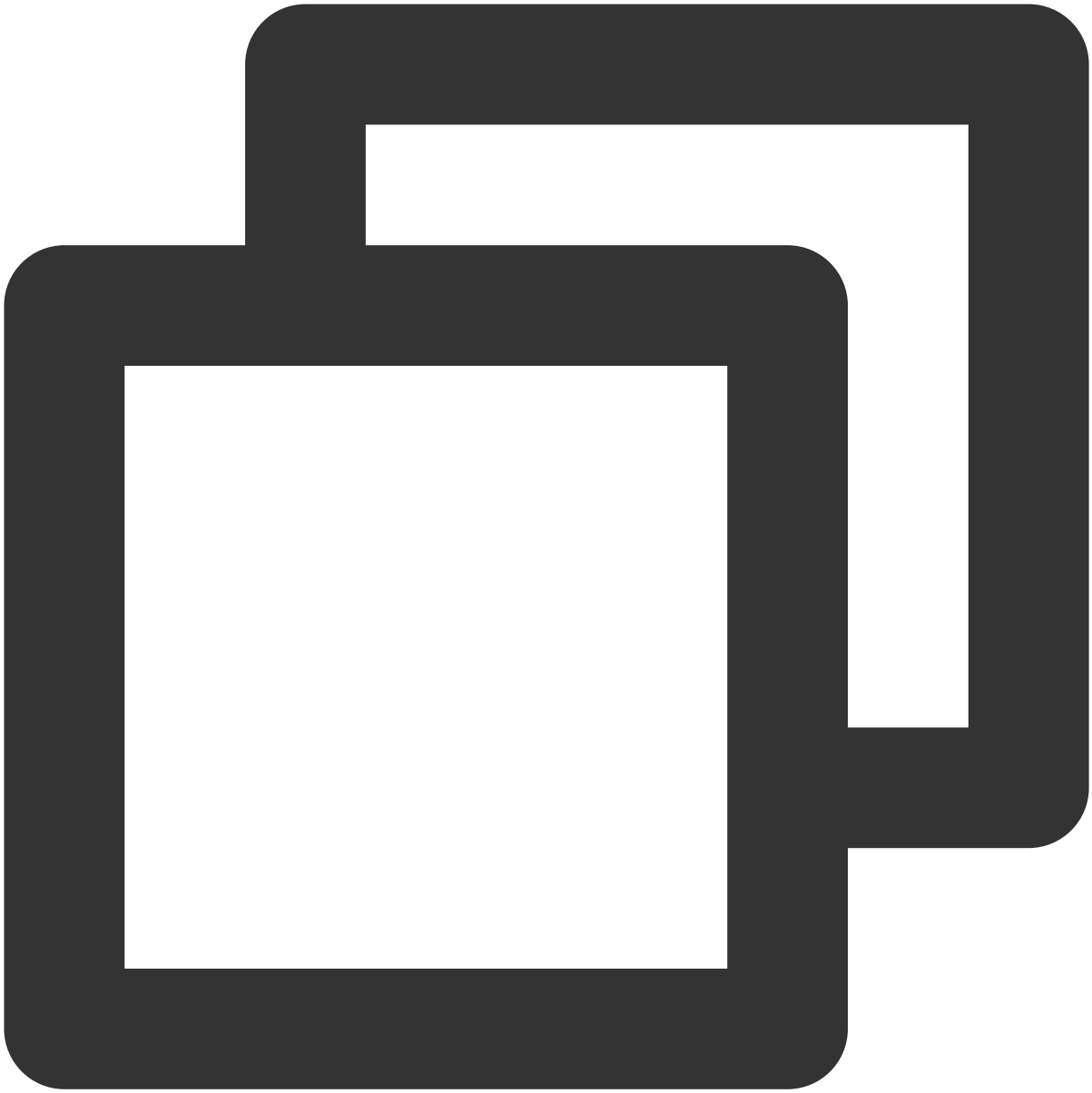
返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

删除清单任务

功能说明

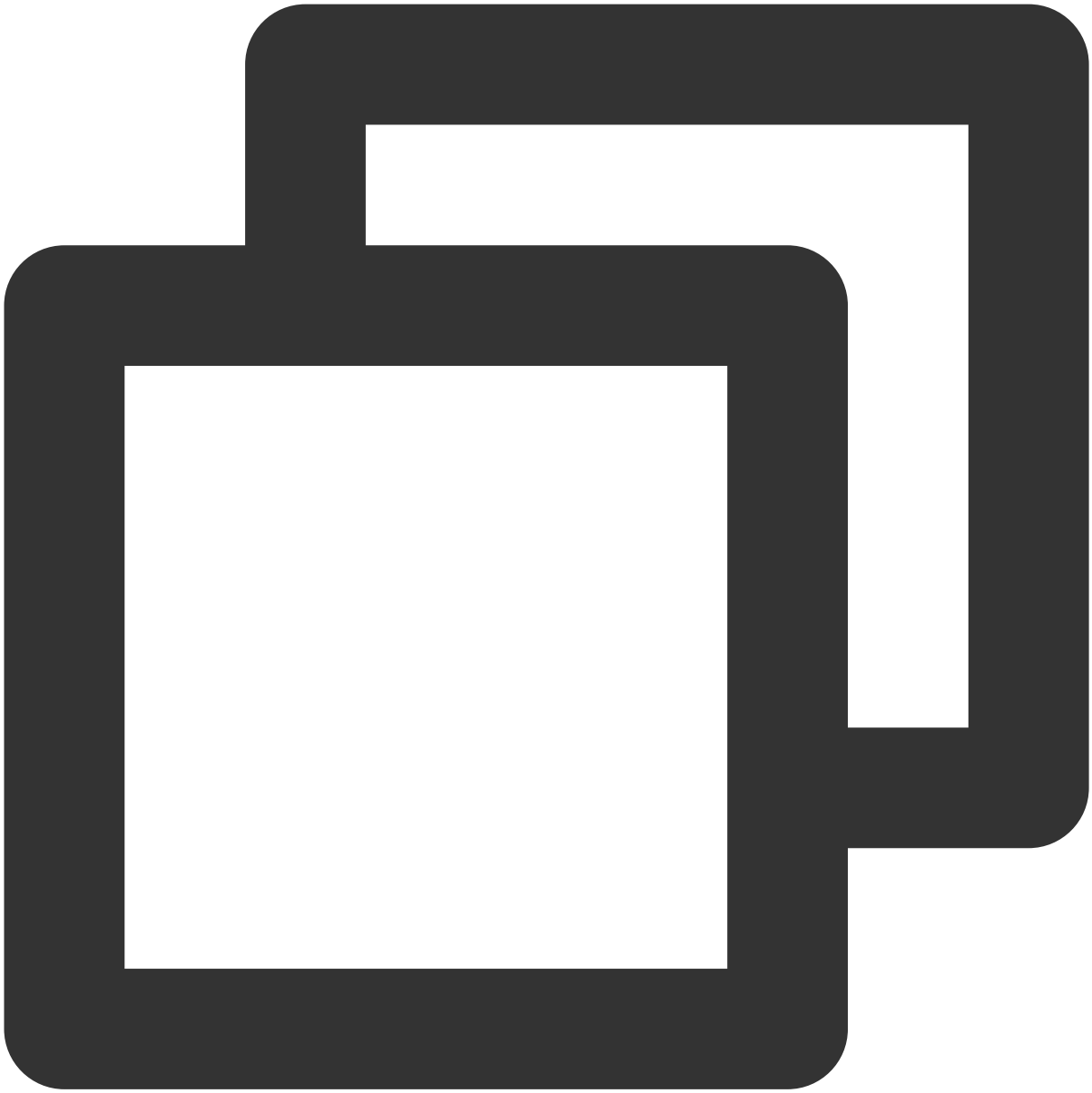
DELETE Bucket inventory 用于删除存储桶中指定的清单任务。

方法原型



```
cos_status_t *cos_delete_bucket_inventory(const cos_request_options_t *options,
                                          const cos_string_t *bucket,
                                          const cos_string_t *id,
                                          cos_table_t **resp_headers);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_bucket_inventory()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;
    int inum = 3, i;
    char buf[inum][32];

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    // delete inventory
```

```
    for (i = 0; i < inum; i++) {
        cos_string_t id;
        cos_str_set(&id, buf[i]);
        status = cos_delete_bucket_inventory(options, &bucket, &id, &resp_headers);
        log_status(status);
    }

    // 销毁内存池
    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_delete_bucket_inventory();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	被删除清单任务的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
id	清单的名称	String
resp_headers	返回 HTTP 响应消息的头域	Struct

版权所有：腾讯云计算（北京）有限责任公司

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

内容审核
视频审核

最近更新时间：2024-01-04 17:06:08

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

说明

使用内容审核服务需拥有数据万象使用权限：

主账号请 单击此处 进行角色授权。

子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于视频审核的 API 概览和 SDK 示例代码。

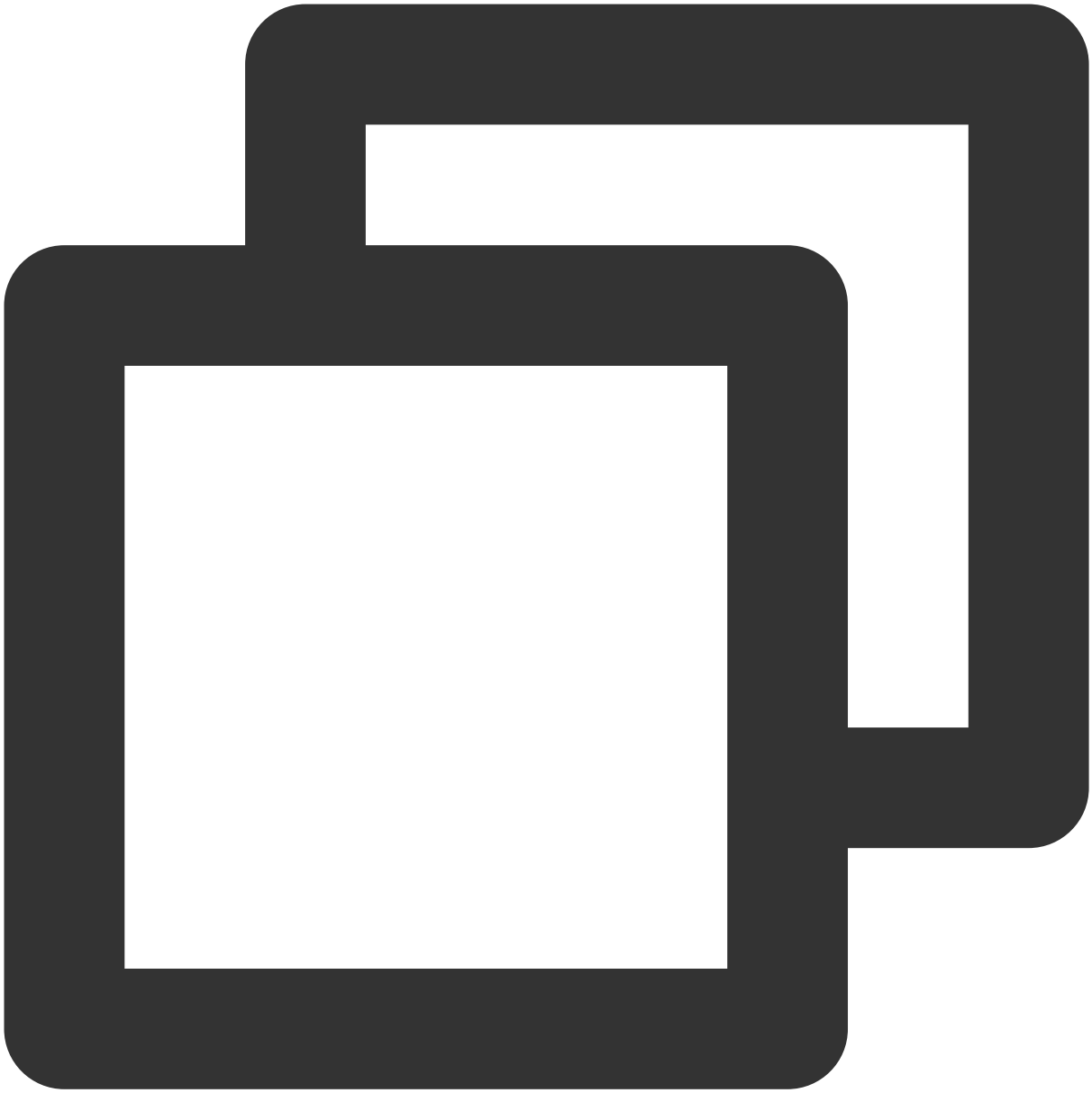
API	操作描述
提交视频审核任务	用于提交一个视频审核任务
查询视频审核任务结果	用于查询指定的视频审核任务

提交视频审核任务

功能说明

用于提交一个视频审核任务。

方法原型



```
cos_status_t *ci_create_video_auditing_job(const cos_request_options_t *options,
                                           const cos_string_t *bucket,
                                           const ci_video_auditing_job_options_t *j
                                           cos_table_t *headers,
                                           cos_table_t **resp_headers,
                                           ci_video_auditing_job_result_t **job_res
```

参数说明

参数名称	参数描述	类型

options	COS 请求选项	Struct
bucket	存储桶名称，Bucket 的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
job_options	提交视频审核任务的参数配置	Struct
input_object	对象名	String
detect_type	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads	String
callback	回调地址，以http://或者https://开头的地址	String
callback_version	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple	String
biz_type	审核策略，不带审核策略时使用默认策略	String
detect_content	用于指定是否审核视频声音，当值为0时：表示只审核视频画面截图；值为1时：表示同时审核视频画面截图和视频声音。默认值为0	Integer
snapshot	视频画面的审核通过视频截帧能力截取出一定量的截图，通过对截图逐一审核而实现的，该参数用于指定视频截帧的配置	Struct
mode	截帧模式。Interval 表示间隔模式，Average 表示平均模式，Fps 表示固定帧率模式。 Interval 模式：TimeInterval，Count 参数生效。当设置 Count，未设置 TimeInterval 时，表示截取所有帧，共 Count 张图片 Average 模式：Count 参数生效。表示整个视频，按平均间隔截取共 Count 张图片 Fps 模式：TimeInterval 表示每秒截取多少帧，Count 表示共截取多少帧	String
count	视频截帧数量，范围为(0, 10000]	Integer
time_interval	视频截帧频率，范围为(0, 60]，单位为秒，支持 float 格式，执行精度精确到毫秒	Float
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct
job_result	返回提交任务的响应信息	Struct
job_id	本次视频审核任务的 ID	String
state	视频审核任务的状态，值为 Submitted（已提交审核）、Snapshoting（视频截帧中）、Success（审核成功）、Failed（审核失败）、Auditing（审核	String

版权所有：腾讯云计算（北京）有限责任公司

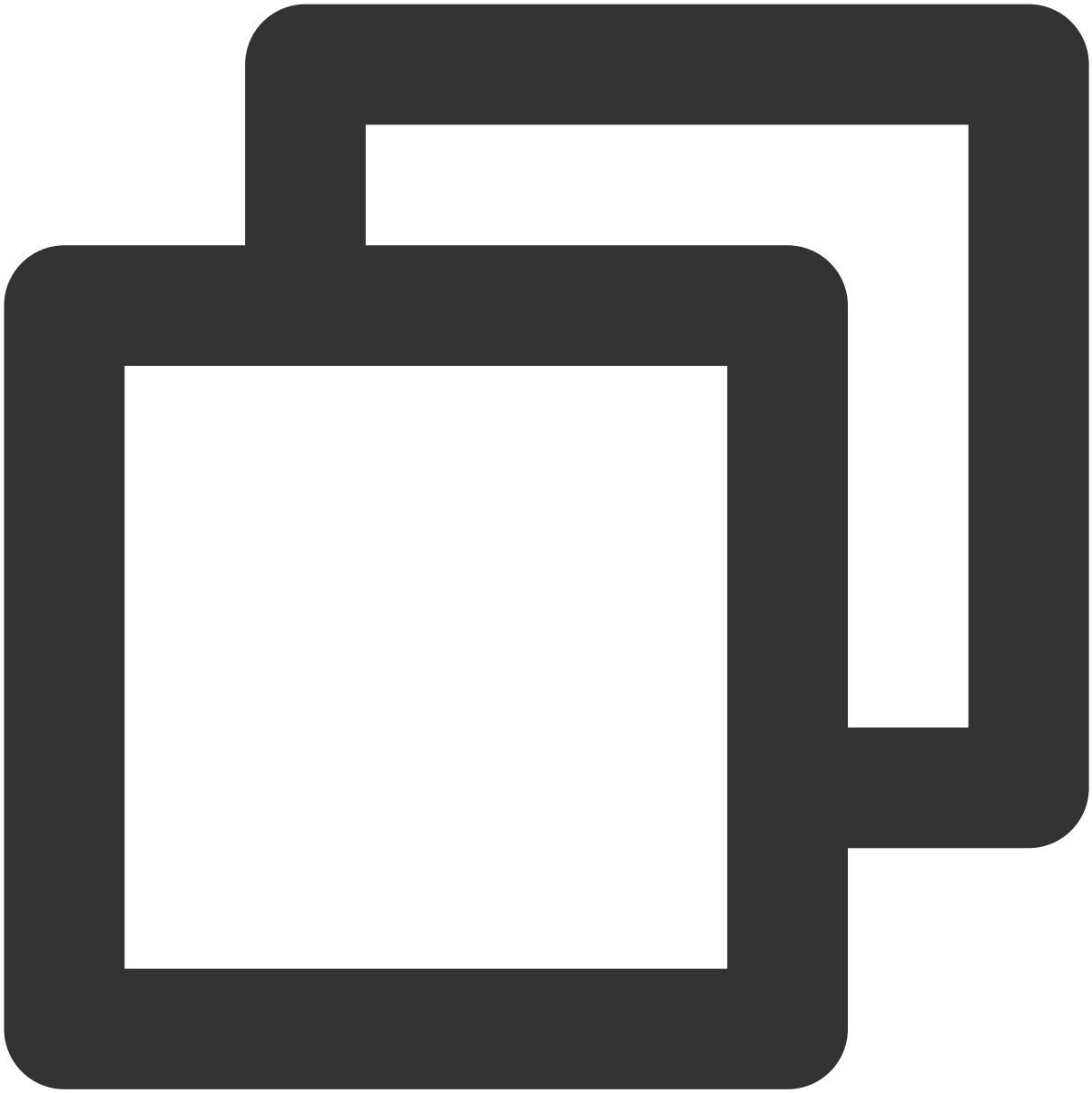
	中）其中一个	
creation_time	视频审核任务的创建时间	String

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

完整代码请参见 cos_demo.c 中的 test_ci_video_auditing() 函数。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <sys/stat.h>

// 数据万象的访问域名，详情请参见 https://intl.cloud.tencent.com/document/product/1045/3
static char TEST_CI_ENDPOINT[] = "https://ci.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

static void log_video_auditing_result(ci_video_auditing_job_result_t *result)
{
    cos_warn_log("jobid: %s", result->jobs_detail.job_id.data);
    cos_warn_log("state: %s", result->jobs_detail.state.data);
    cos_warn_log("creation_time: %s", result->jobs_detail.creation_time.data);
}

void test_ci_create_video_auditing()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers;
    ci_video_auditing_job_options_t *job_options;
    ci_video_auditing_job_result_t *job_result;

    // 基本配置
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    options->config = cos_config_create(options->pool);
    cos_str_set(&options->config->endpoint, TEST_CI_ENDPOINT);     // https://ci.<R
    cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&options->config->appid, TEST_APPID);
    options->config->is_cname = is_cname;
    options->ctl = cos_http_controller_create(options->pool, 0);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    // 替换为您的配置信息，可参见文档 https://intl.cloud.tencent.com/document/product/43
    job_options = ci_video_auditing_job_options_create(p);
    cos_str_set(&job_options->input_object, "test.mp4");
    cos_str_set(&job_options->job_conf.detect_type, "Porn,Ads");
    cos_str_set(&job_options->job_conf.callback_version, "Detail");
```

```
    job_options->job_conf.detect_content = 1;
    cos_str_set(&job_options->job_conf.snapshot.mode, "Interval");
    job_options->job_conf.snapshot.time_interval = 1.5;
    job_options->job_conf.snapshot.count = 10;

    // 提交一个视频审核任务
    s = ci_create_video_auditing_job(options, &bucket, job_options, NULL, &resp_hea
    log_status(s);
    if (s->code == 200) {
        log_video_auditing_result(job_result);
    }

    // 销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_ci_create_video_auditing();

    cos_http_io_deinitialize();

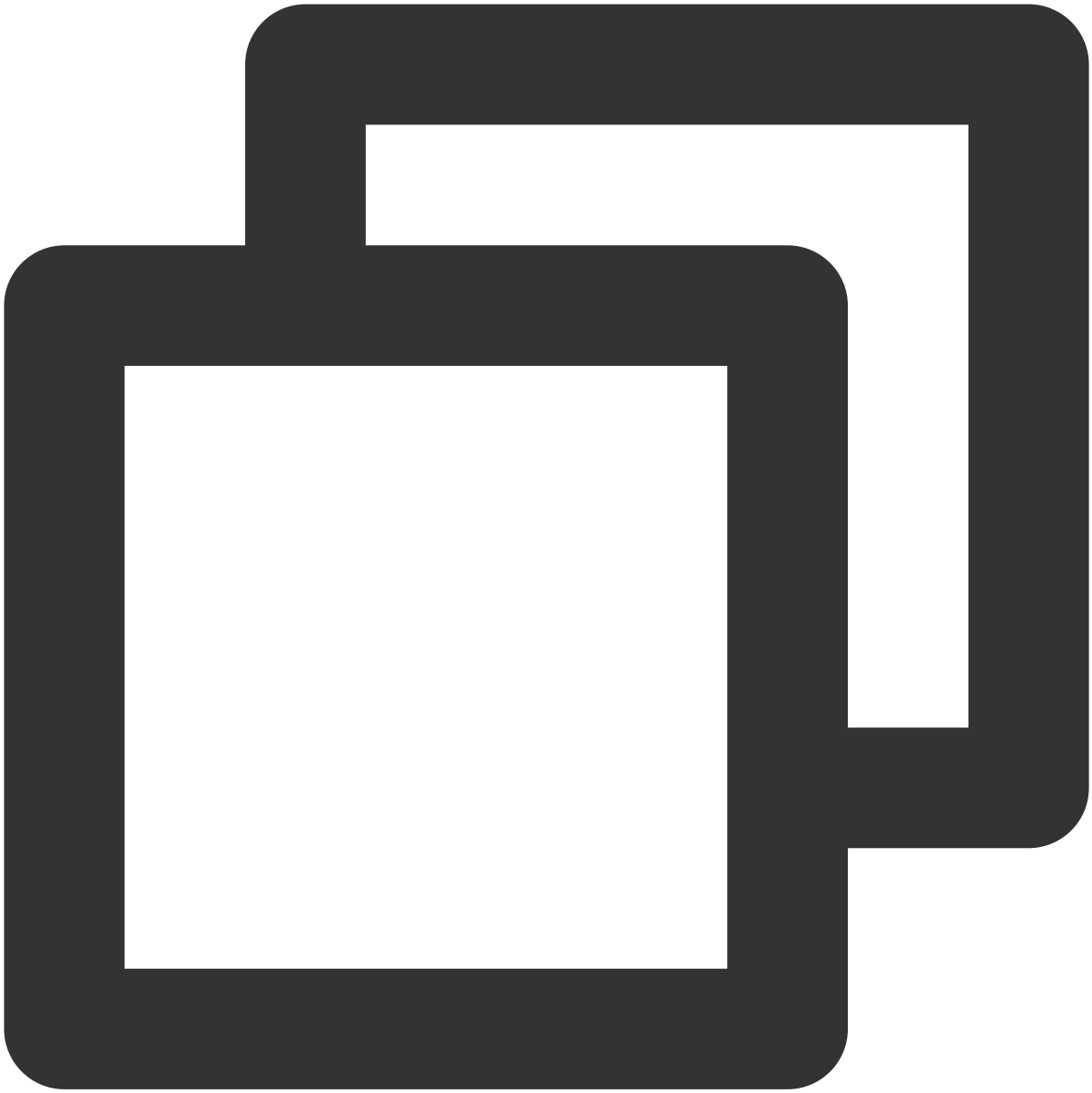
    return 0;
}
```

查询视频审核任务

功能说明

用于查询视频审核任务执行情况和结果。

方法原型



```
cos_status_t *ci_get_auditing_job(const cos_request_options_t *options,
                                  const cos_string_t *bucket,
                                  const cos_string_t *job_id,
                                  cos_table_t *headers,
                                  cos_table_t **resp_headers,
                                  ci_auditing_job_result_t **job_result);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，Bucket 的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
job_id	本次视频审核任务的 ID	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct
job_result	返回查询任务的响应信息	Struct
nonexist_job_ids	查询的 ID 中不存在任务，所有任务都存在时不返回	String
jobs_detail	视频审核任务的详细信息	Struct
code	错误码，只有 State为 Failed 时有意义	String
message	错误描述，只有 State 为 Failed 时有意义	String
job_id	视频审核任务的 ID	String
state	视频审核任务的状态，值为 Submitted（已提交审核）、Snapshoting（视频截帧中）、Success（审核成功）、Failed（审核失败）、Auditing（审核中）其中一个	String
creation_time	视频审核任务的创建时间	String
object	被审核的视频文件的名称	String
snapshot_count	视频截图的总数量	String
result	该字段表示本次判定的审核结果，您可以根据该结果，进行后续的操作；建议您按照业务所需，对不同的审核结果进行相应处理。有效值：0（审核正常），1 （判定为违规敏感文件），2（疑似敏感，建议人工复核）	Integer
porn_info	审核场景为涉黄的审核结果信息	Struct
ads_info	审核场景为广告引导的审核结果信息	Struct
hit_flag	是否命中该审核分类，0表示未命中，1表示命中，2表示疑似	Integer
count	命中该审核分类的截图张数	Integer
snapshot_info_list	该字段用于返回视频中视频画面截图审核的结果。注意：每次查看数据	Struct

版权所有：腾讯云计算（北京）有限责任公司

	的有效期为2小时，2小时后如需查看，请重新发起查询请求	
audio_section_info_list	字段用于返回视频中视频声音审核的结果。注意：每次查看数据的有效期为2小时，2小时后如需查看，请重新发起查询请求	Struct

snapshot_info_list 所对应的结构体说明

参数名称	参数描述	类型
url	视频截图的访问地址，您可以通过该地址查看该截图内容，地址格式为标准 URL 格式	String
snapshot_time	该字段用于返回当前截图位于视频中的时间，单位为毫秒。例如5000（视频开始后5000毫秒）	Integer
text	本该字段用于返回当前截图的图片 OCR 文本识别的检测结果（仅在审核策略开启文本内容检测时返回），识别上限为5000字节	String
porn_info	审核场景为涉黄的审核结果信息	Struct
ads_info	审核场景为广告引导的审核结果信息	Struct
hit_flag	是否命中该审核分类，0表示未命中，1表示命中，2表示疑似	Integer
score	该字段表示审核结果命中审核信息的置信度，取值范围：0（置信度最低）-100（置信度最高），越高代表该内容越有可能属于当前返回审核信息。例如：色情 99，则表明该内容非常有可能属于色情内容	Integer
label	该字段为兼容旧版本的保留字段，表示该截图的结果标签（可能为 SubLabel，可能为人物名字等）	String
sub_lable	该字段表示审核命中的具体子标签，例如：Porn 下的 SexBehavior 子标签。注意：该字段可能返回空，表示未命中具体的子标签	String

audio_section_info_list 所对应的结构体说明

参数名称	参数描述	类型
url	视频声音片段的访问地址，您可以通过该地址获取该声音片段的内容，地址格式为标准 URL 格式	String
text	该字段用于返回当前视频声音的 ASR 文本识别的检测结果（仅在审核策略开启文本内容检测时返回），识别上限为5小时	String
offset_time	视该字段用于返回当前声音片段位于视频中的时间，单位为毫秒，例如5000（视频开始后5000毫秒）	Integer

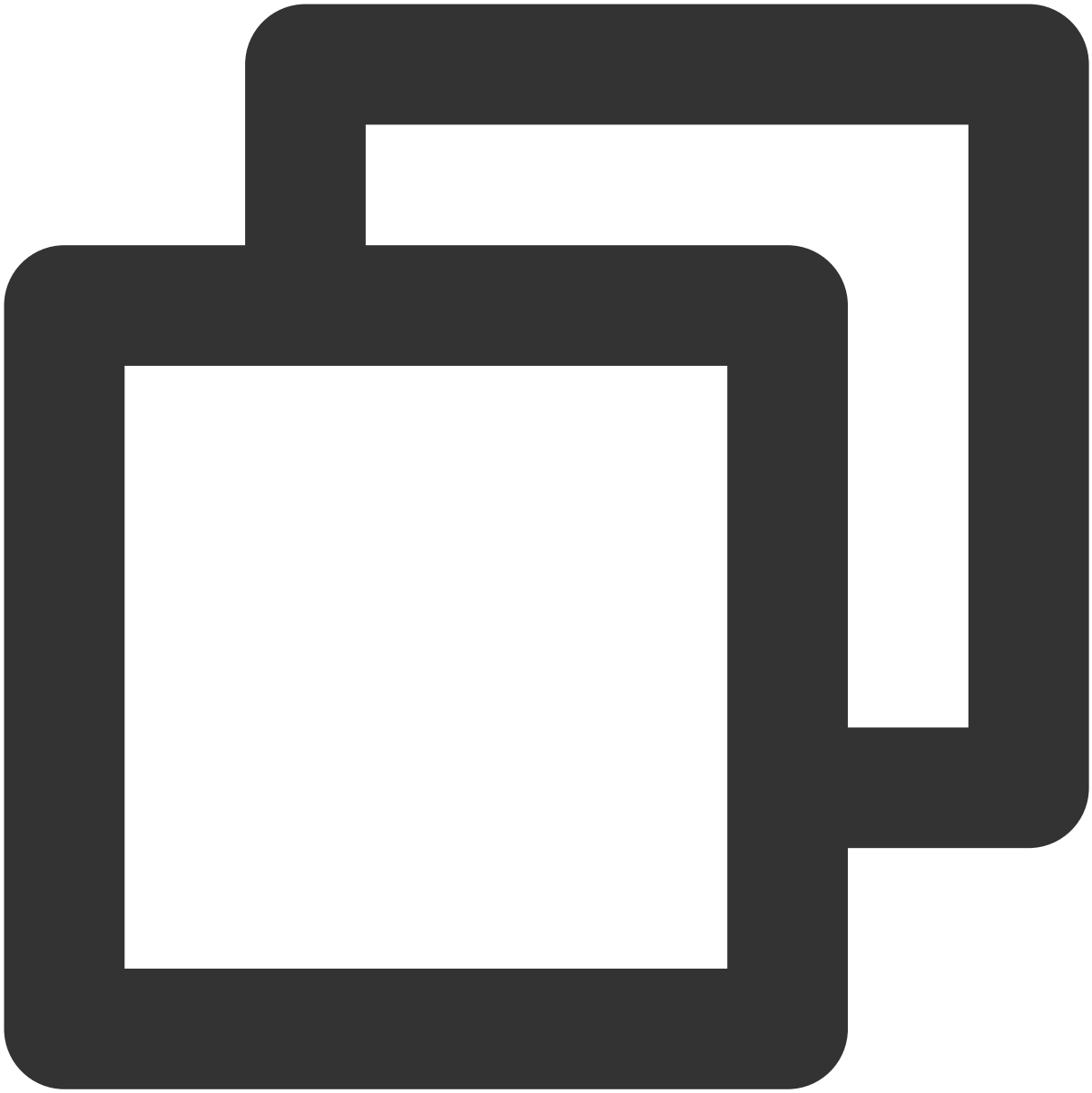
duration	当前视频声音片段的时长，单位毫秒	Integer
porn_info	审核场景为涉黄的审核结果信息	Struct
ads_info	审核场景为广告引导的审核结果信息	Struct
hit_flag	是否命中该审核分类，0表示未命中，1表示命中，2表示疑似	Integer
score	该字段表示审核结果命中审核信息的置信度，取值范围：0（置信度最低）-100（置信度最高），越高代表该内容越有可能属于当前返回审核信息。例如：色情99，则表明该内容非常有可能属于色情内容	Integer
key_words	在当前审核场景下命中的关键词	String

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

完整代码请参见 cos_demo.c 中的 test_ci_video_auditing() 函数。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// 数据万象的访问域名，详情请参见 https://intl.cloud.tencent.com/document/product/1045/3
static char TEST_CI_ENDPOINT[] = "https://ci.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

static void log_get_auditing_result(ci_auditing_job_result_t *result)
{
    int index = 0;
    ci_auditing_snapshot_result_t *snapshot_info;
    ci_auditing_audio_section_result_t *audio_section_info;

    cos_warn_log("nonexist_job_ids: %s", result->nonexist_job_ids.data);
    cos_warn_log("code: %s", result->jobs_detail.code.data);
    cos_warn_log("message: %s", result->jobs_detail.message.data);
    cos_warn_log("state: %s", result->jobs_detail.state.data);
    cos_warn_log("creation_time: %s", result->jobs_detail.creation_time.data);
    cos_warn_log("object: %s", result->jobs_detail.object.data);
    cos_warn_log("snapshot_count: %s", result->jobs_detail.snapshot_count.data);
    cos_warn_log("result: %d", result->jobs_detail.result);

    cos_warn_log("porn_info.hit_flag: %d", result->jobs_detail.porn_info.hit_flag);
    cos_warn_log("porn_info.count: %d", result->jobs_detail.porn_info.count);
    cos_warn_log("ads_info.hit_flag: %d", result->jobs_detail.ads_info.hit_flag);
    cos_warn_log("ads_info.count: %d", result->jobs_detail.ads_info.count);

    cos_list_for_each_entry(ci_auditing_snapshot_result_t, snapshot_info, &result->
        cos_warn_log("snapshot index:%d", ++index);
        cos_warn_log("snapshot_info->url: %s", snapshot_info->url.data);
        cos_warn_log("snapshot_info->snapshot_time: %d", snapshot_info->snapshot_ti
        cos_warn_log("snapshot_info->text: %s", snapshot_info->text.data);

        cos_warn_log("snapshot_info->porn_info.hit_flag: %d", snapshot_info->porn_i
        cos_warn_log("snapshot_info->porn_info.score: %d", snapshot_info->porn_info
        cos_warn_log("snapshot_info->porn_info.label: %s", snapshot_info->porn_info
        cos_warn_log("snapshot_info->porn_info.sub_lable: %s", snapshot_info->porn_
        cos_warn_log("snapshot_info->ads_info.hit_flag: %d", snapshot_info->ads_inf
        cos_warn_log("snapshot_info->ads_info.score: %d", snapshot_info->ads_info.s
        cos_warn_log("snapshot_info->ads_info.label: %s", snapshot_info->ads_info.l
        cos_warn_log("snapshot_info->ads_info.sub_lable: %s", snapshot_info->ads_in
    }
```

```
    index = 0;
    cos_list_for_each_entry(ci_auditing_audio_section_result_t, audio_section_info,
        cos_warn_log("audio_section index:%d", ++index);
        cos_warn_log("audio_section_info->url: %s", audio_section_info->url.data);
        cos_warn_log("audio_section_info->text: %s", audio_section_info->text.data)
        cos_warn_log("audio_section_info->offset_time: %d", audio_section_info->off
        cos_warn_log("audio_section_info->duration: %d", audio_section_info->durati

        cos_warn_log("audio_section_info->porn_info.hit_flag: %d", audio_section_in
        cos_warn_log("audio_section_info->porn_info.score: %d", audio_section_info-
        cos_warn_log("audio_section_info->porn_info.key_words: %s", audio_section_i
        cos_warn_log("audio_section_info->ads_info.hit_flag: %d", audio_section_inf
        cos_warn_log("audio_section_info->ads_info.score: %d", audio_section_info->
        cos_warn_log("audio_section_info->ads_info.key_words: %s", audio_section_in
    }
}

void test_ci_get_video_auditing()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers;
    ci_video_auditing_job_options_t *job_options;
    ci_video_auditing_job_result_t *job_result;
    ci_auditing_job_result_t *auditing_result;

    // 基本配置
    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    options->config = cos_config_create(options->pool);
    cos_str_set(&options->config->endpoint, TEST_CI_ENDPOINT);     // https://ci.<R
    cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&options->config->appid, TEST_APPID);
    options->config->is_cname = is_cname;
    options->ctl = cos_http_controller_create(options->pool, 0);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    // 替换为您的配置信息，可参见文档 https://intl.cloud.tencent.com/document/product/43
    job_options = ci_video_auditing_job_options_create(p);
    cos_str_set(&job_options->input_object, "test.mp4");
    cos_str_set(&job_options->job_conf.detect_type, "Porn,Ads");
    cos_str_set(&job_options->job_conf.callback_version, "Detail");
```

```
    job_options->job_conf.detect_content = 1;
    cos_str_set(&job_options->job_conf.snapshot.mode, "Interval");
    job_options->job_conf.snapshot.time_interval = 1.5;
    job_options->job_conf.snapshot.count = 10;

    // 提交一个视频审核任务
    s = ci_create_video_auditing_job(options, &bucket, job_options, NULL, &resp_hea
    log_status(s);

    // 等待视频审核任务完成，此处可修改您的等待时间
    sleep(300);

    // 获取审核任务结果
    s = ci_get_auditing_job(options, &bucket, &job_result->jobs_detail.job_id, NULL
    log_status(s);
    if (s->code == 200) {
        log_get_auditing_result(auditing_result);
    }

    // 销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_ci_get_video_auditing();

    cos_http_io_deinitialize();

    return 0;
}
```

图片处理
图片持久化处理

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于图片持久化处理的 API 概览以及 SDK 示例代码。

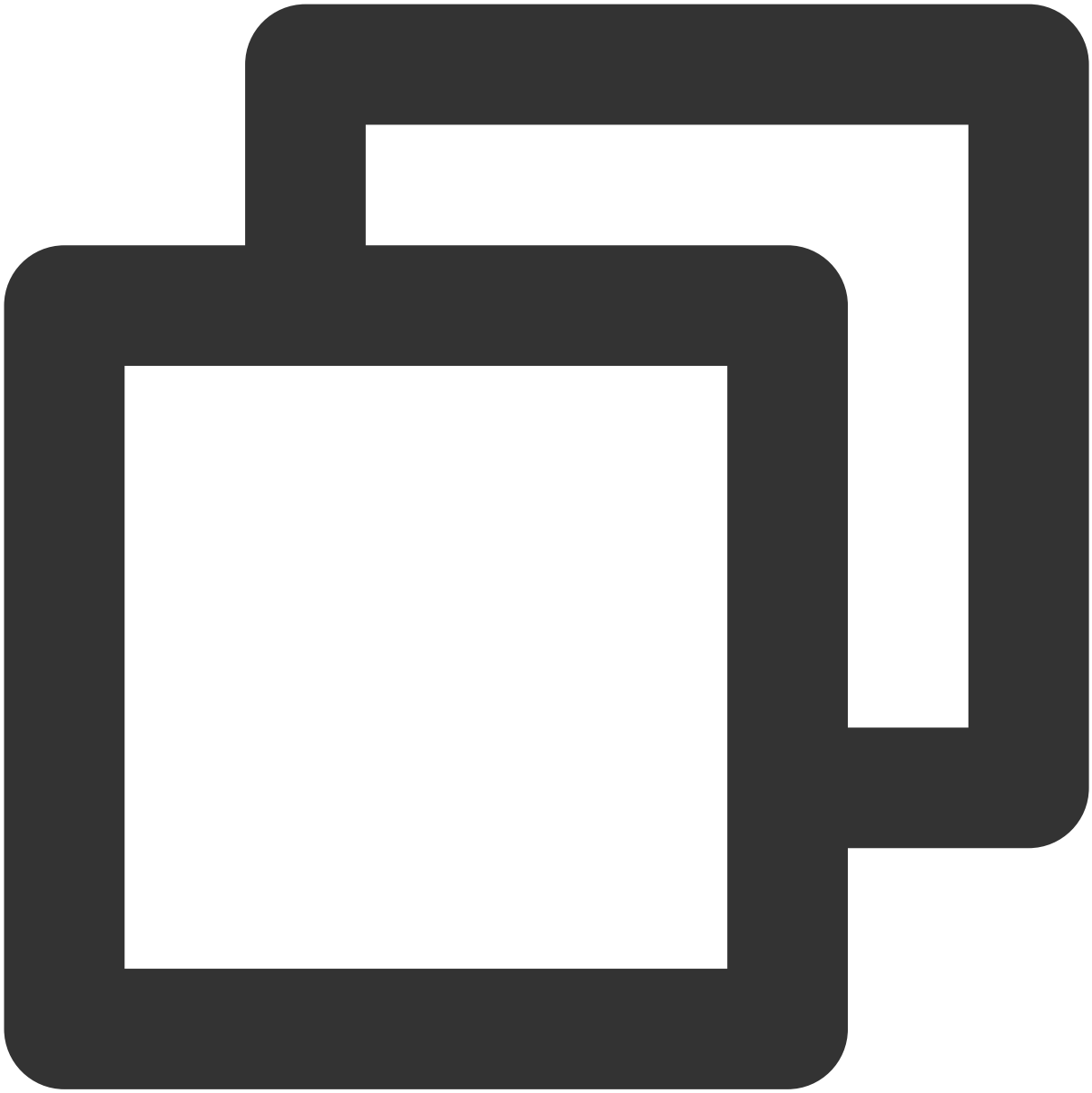
API	说明
图片持久化处理	对象存储提供的上传时处理功能可以帮助使用者在上传时实现图片处理。此外能够对已存储在 COS 的图片进行相应处理操作，并将结果存入到对象存储（Cloud Object Storage，COS）

上传时处理

功能说明

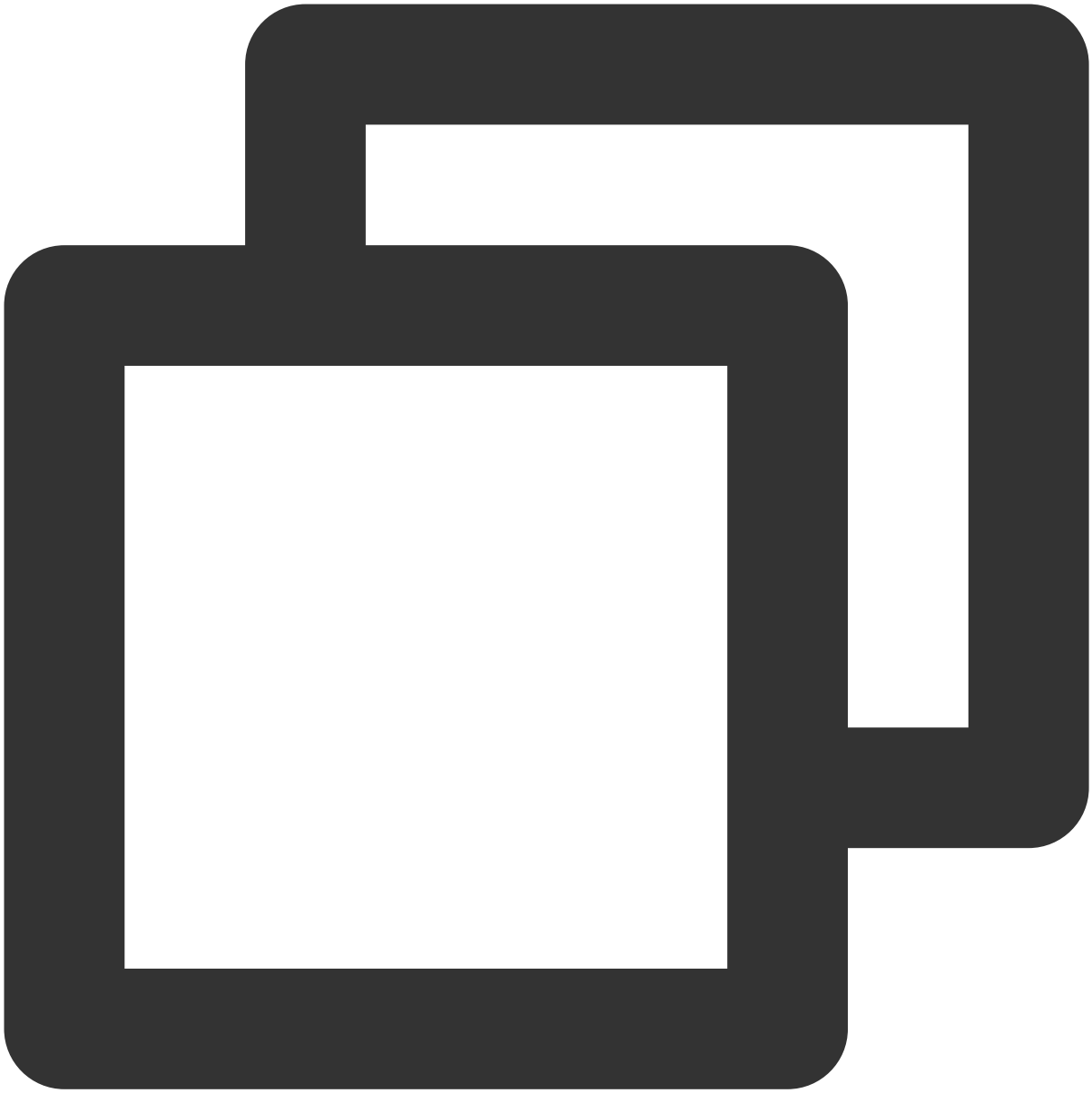
COS 提供的上传时处理功能可以帮助使用者在上传时实现图片处理。您只需要在请求包头部中加入 Pic-Operations 项并设置好相应参数，即可在图片上传时实现相应的图片处理，并可将原图和处理结果存入到 COS。目前支持20M以内文件处理。

方法原型



```
cos_status_t *ci_put_object_from_file(const cos_request_options_t *options,
                                      const cos_string_t *bucket,
                                      const cos_string_t *object,
                                      const cos_string_t *filename,
                                      cos_table_t *headers,
                                      cos_table_t **resp_headers,
                                      ci_operation_result_t **results)
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_image_process()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t file;
    cos_table_t *headers = NULL;
    cos_table_t *resp_headers;
    ci_operation_result_t *results = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //上传时处理
```

```
    headers = cos_table_make(p, 1);
    apr_table_addn(headers, "pic-operations", "{\\"is_pic_info\\":1,\\"rules\\":[{\
    cos_str_set(&file, "test.jpg");
    cos_str_set(&object, "test.jpg");
    s = ci_put_object_from_file(options, &bucket, &object, &file, headers, &resp_he
    if (!cos_status_is_ok(s)) {
        printf("put object failed: %s\\n", s->req_id);
    }
    printf("origin key: %s\\n", results->origin.key.data);
    printf("process key: %s\\n", results->object.key.data);

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_image_process();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String

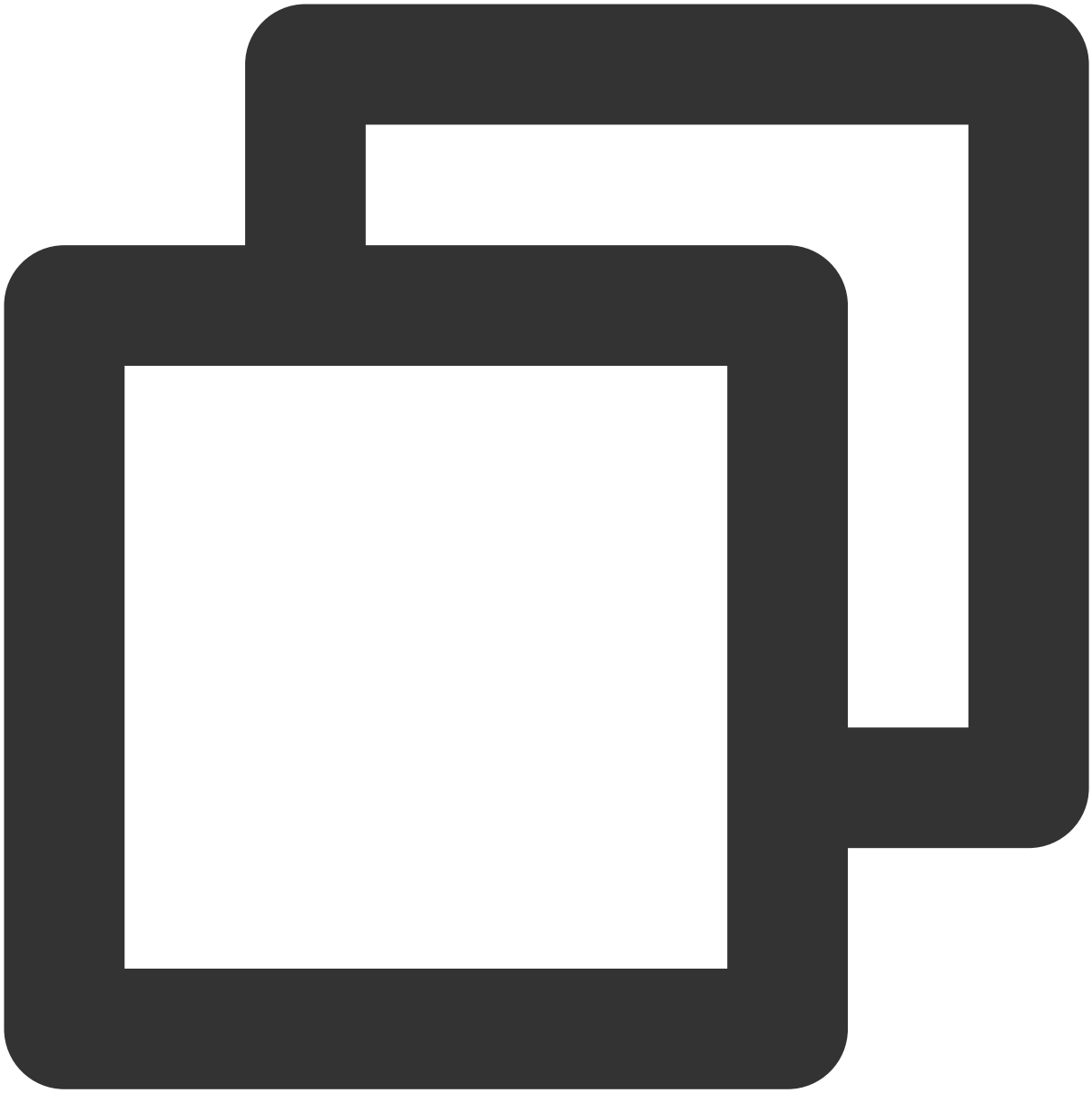
object	Object 名称	String
filename	Object 本地保存文件名称	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct
results	云上数据处理返回信息	Struct

云上数据处理

功能说明

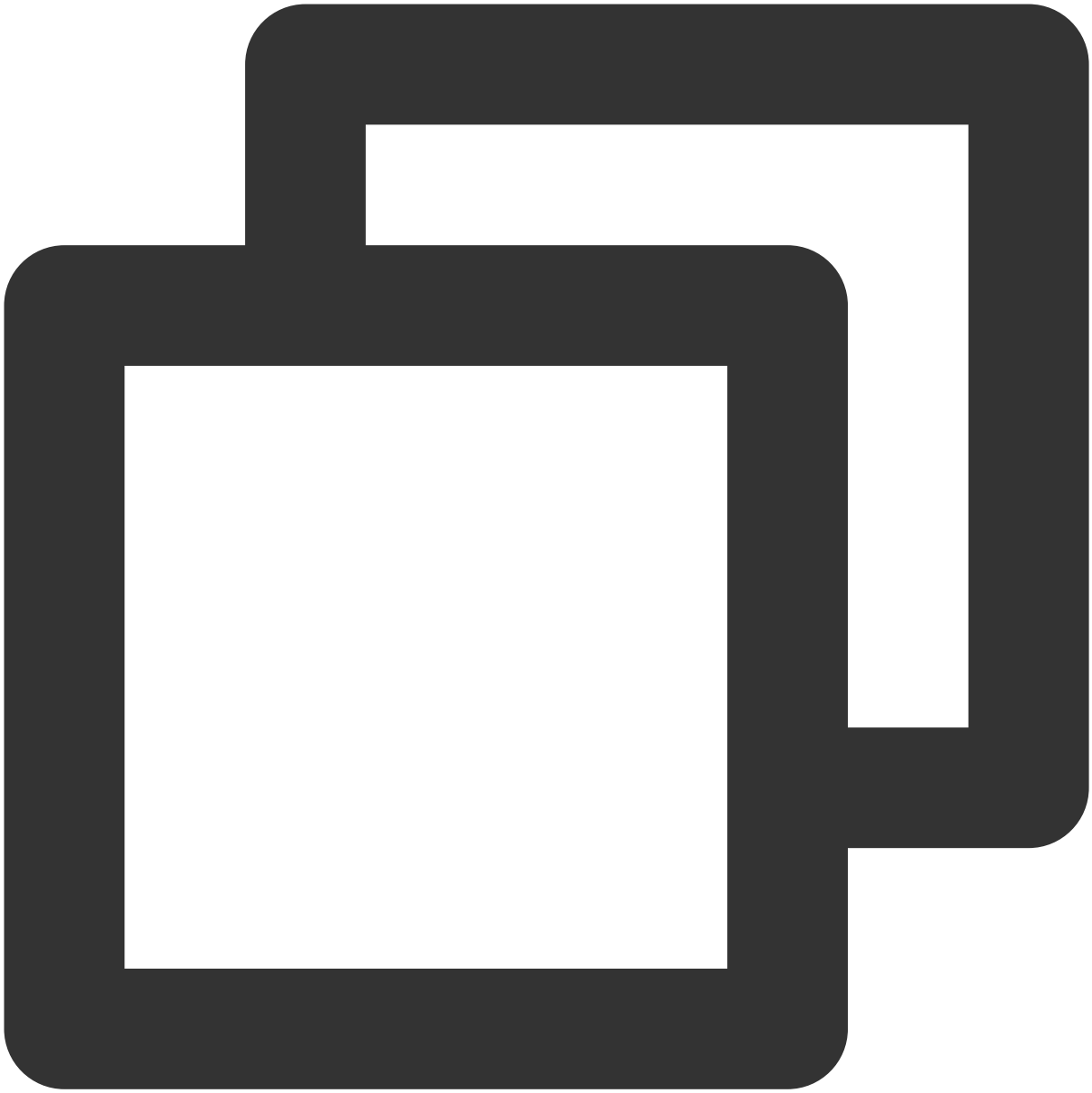
该接口能够对已存储在 COS 的图片进行相应处理操作，并将结果存入到 COS。

方法原型



```
cos_status_t *ci_image_process(const cos_request_options_t *options,
                               const cos_string_t *bucket,
                               const cos_string_t *object,
                               cos_table_t *headers,
                               cos_table_t **resp_headers,
                               ci_operation_result_t *results);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_ci_image_process()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_table_t *headers = NULL;
    cos_table_t *resp_headers;
    ci_operation_result_t *results = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    cos_str_set(&object, "test.jpg");
    headers = cos_table_make(p, 1);
```

```
    apr_table_addn(headers, "pic-operations", "{\\"is_pic_info\\":1,\\"rules\\":[{\

    s = ci_image_process(options, &bucket, &object, headers, &resp_headers, &result
    if (!cos_status_is_ok(s)) {
        printf("ci image process fail, req_id:%s\\n", s->req_id);
    }
    printf("origin key: %s\\n", results->origin.key.data);
    printf("process key: %s\\n", results->object.key.data);
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_ci_image_process();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，Bucket 的命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	string
object	Object 名称	string
headers	COS 请求附加头域，需要自行添加 Pic-Operations 万象处理头部	Struct

resp_headers	返回 HTTP 响应消息的头域	Struct
results	云上数据处理返回信息	Struct

版权所有：腾讯云计算（北京）有限责任公司

基础图片处理

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于基础图片处理的 API 概览以及 SDK 示例代码。

API	操作描述
缩放	对图片进行缩小或放大
裁剪	对图片进行裁剪，包括普通裁剪、缩放裁剪、内切圆裁剪、圆角裁剪和人脸智能裁剪
旋转	对图片进行旋转，包括普通旋转和自适应旋转
格式转换	对图片进行格式转换、gif 格式优化、渐进显示
质量变换	对图片质量进行调节
高斯模糊	对图片进行模糊处理
锐化	对图片进行锐化
图片水印	对图片进行水印处理
文字水印	对图片进行实时文字水印处理
获取图片基本信息	查询图片基本信息，包括格式、长、宽等
获取图片 EXIF	查询 EXIF 信息
获取图片主色调	查询图片主色调信息
去除元信息	去除图片元信息，包括 exif 信息
快速缩略模板	通过图片处理模板，生成相应的缩略图
管道操作符	实现对图片按顺序进行多种处理

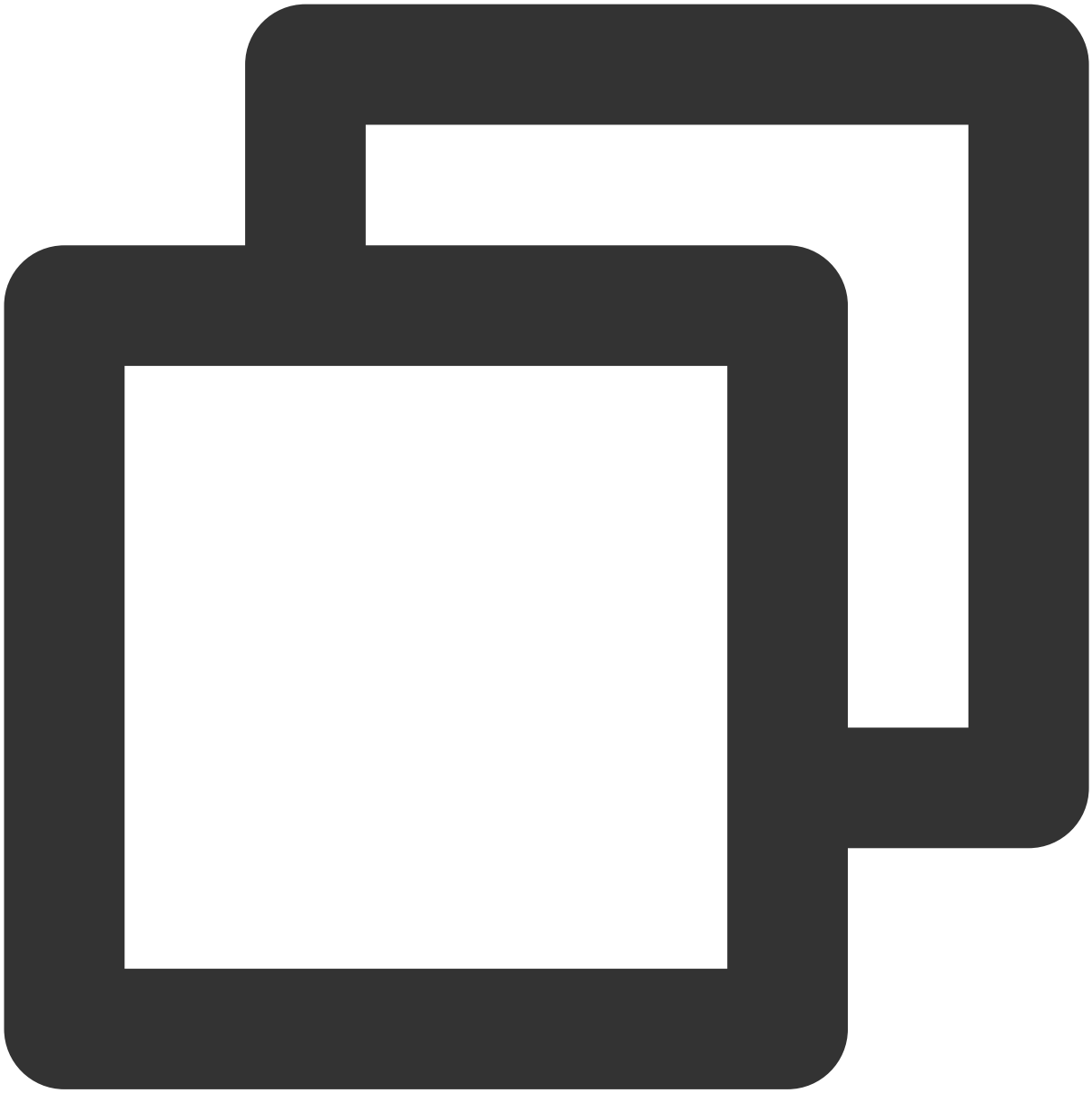
缩放

功能说明

腾讯云数据万象通过 **imageMogr2** 接口提供图片缩放功能。

版权所有：腾讯云计算（北京）有限责任公司

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
```

```
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_image_scale()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t file;
    cos_table_t *resp_headers;
    cos_table_t *params = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
```

```
    params = cos_table_make(p, 1);
    apr_table_addn(params, "imageMogr2/thumbnail/!50p", "");
    cos_str_set(&file, "test.jpg");
    cos_str_set(&object, "test.jpg");
    s = cos_get_object_to_file(options, &bucket, &object, NULL, params, &file, &res
    log_status(s);
    if (!cos_status_is_ok(s)) {
        printf("cos_get_object_to_file fail, req_id:%s\\n", s->req_id);
    }
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_image_scale();

    cos_http_io_deinitialize();

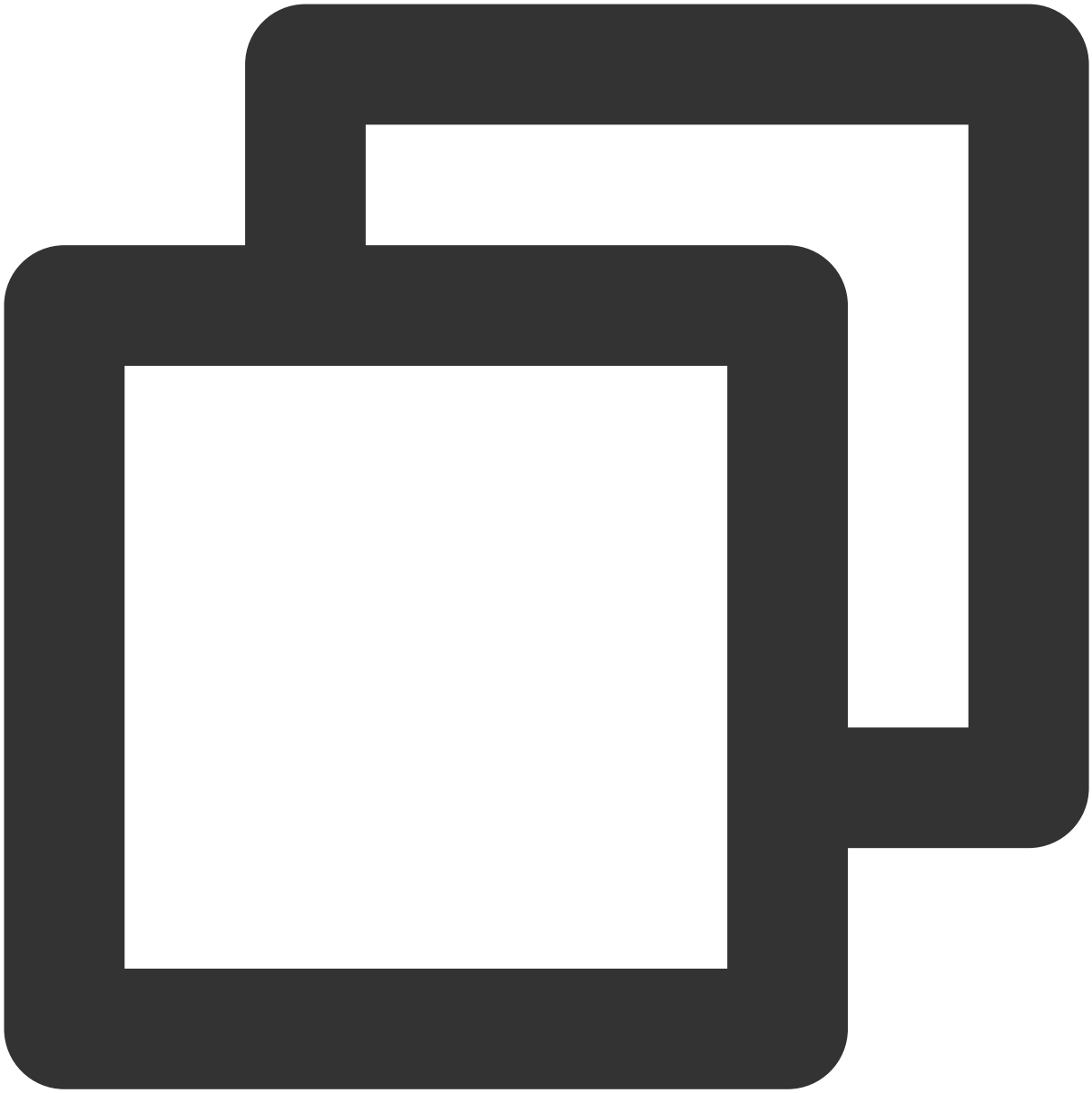
    return 0;
}
```

裁剪

功能说明

腾讯云数据万象通过 **imageMogr2** 接口提供裁剪功能，包括普通裁剪、缩放裁剪、内切圆裁剪、圆角裁剪和人脸智能裁剪。

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_image_cut()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t file;
    cos_table_t *resp_headers;
    cos_table_t *params = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    params = cos_table_make(p, 1);
    apr_table_addn(params, "imageMogr2/cut/600x600x100x10", "");
```

```
    cos_str_set(&file, "test.jpg");
    cos_str_set(&object, "test.jpg");
    s = cos_get_object_to_file(options, &bucket, &object, NULL, params, &file, &res
    log_status(s);
    if (!cos_status_is_ok(s)) {
        printf("cos_get_object_to_file fail, req_id:%s\\n", s->req_id);
    }
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_image_cut();

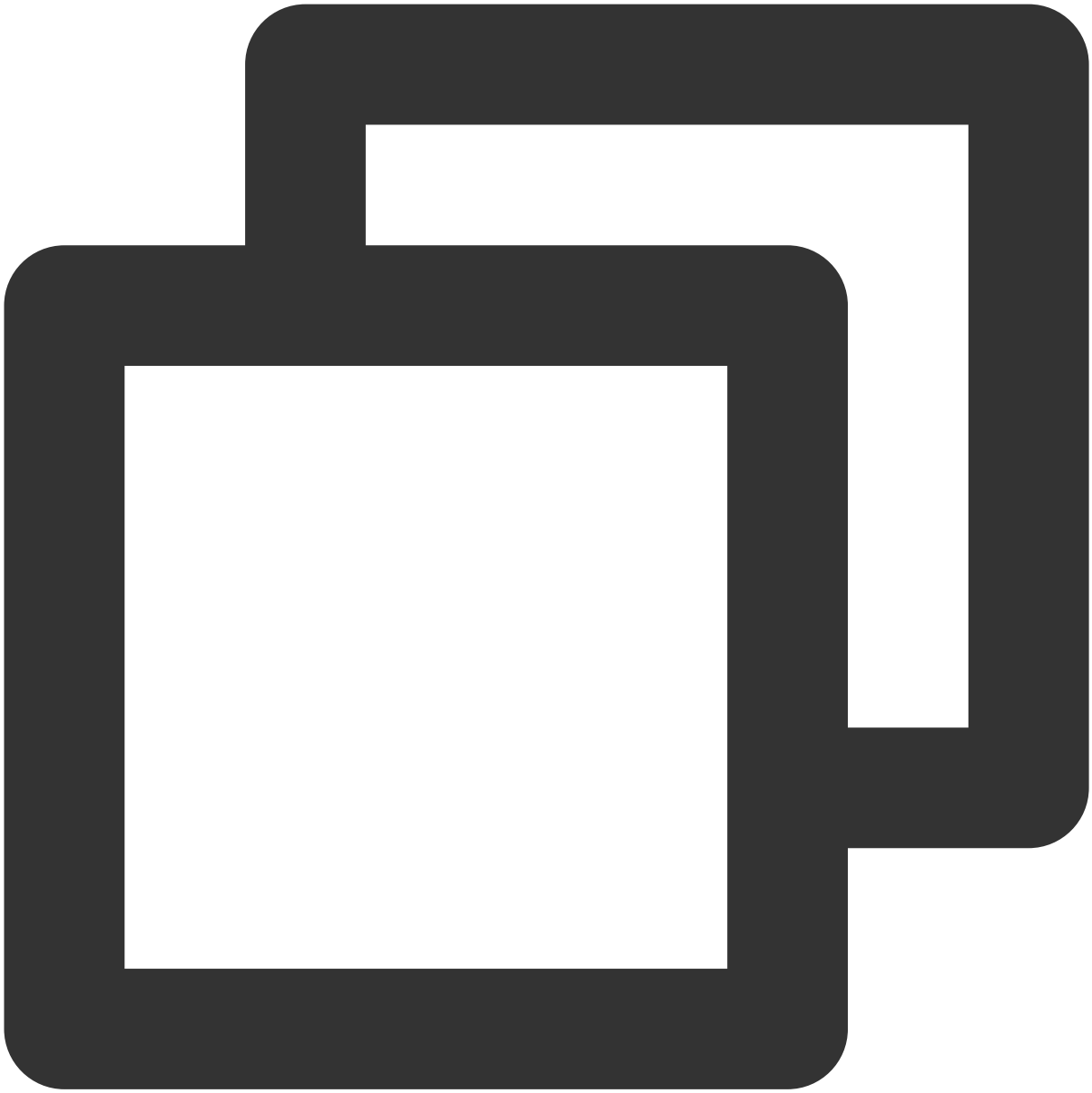
    cos_http_io_deinitialize();

    return 0;
}
```

其他基础图片处理

功能说明

基础图片处理统一使用上述方法完成，其他基础图片处理只需要修改 operation 参数的值即可。例如，如下样例：将图片顺时针旋转90度。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_image_rotate()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t file;
    cos_table_t *resp_headers;
    cos_table_t *params = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    params = cos_table_make(p, 1);
    apr_table_addn(params, "imageMogr2/rotate/90", "");
```

```
    cos_str_set(&file, "test.jpg");
    cos_str_set(&object, "test.jpg");
    s = cos_get_object_to_file(options, &bucket, &object, NULL, params, &file, &res
    log_status(s);
    if (!cos_status_is_ok(s)) {
        printf("cos_get_object_to_file fail, req_id:%s\\n", s->req_id);
    }
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_image_rotate();

    cos_http_io_deinitialize();

    return 0;
}
```

图片高级压缩

最近更新时间：2024-01-04 17:06:08

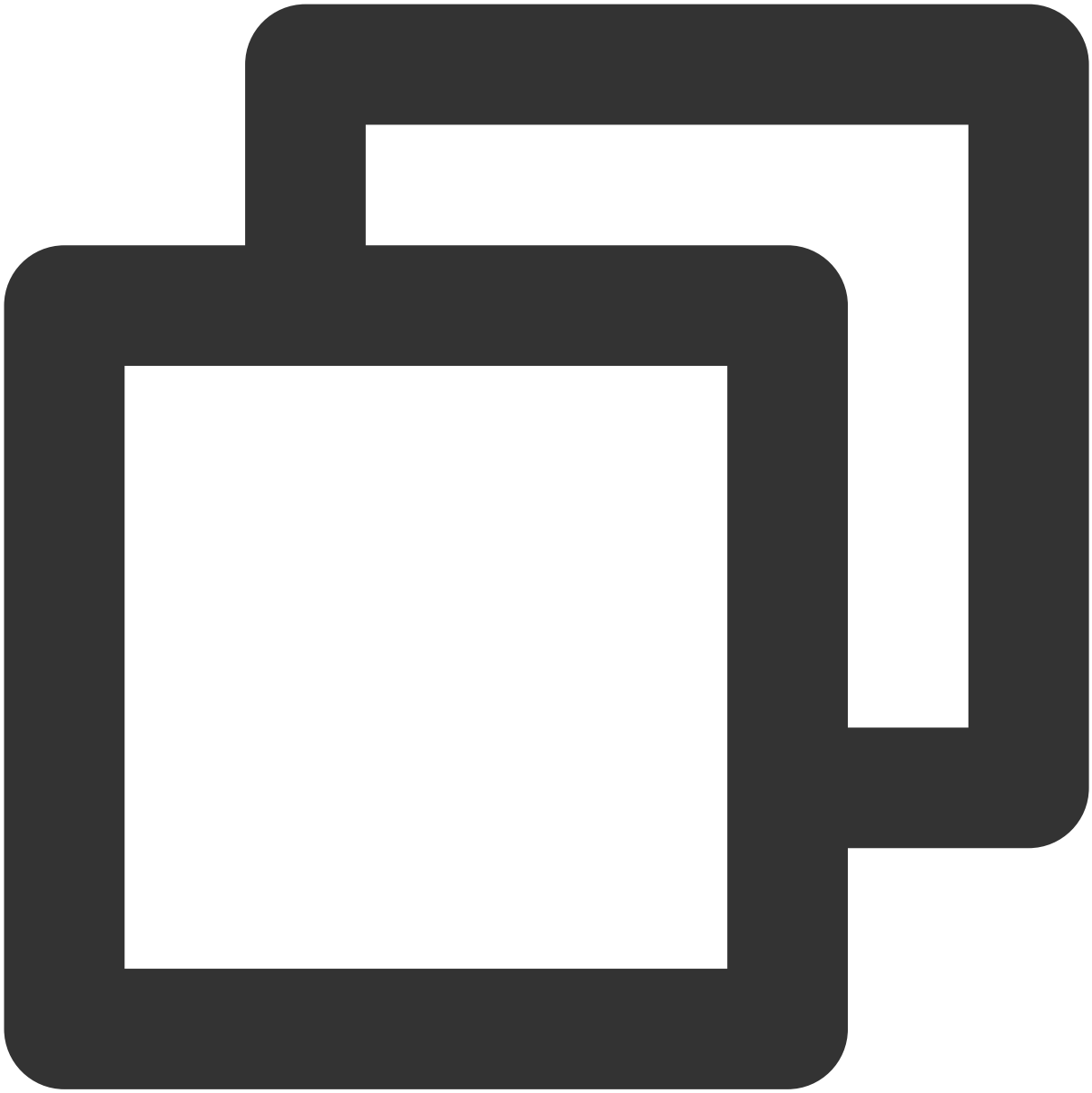
简介

本文档提供关于图片高级压缩的 API 概览以及 SDK 示例代码。

API	操作描述
图片高级压缩	图片高级压缩可以更加高效地将图片转换为 TPG 或 HEIF 等高压缩比格式，有效降低图片传输链路及加载耗时，降低带宽及流量成本

方法原型

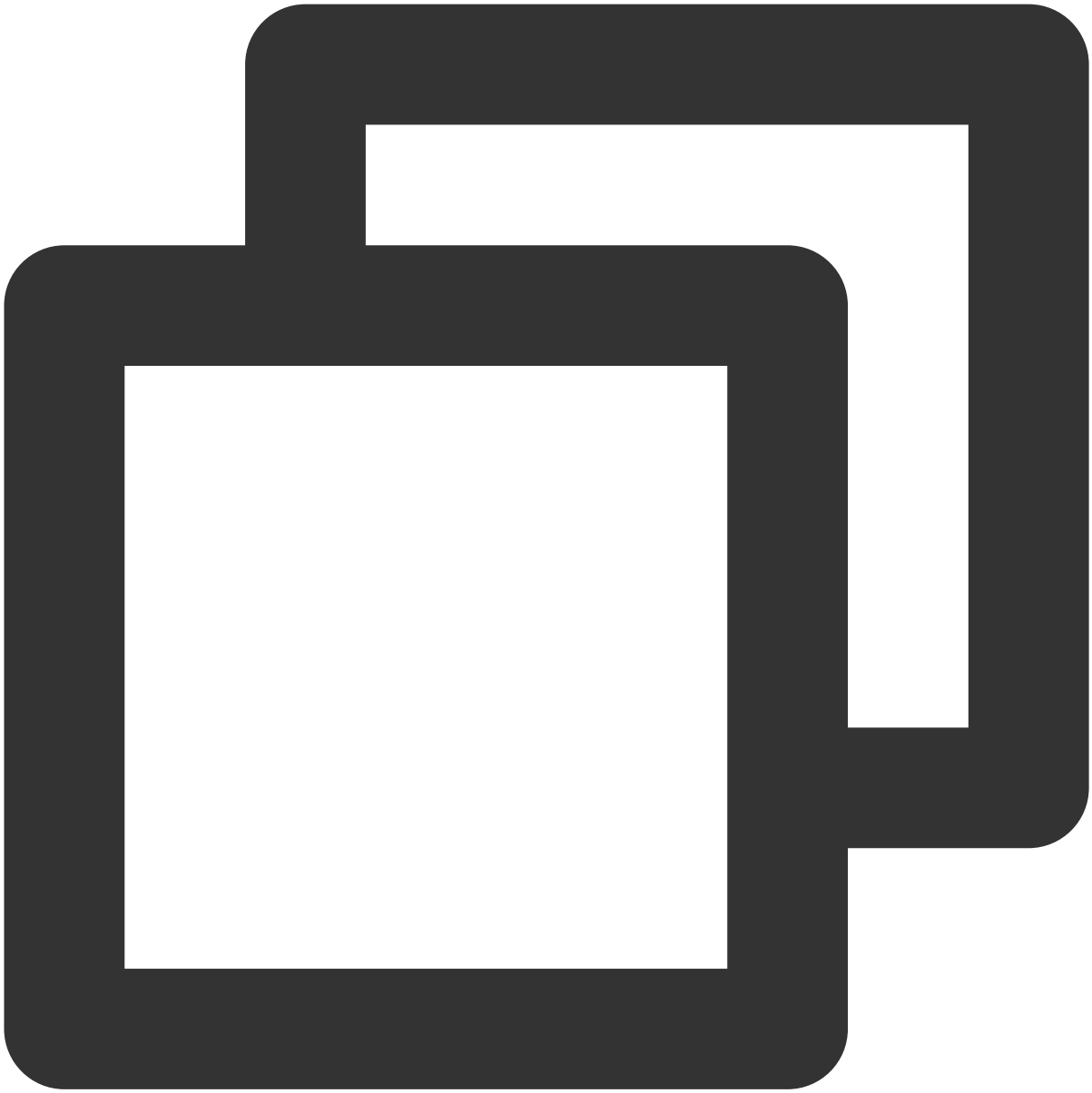
通过下载对象方法实现，用户只需要添加请求参数，请求参数格式如下，关于下载对象详细说明请参见 下载对象。



```
imageMogr2/format/<Format>
```

参数	含义
format	压缩格式，目标缩略图的图片格式为 TPG 或 HEIF。

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_image_compress()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t file;
    cos_table_t *resp_headers;
    cos_table_t *params = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    params = cos_table_make(p, 1);
    apr_table_addn(params, "imageMogr2/format/tpg", "");
```

```
    cos_str_set(&object, "test.jpg");
    cos_str_set(&file, "test.tpg");
    s = cos_get_object_to_file(options, &bucket, &object, NULL, params, &file, &res
    log_status(s);
    if (!cos_status_is_ok(s)) {
        printf("cos_get_object_to_file fail, req_id:%s\\n", s->req_id);
    }

    params = cos_table_make(p, 1);
    apr_table_addn(params, "imageMogr2/format/heif", "");
    cos_str_set(&file, "test.heif");
    s = cos_get_object_to_file(options, &bucket, &object, NULL, params, &file, &res
    log_status(s);
    if (!cos_status_is_ok(s)) {
        printf("cos_get_object_to_file fail, req_id:%s\\n", s->req_id);
    }

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_image_compress();

    cos_http_io_deinitialize();

    return 0;
}
```

访问管理
自定义域名

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于自定义域名的 API 概览以及 SDK 示例代码。

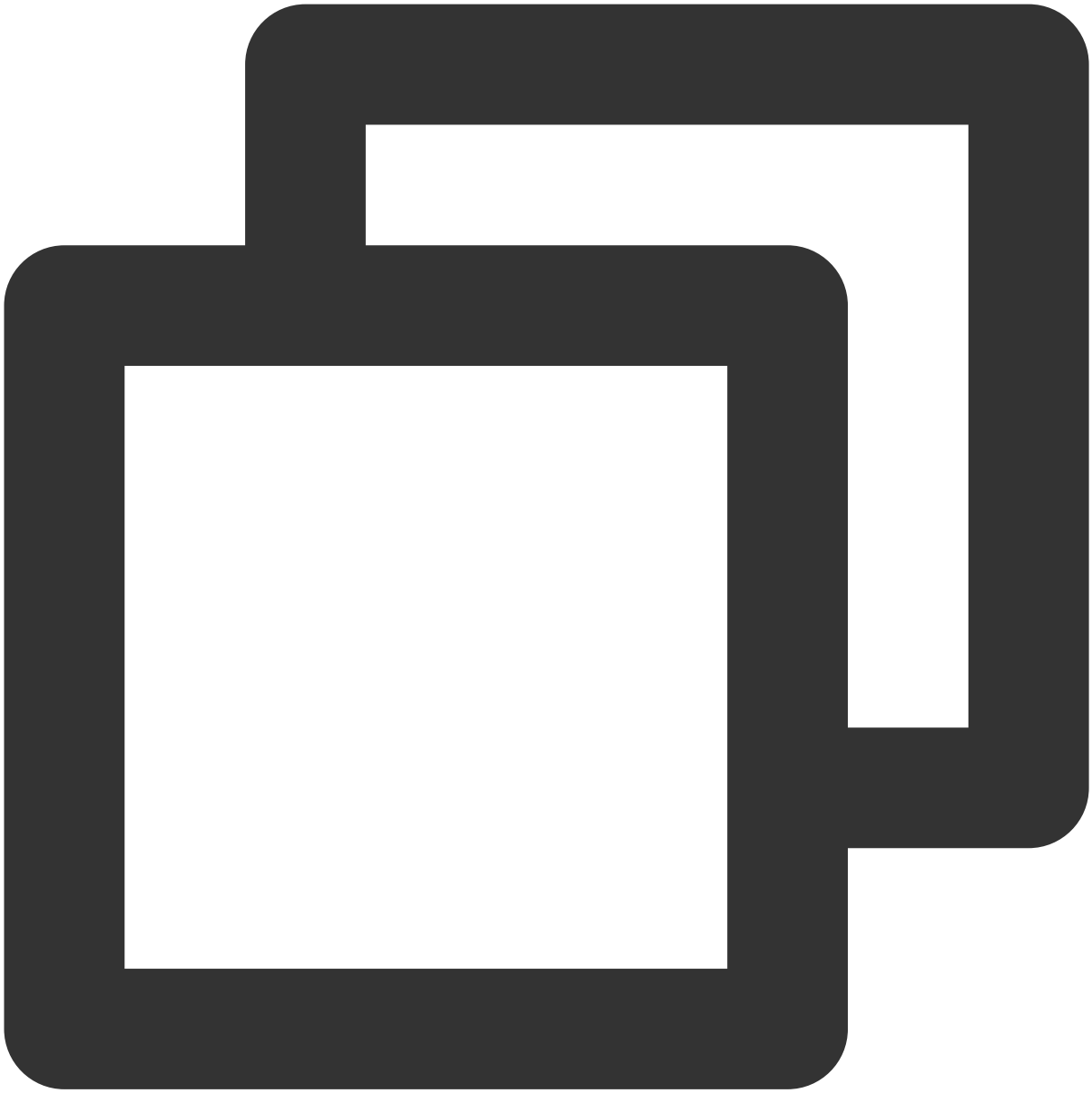
API	操作名	操作描述
PUT Bucket domain	设置自定义域名	设置存储桶的自定义域名信息
GET Bucket domain	查询自定义域名	查询存储桶的自定义域名信息

设置自定义域名

功能说明

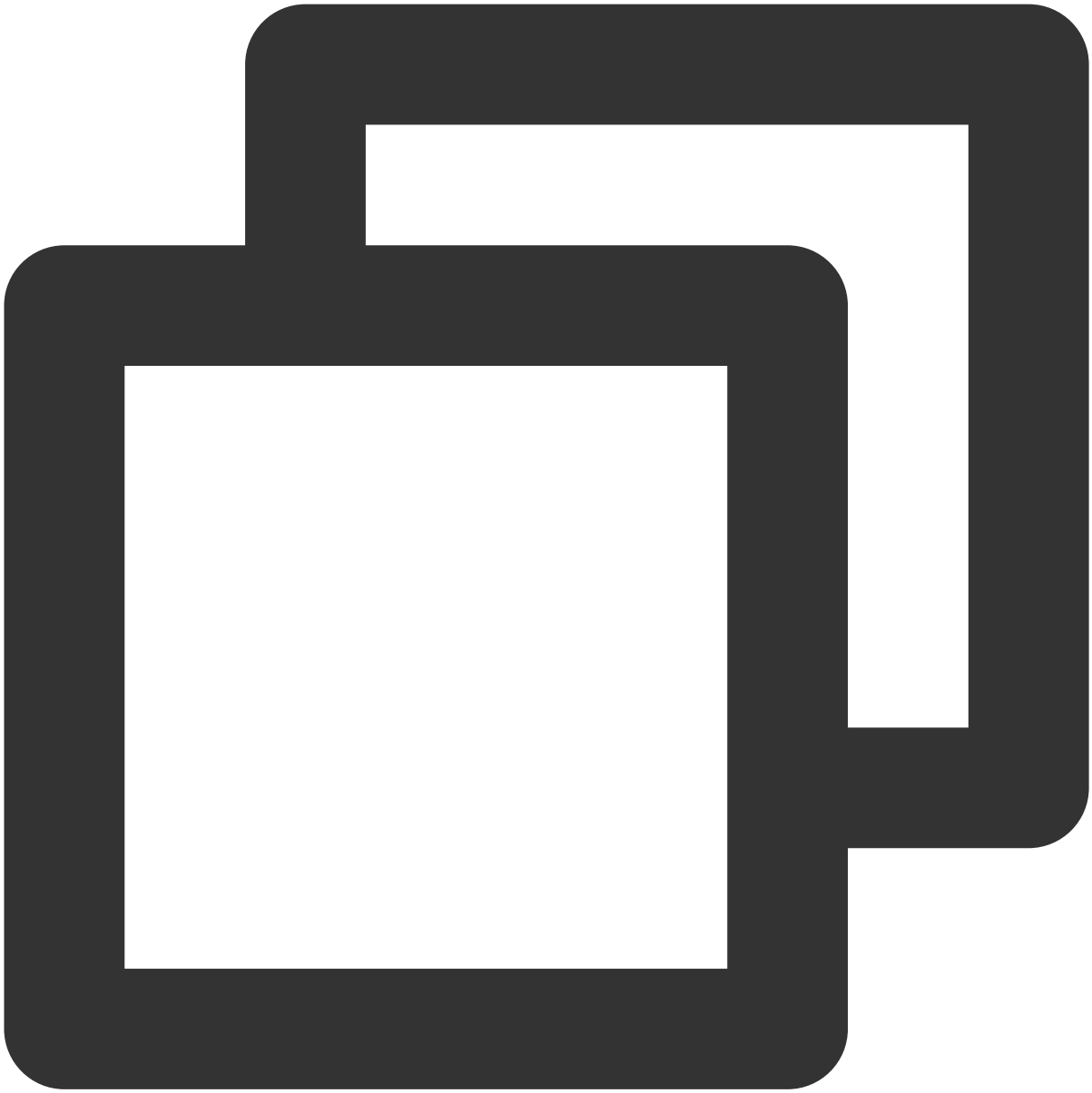
PUT Bucket domain 用于为存储桶配置自定义域名。

方法原型



```
cos_status_t *cos_put_bucket_domain(const cos_request_options_t *options,
                                    const cos_string_t *bucket,
                                    cos_domain_params_t *domain_params,
                                    cos_table_t **resp_header);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_bucket_domain()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_domain_params_t  *domain_params = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //创建 domain 参数
    domain_params = cos_create_domain_params(options->pool);
```

```
    cos_str_set(&domain_params->status, "ENABLED");
    cos_str_set(&domain_params->name, "www.exmaple.com");
    cos_str_set(&domain_params->type, "REST");
    cos_str_set(&domain_params->forced_replacement, "CNAME");

    status = cos_put_bucket_domain(options, &bucket, domain_params, &resp_headers);
    log_status(status);

    // 销毁内存池
    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_bucket_domain();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	设置自定义域名的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
domain_params	存储桶自定义域名配置信息	Struct
status	域名上线/下线状态，有效值 ENABLED/DISABLED	String

name	用户的自定义域名，有效值：字母、数字、点	String
type	绑定的源站类型，有效值 REST/WEBSITE	String
forced_replacement	替换已存在的配置，有效值 CNAME/TXT。填写则强制校验域名所有权后，再下发配置	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

返回错误码说明

该请求可能会发生的一些常见的特殊错误如下：

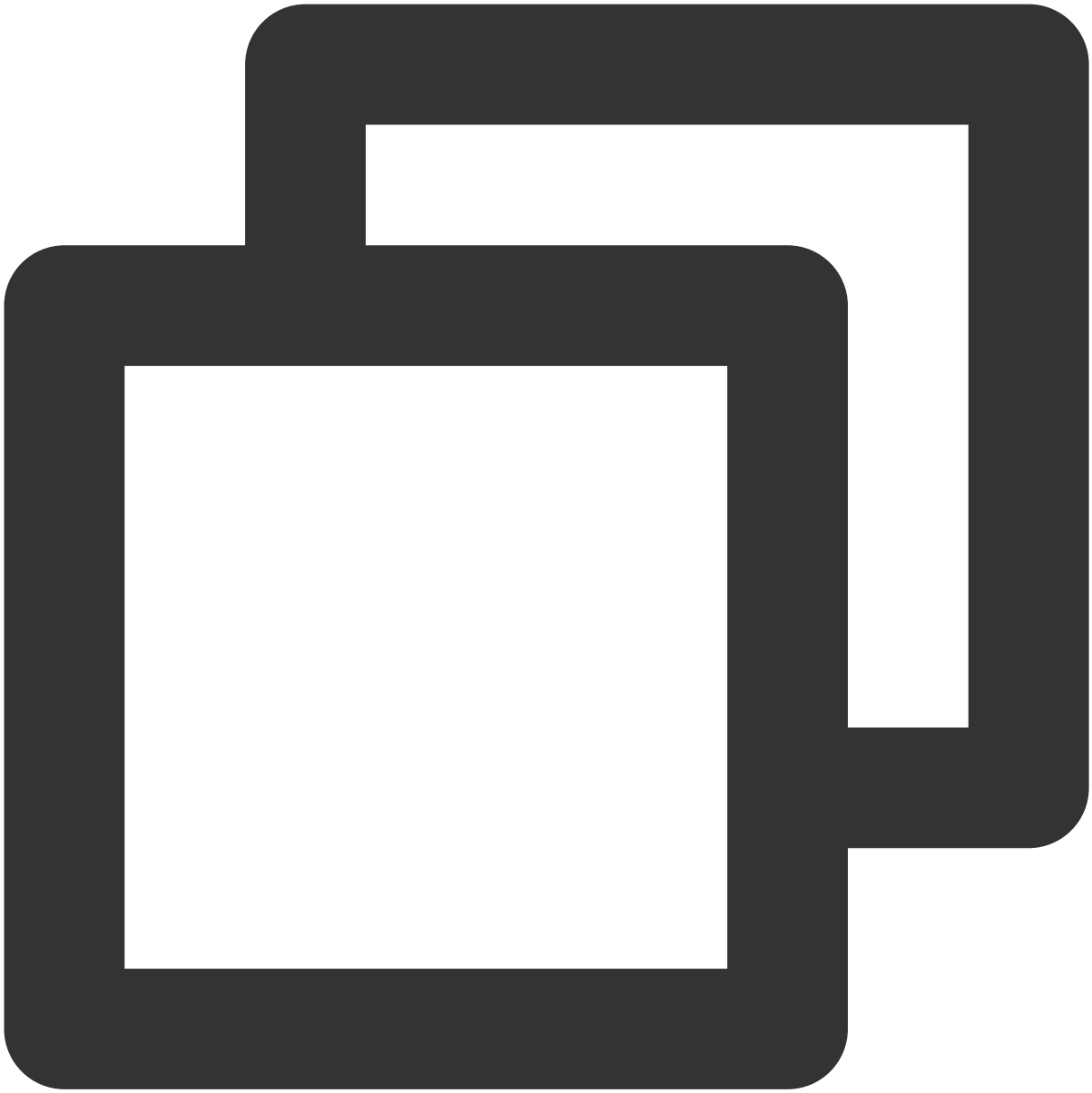
状态码	说明
HTTP 409 Conflict	该域名记录已存在，且请求中没有设置强制覆盖。或者该域名记录不存在，且请求中设置了强制覆盖
HTTP 451 Unavailable For Legal Reasons	该域名是中国境内域名，并且没有备案

查询自定义域名

功能说明

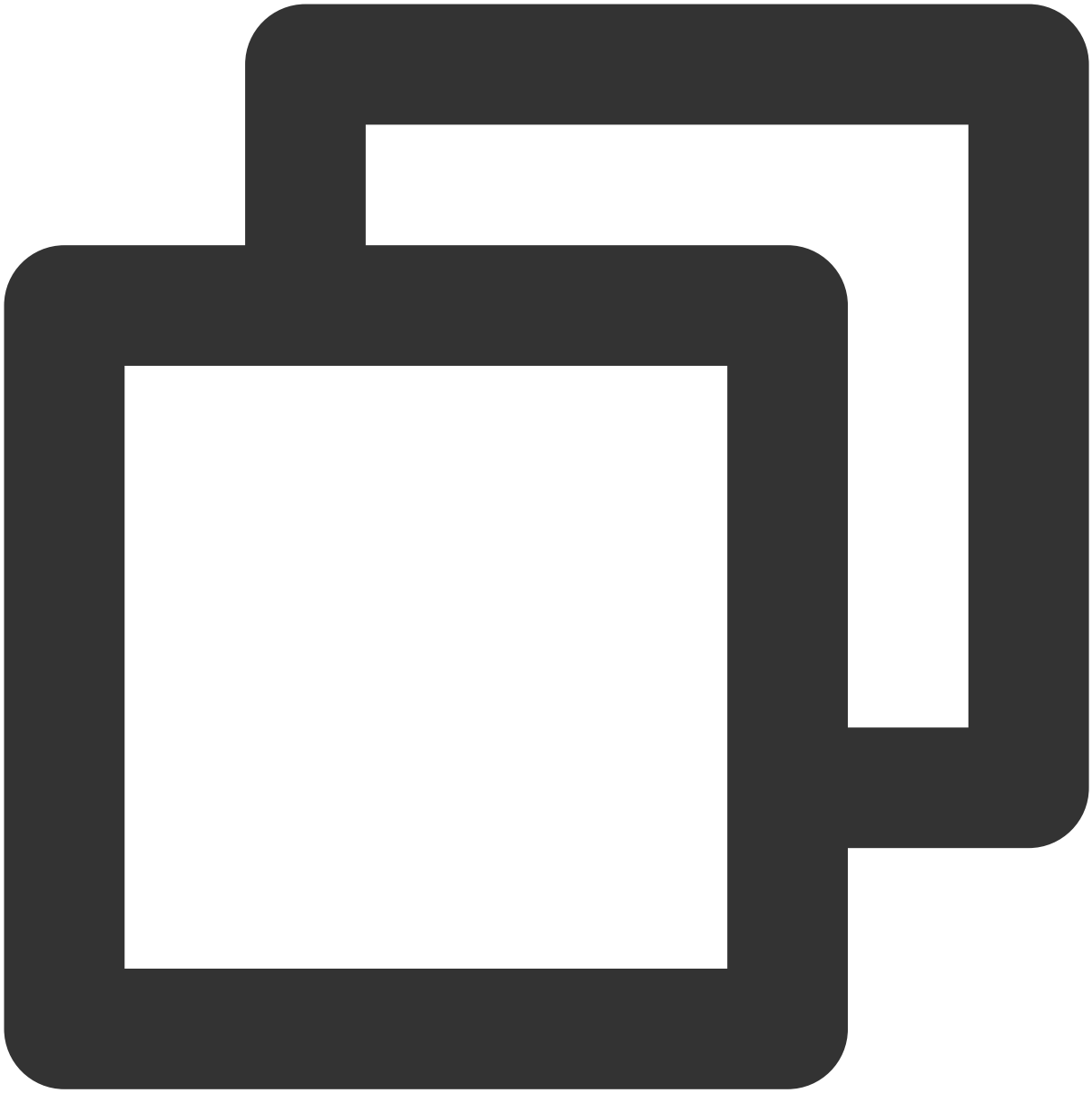
GET Bucket domain 用于查询存储桶的自定义域名信息。

方法原型



```
cos_status_t *cos_get_bucket_domain(const cos_request_options_t *options,
                                    const cos_string_t *bucket,
                                    cos_domain_params_t *domain_params,
                                    cos_table_t **resp_header);
```

请求示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_get_bucket_domain()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_domain_params_t  *domain_result = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    // 查询自定义域名配置
    domain_result = cos_create_domain_params(options->pool);
```

```
    status = cos_get_bucket_domain(options, &bucket, domain_result, &resp_headers);
    log_status(status);
    if (!cos_status_is_ok(status)) {
        cos_pool_destroy(pool);
        return;
    }

    //查看结果
    char *line = NULL;
    line = apr_psprintf(options->pool, "%.*s\\n", domain_result->status.len, domain
    printf("status: %s", line);
    line = apr_psprintf(options->pool, "%.*s\\n", domain_result->name.len, domain_r
    printf("name: %s", line);
    line = apr_psprintf(options->pool, "%.*s\\n", domain_result->type.len, domain_r
    printf("type: %s", line);
    line = apr_psprintf(options->pool, "%.*s\\n", domain_result->forced_replacement
    printf("forced_replacement: %s", line);

    // 销毁内存池
    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_get_bucket_domain();

    cos_http_io_deinitialize();

    return 0;
}
```

参数说明

参数名称	描述	类型
options	COS 请求选项	Struct
bucket	查询自定义域名的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
domain_params	存储桶自定义域名配置信息	Struct
status	域名上线/下线状态，有效值 ENABLED/DISABLED	String
name	用户的自定义域名，有效值：字母、数字、点	String
type	绑定的源站类型，有效值 REST/WEBSITE	String
forced_replacement	替换已存在的配置，有效值 CNAME/TXT。填写则强制校验域名所有权后，再下发配置	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

访问控制

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于存储桶、对象的访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

存储桶 ACL

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶的访问权限控制列表（ACL）
GET Bucket acl	查询存储桶 ACL	获取指定存储桶的访问权限控制列表（ACL）

对象 ACL

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置 Bucket 中某个 Object （文件/对象）的 ACL
GET Object acl	查询对象 ACL	查询 Object（文件/对象）的 ACL

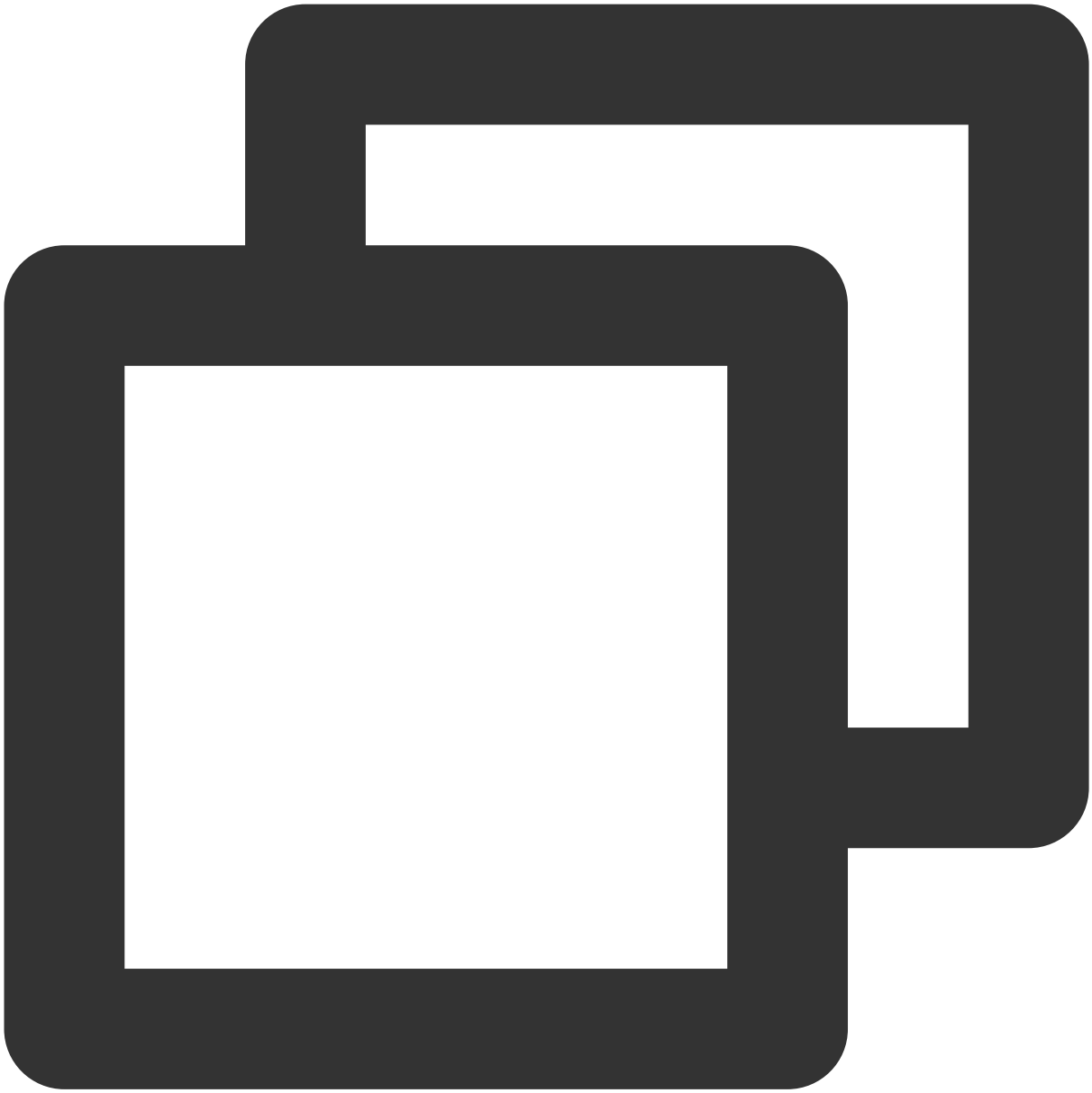
存储桶 ACL

设置存储桶 ACL

功能说明

设置指定存储桶的访问权限控制列表（ACL）。

方法原型



```
cos_status_t *cos_put_bucket_acl(const cos_request_options_t *options,
                                 const cos_string_t *bucket,
                                 cos_acl_e cos_acl,
                                 const cos_string_t *grant_read,
                                 const cos_string_t *grant_write,
                                 const cos_string_t *grant_full_ctrl,
                                 cos_table_t **resp_headers);
```

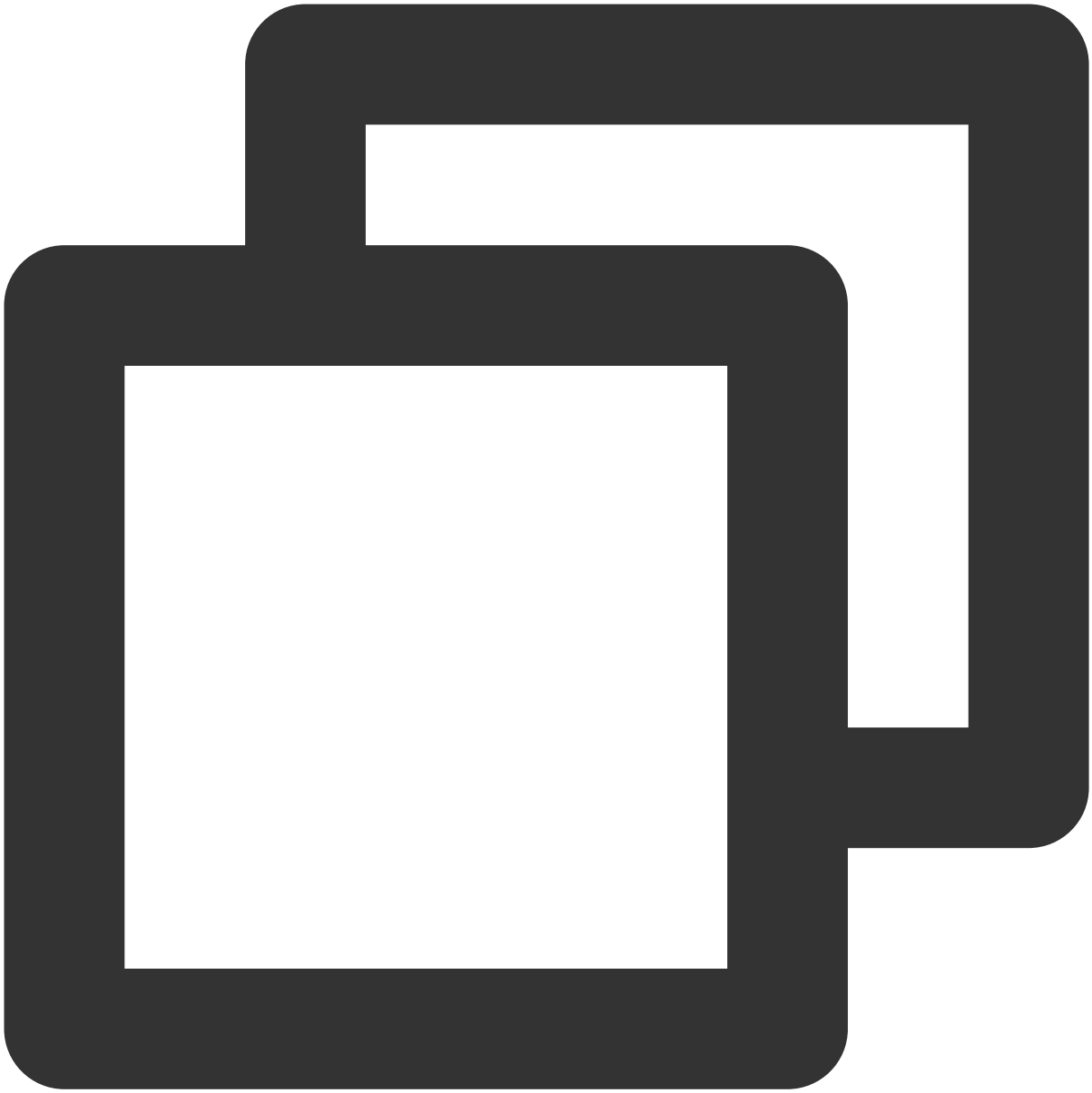
参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
cos_acl	允许用户自定义权限。 有效值：COS_ACL_PRIVATE(0)，COS_ACL_PUBLIC_READ(1)，COS_ACL_PUBLIC_READ_WRITE(2) 默认值：COS_ACL_PRIVATE(0)	Enum
grant_read	读权限授予者	String
grant_write	写权限授予者	String
grant_full_ctrl	读写权限授予者	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_bucket_acl()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_acl_e cos_acl = COS_ACL_PRIVATE; //替换为您自己的配置
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //设置存储桶 ACL
    cos_string_t read;
```

```
    cos_str_set(&read, "id=\\"qcs::cam::uin/100000000001:uin/100000000001\\", id=\\
    s = cos_put_bucket_acl(options, &bucket, cos_acl, &read, NULL, NULL, &resp_head
    if (cos_status_is_ok(s)) {
            printf("put bucket acl succeeded\\n");
    } else {
            printf("put bucket acl failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_bucket_acl();

    cos_http_io_deinitialize();

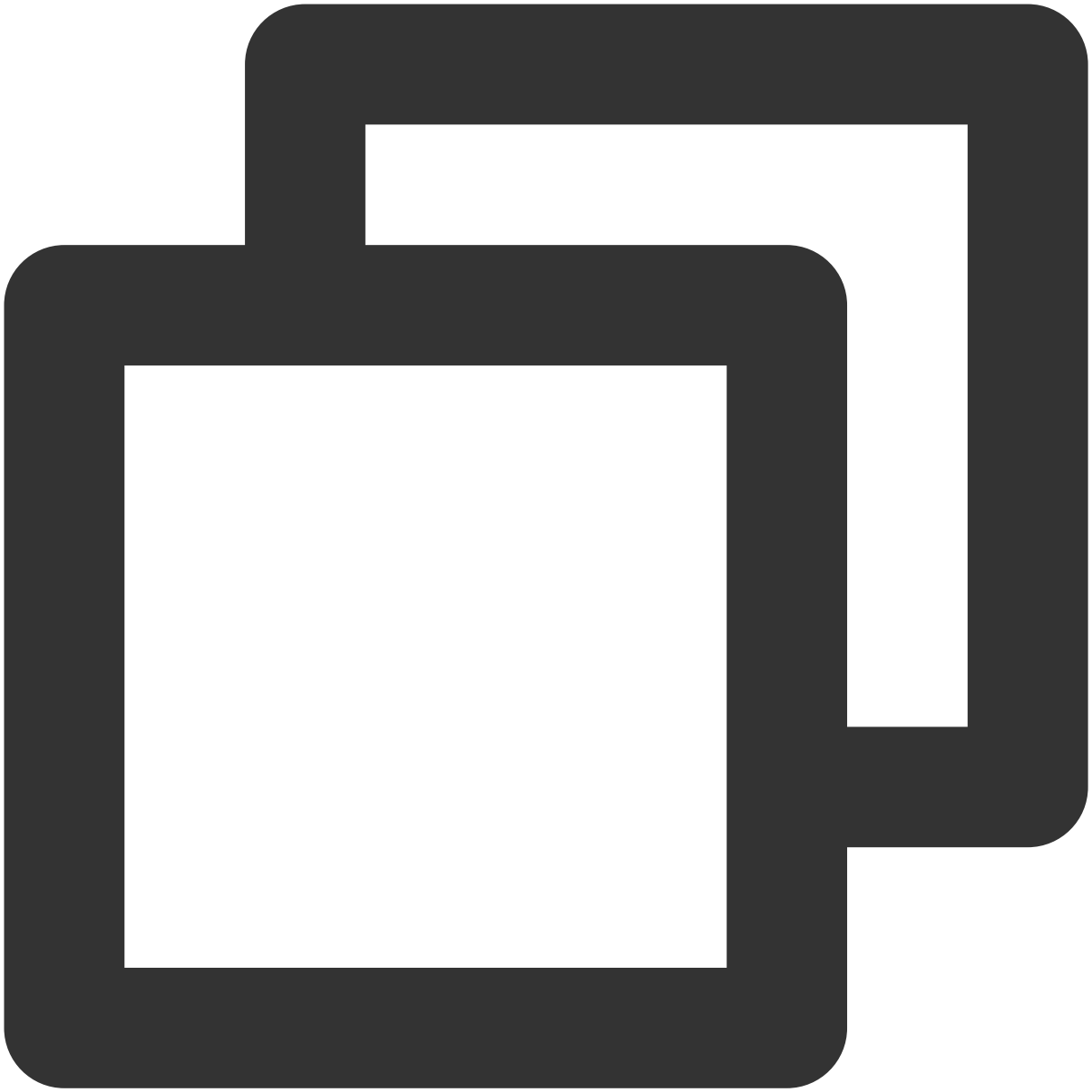
    return 0;
}
```

查询存储桶 ACL

功能说明

获取指定存储桶的访问权限控制列表（ACL）。

方法原型



```
cos_status_t *cos_get_bucket_acl(const cos_request_options_t *options,
                                 const cos_string_t *bucket,
                                 cos_acl_params_t *acl_param,
                                 cos_table_t **resp_headers)
```

参数说明

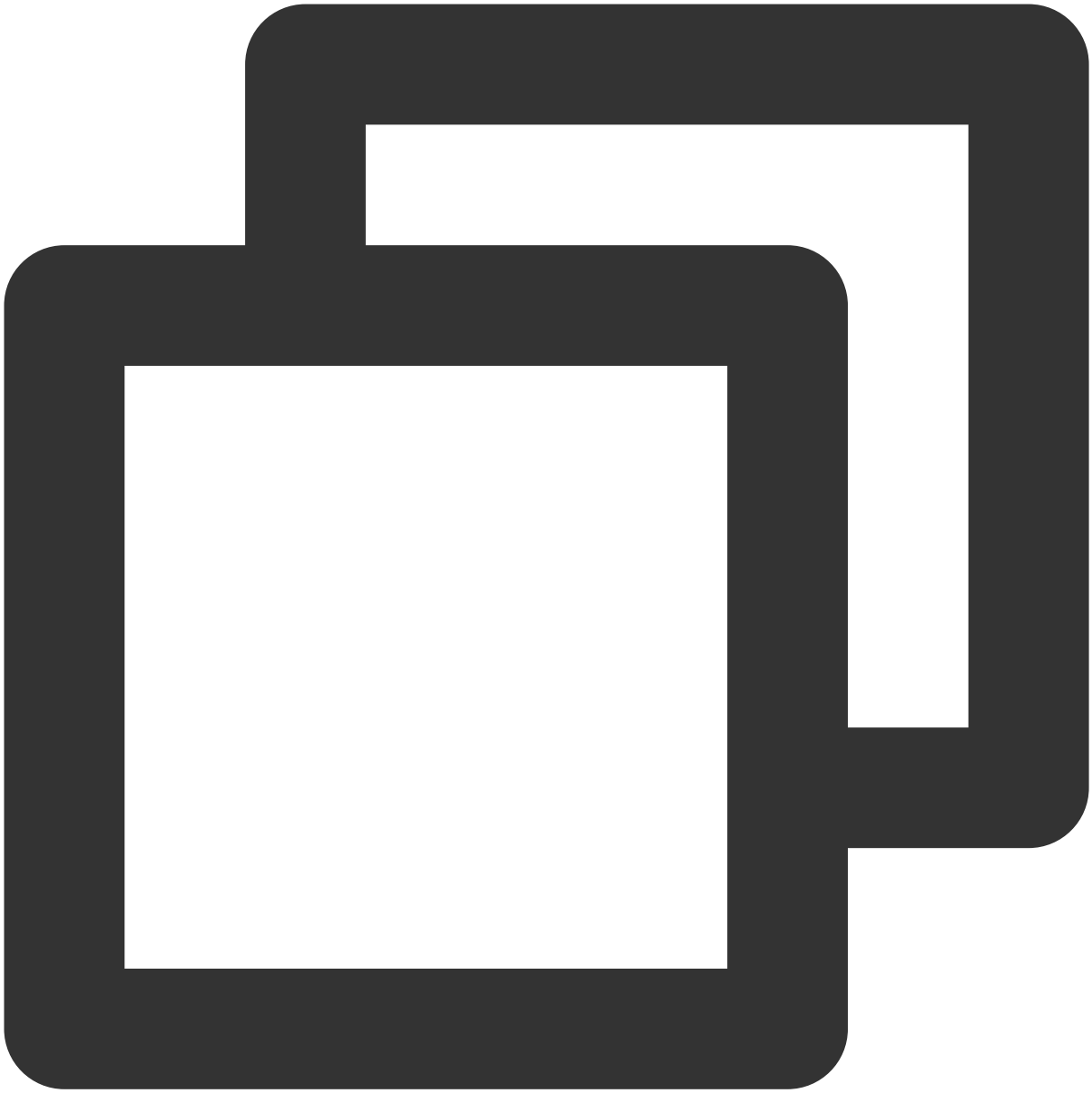
参数名称	参数描述	类型
options	COS 请求选项	Struct

bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
acl_param	请求操作参数	Struct
owner_id	请求操作返回的存储桶持有者 ID	String
owner_name	请求操作返回的存储桶持有者的名称	String
object_list	请求操作返回的被授权者信息与权限信息	Struct
type	请求操作返回的被授权者账户类型	String
id	请求操作返回的被授权者用户 ID	String
name	请求操作返回的被授权者用户名称	String
permission	请求操作返回的被授权者权限信息	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_get_bucket_acl()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //获取存储桶 ACL
    cos_acl_params_t *acl_params = NULL;
    acl_params = cos_create_acl_params(p);
```

```
    s = cos_get_bucket_acl(options, &bucket, acl_params, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("get bucket acl succeeded\\n");
        printf("acl owner id:%s, name:%s\\n", acl_params->owner_id.data, acl_params
        cos_acl_grantee_content_t *acl_content = NULL;
        cos_list_for_each_entry(cos_acl_grantee_content_t, acl_content, &acl_params
            printf("acl grantee type:%s, id:%s, name:%s, permission:%s\\n", acl_con
        }
    } else {
        printf("get bucket acl failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_get_bucket_acl();

    cos_http_io_deinitialize();

    return 0;
}
```

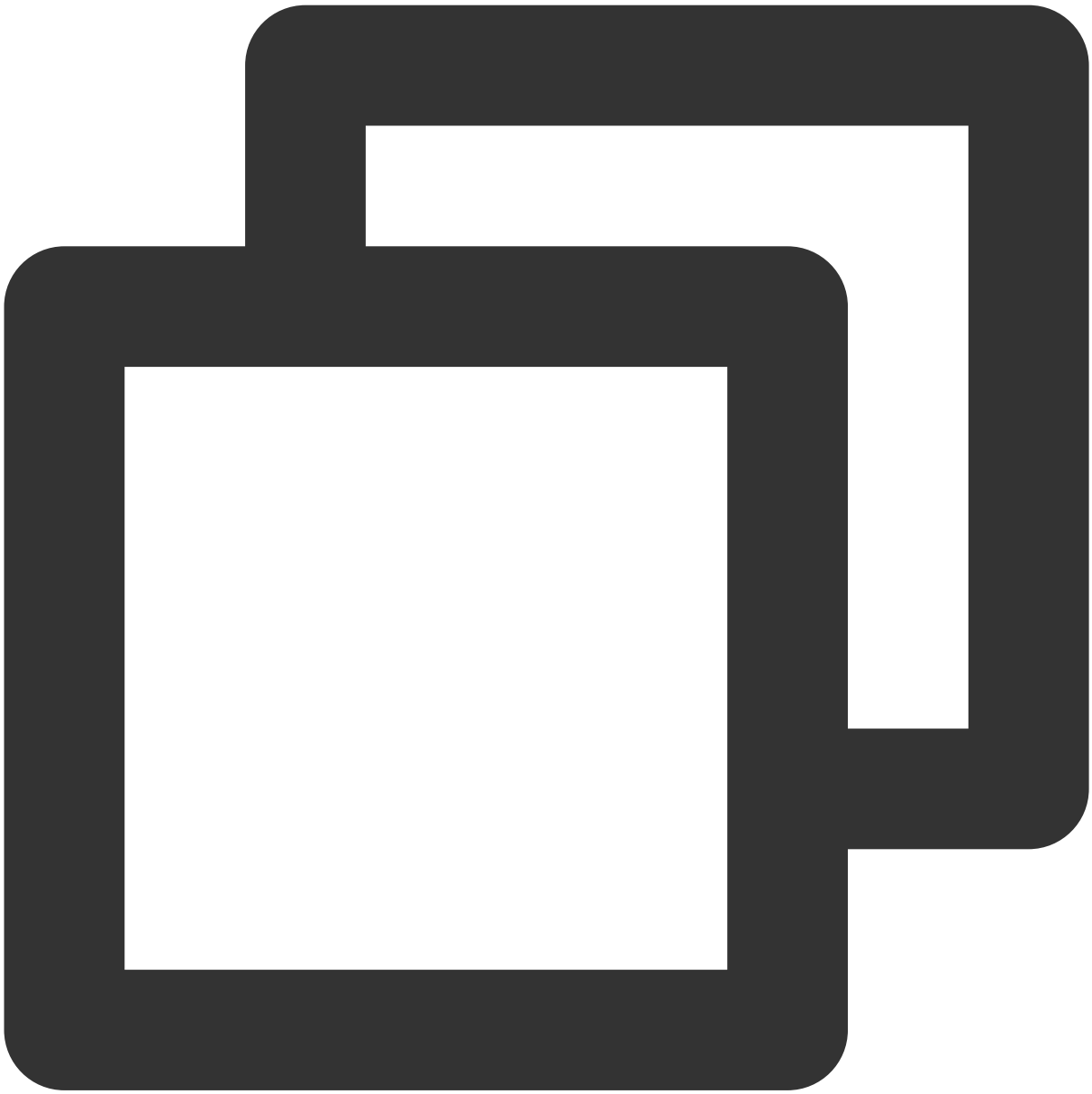
对象 ACL

设置对象 ACL

功能说明

设置指定对象访问权限控制列表（ACL）（PUT Object acl）。

方法原型



```
cos_status_t *cos_put_object_acl(const cos_request_options_t *options,
                                 const cos_string_t *bucket,
                                 const cos_string_t *object,
                                 cos_acl_e cos_acl,
                                 const cos_string_t *grant_read,
                                 const cos_string_t *grant_write,
                                 const cos_string_t *grant_full_ctrl,
```

```
                                    cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
cos_acl	允许用户自定义权限。有效值：COS_ACL_PRIVATE(0), COS_ACL_PUBLIC_READ(1), COS_ACL_PUBLIC_READ_WRITE(2) 默认值：COS_ACL_PRIVATE(0)	Enum
grant_read	赋予被授权者读取对象的权限，格式为 id="[OwnerUin]"，例如 id="100000000001"，可使用半角逗号（,）分隔多组被授权者，例如 `id="100000000001",id="100000000002"`	String
grant_write	赋予被授权者写入对象的权限，格式为 id="[OwnerUin]"，例如 id="100000000001"，可使用半角逗号（,）分隔多组被授权者，例如 `id="100000000001",id="100000000002"`	String
grant_full_ctrl	赋予被授权者操作对象的所有权限，格式为 id="[OwnerUin]"，例如 id="100000000001"，可使用半角逗号（,）分隔多组被授权者，例如 `id="100000000001",id="100000000002"`	String
resp_headers	返回 HTTP 响应消息的头域	Struct

说明

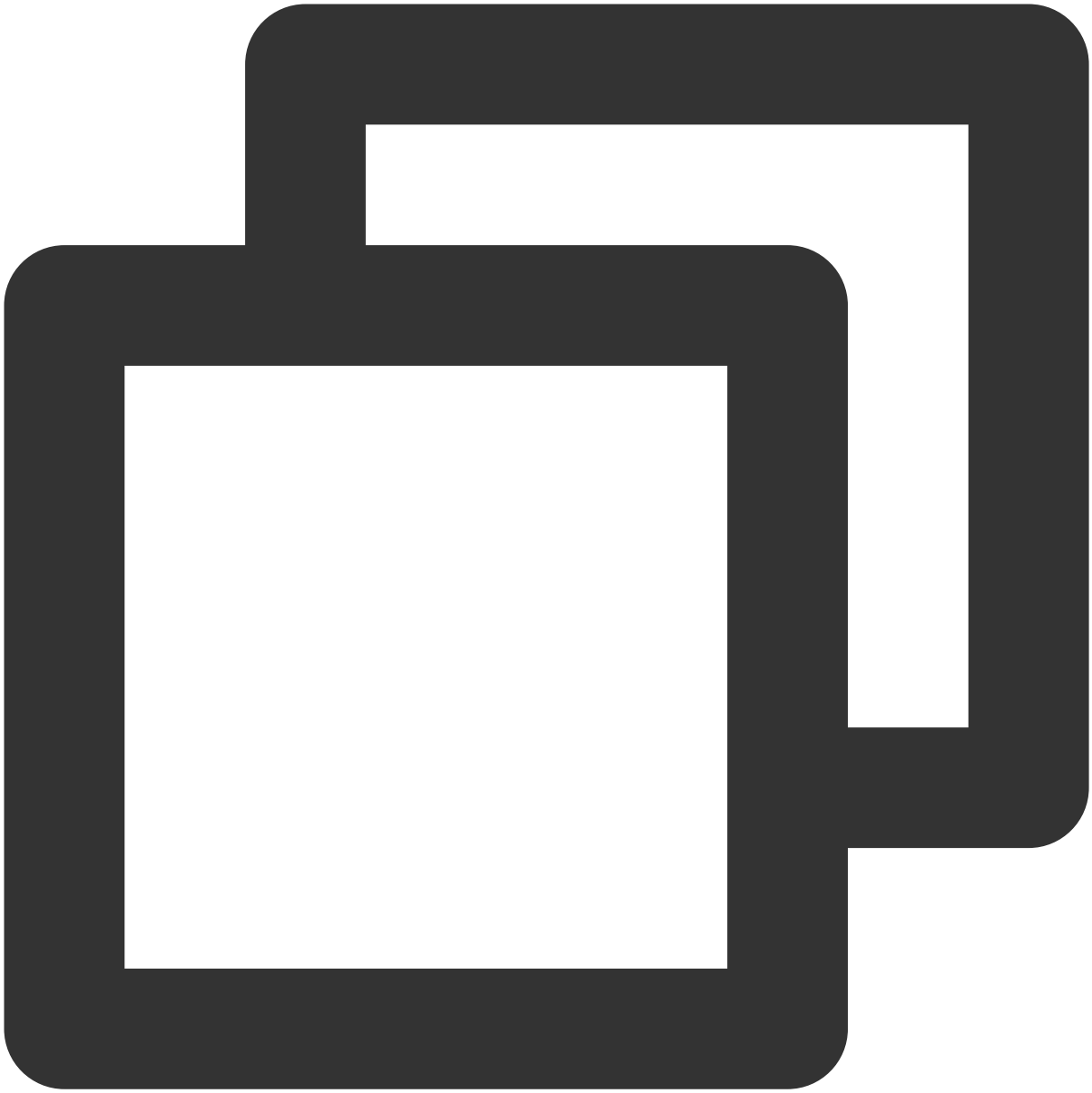
详情请参见 API 文档 PUT Object acl 和 ACL 概述。

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

版权所有：腾讯云计算（北京）有限责任公司

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
```

```
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";     //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_object_acl()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_acl_e cos_acl = COS_ACL_PRIVATE; //替换为您自己的配置
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
```

```
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //设置对象 ACL，替换您自己的acl配置
    cos_str_set(&object, TEST_OBJECT_NAME1);
    cos_string_t read;
    cos_str_set(&read, "id=\\"qcs::cam::uin/12345:uin/12345\\", id=\\"qcs::cam::uin
    s = cos_put_object_acl(options, &bucket, &object, cos_acl, &read, NULL, NULL, &
    if (cos_status_is_ok(s)) {
        printf("put object acl succeeded\\n");
    } else {
        printf("put object acl failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_object_acl();

    cos_http_io_deinitialize();

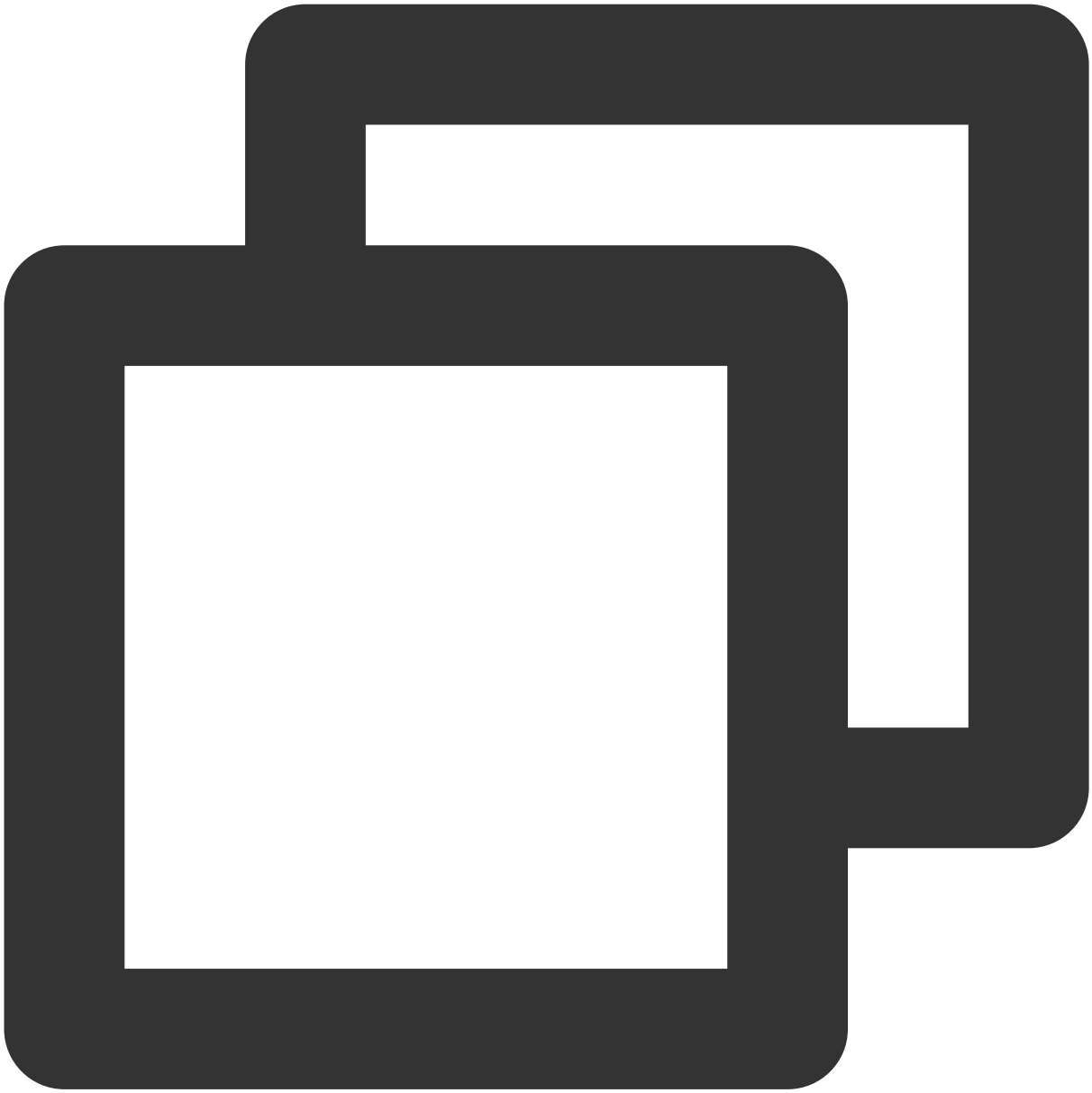
    return 0;
}
```

查询对象 ACL

功能说明

查询指定对象的访问权限控制列表（GET Object acl）。

方法原型



```
cos_status_t *cos_get_object_acl(const cos_request_options_t *options,
                                 const cos_string_t *bucket,
                                 const cos_string_t *object,
                                 cos_acl_params_t *acl_param,
                                 cos_table_t **resp_headers)
```

参数说明

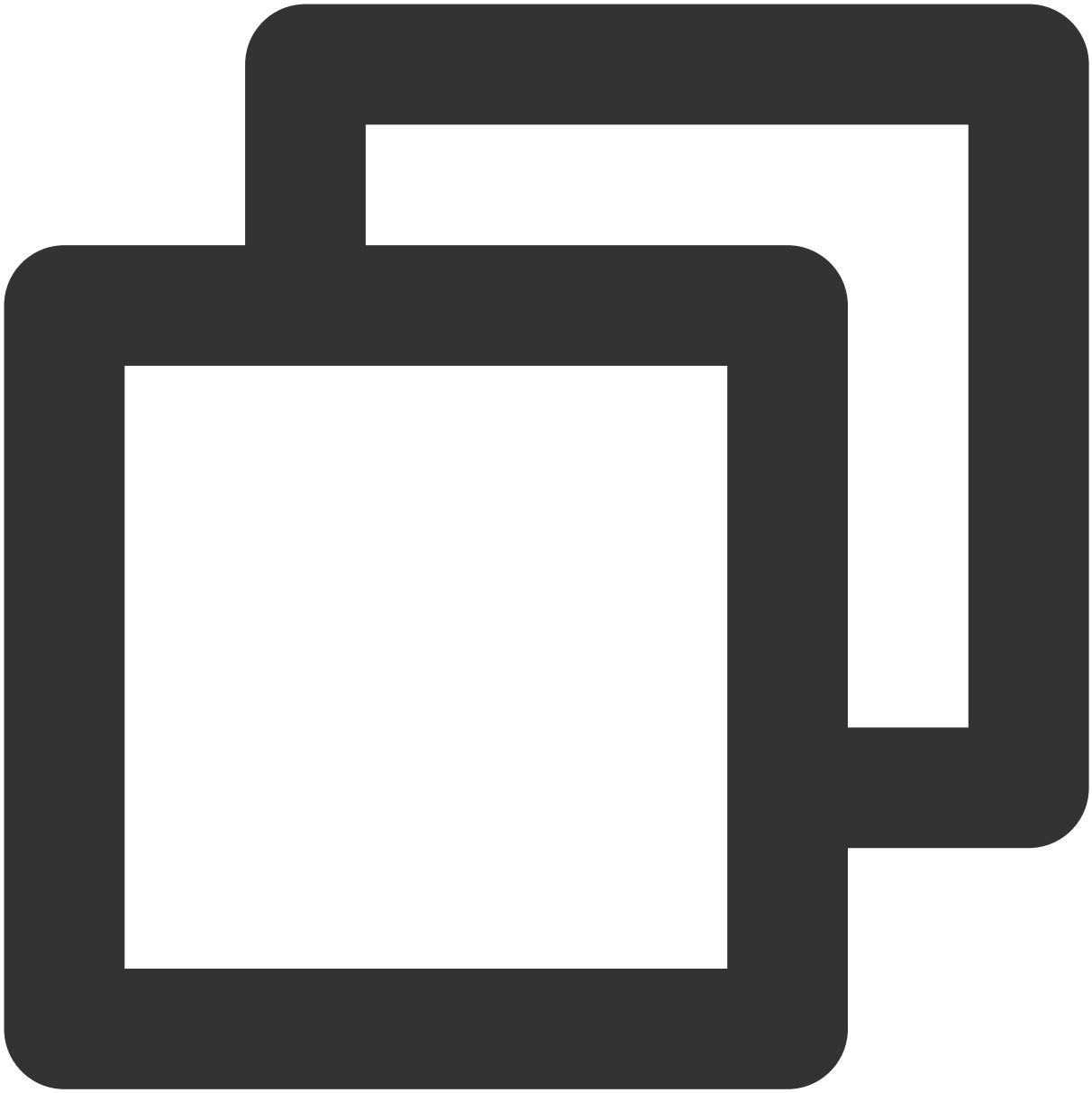
参数名称	参数描述	类型

options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
acl_param	请求操作参数	Struct
owner_id	请求操作返回的 Bucket 持有者 ID	String
owner_name	请求操作返回的 Bucket 持有者的名称	String
object_list	请求操作返回的被授权者信息与权限信息	Struct
type	请求操作返回的被授权者账户类型	String
id	请求操作返回的被授权者用户 ID	String
name	请求操作返回的被授权者用户名称	String
permission	请求操作返回的被授权者权限信息	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_get_object_acl()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_acl_grantee_content_t *acl_content = NULL;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
```

```
    cos_str_set(&object, TEST_OBJECT_NAME1);

    //获取对象 ACL
    cos_acl_params_t *acl_params2 = NULL;
    acl_params2 = cos_create_acl_params(p);
    s = cos_get_object_acl(options, &bucket, &object, acl_params2, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("get object acl succeeded\\n");
        printf("acl owner id:%s, name:%s\\n", acl_params2->owner_id.data, acl_param
        acl_content = NULL;
        cos_list_for_each_entry(cos_acl_grantee_content_t, acl_content, &acl_params
            printf("acl grantee id:%s, name:%s, permission:%s\\n", acl_content->id.
        }
    } else {
        printf("get object acl failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_get_object_acl();

    cos_http_io_deinitialize();

    return 0;
}
```

防盗链

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于存储桶 Referer 白名单或者黑名单的 API 概览以及 SDK 示例代码。

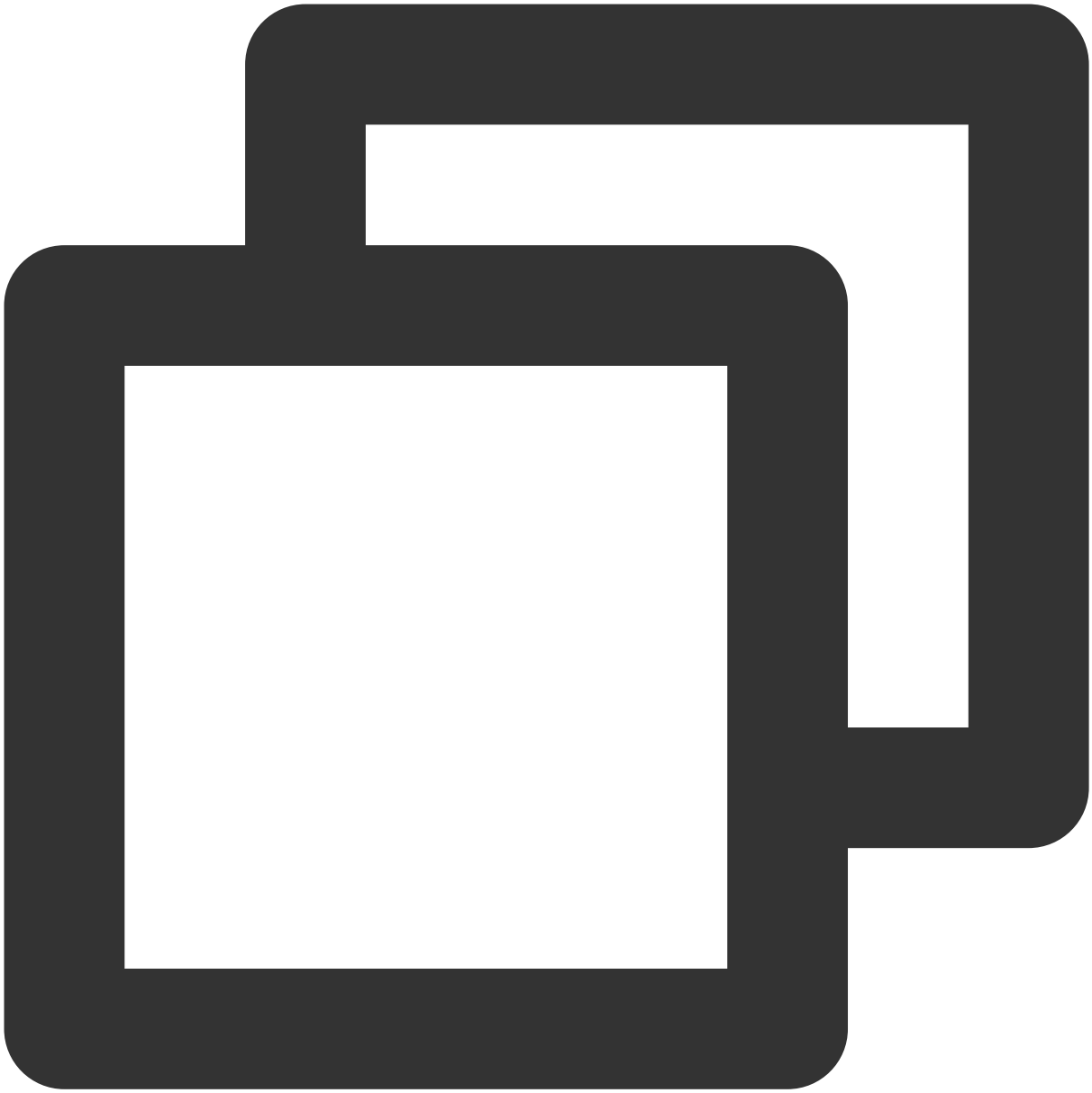
API	操作名	操作描述
PUT Bucket referer	设置存储桶 Referer	设置存储桶 Referer 白名单或者黑名单
GET Bucket referer	查询存储桶 Referer	查询存储桶 Referer 白名单或者黑名单

设置存储桶 Referer

功能说明

设置指定存储桶的 Referer 白名单或者黑名单（PUT Bucket referer）。

方法原型



```
cos_status_t *cos_put_bucket_referer(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     cos_referer_params_t *referer_params,
                                     cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct

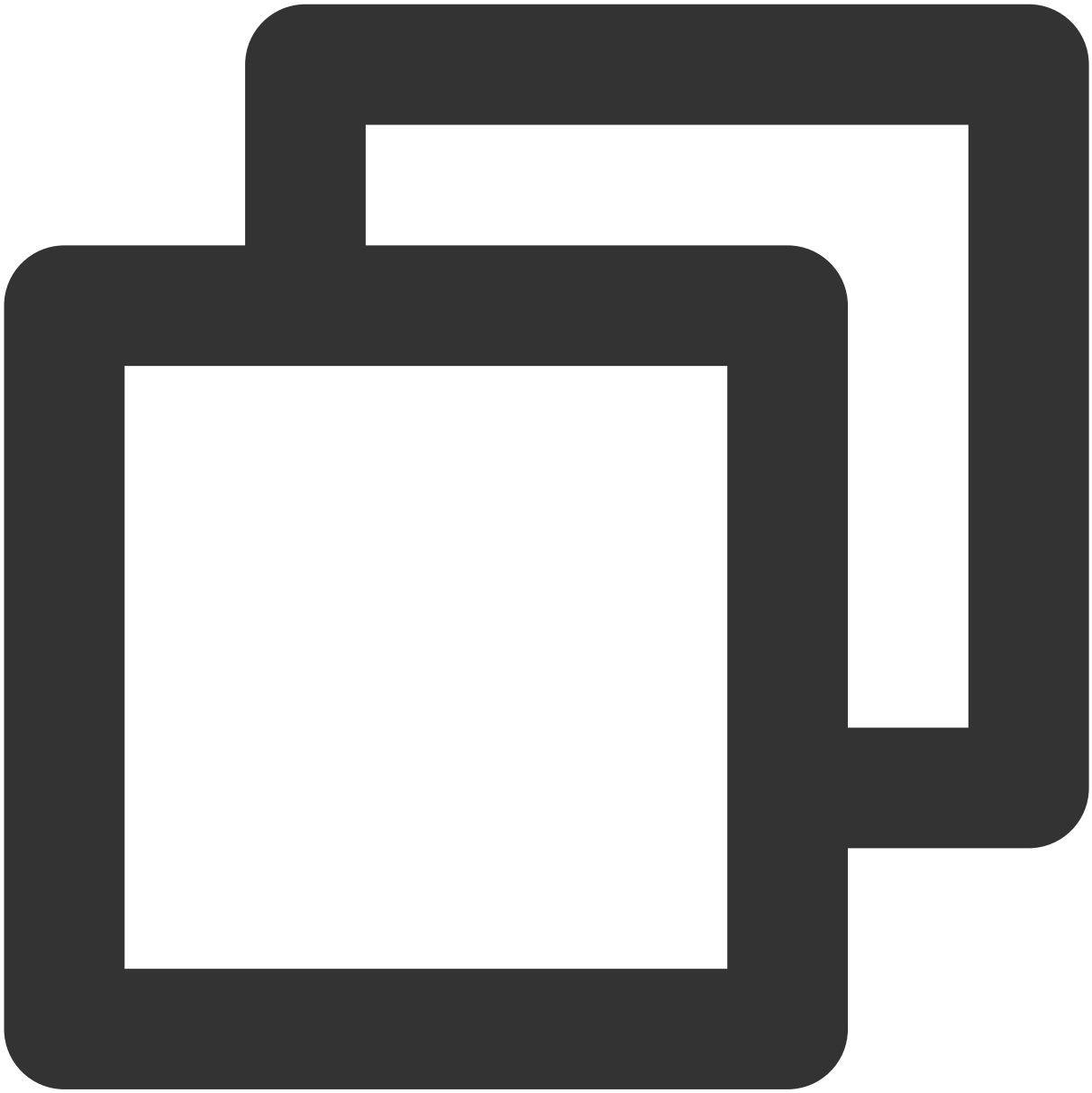
bucket	存储桶名称，Bucket 的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
referer_params	referer配置参数	Struct
status	是否开启防盗链，枚举值：Enabled、Disabled	String
referer_type	防盗链类型，枚举值：Black-List、White-List	String
empty_refer_config	是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny	String
domain_list	生效域名列表， 支持多个域名且为前缀匹配， 支持带端口的域名和 IP，支持通配符*，做二级域名或多级域名的通配	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

完整代码请参见 cos_demo.c 中的 test_referer() 函数。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_put_referer()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
    cos_request_options_t *options = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;
    cos_referer_params_t *params = NULL;
    cos_referer_domain_t *domain = NULL;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    // 替换为您的配置信息，可参见文档 https://intl.cloud.tencent.com/document/product/43
```

```
    params = cos_create_referer_params(pool);
    cos_str_set(&params->status, "Enabled");
    cos_str_set(&params->referer_type, "White-List");
    cos_str_set(&params->empty_refer_config, "Allow");
    domain = cos_create_referer_domain(pool);
    cos_str_set(&domain->domain, "www.qq.com");
    cos_list_add_tail(&domain->node, &params->domain_list);
    domain = cos_create_referer_domain(pool);
    cos_str_set(&domain->domain, "*.tencent.com");
    cos_list_add_tail(&domain->node, &params->domain_list);

    // put referer
    status = cos_put_bucket_referer(options, &bucket, params, &resp_headers);
    log_status(status);

    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_put_referer();

    cos_http_io_deinitialize();

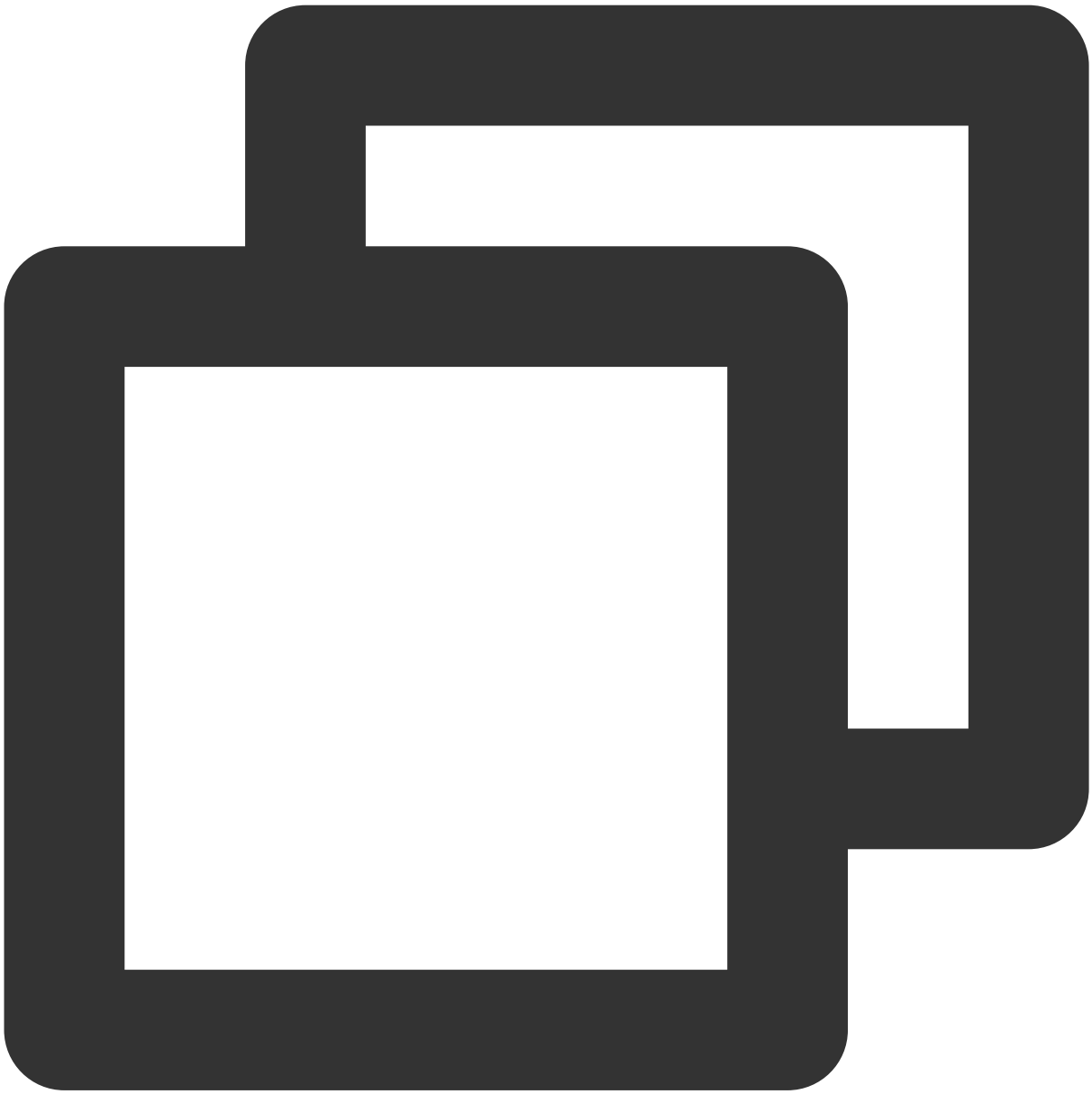
    return 0;
}
```

查询存储桶 Referer

功能说明

查询指定存储桶 Referer 白名单或者黑名单（GET Bucket referer）。

方法原型



```
cos_status_t *cos_get_bucket_referer(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     cos_referer_params_t *referer_params,
                                     cos_table_t **resp_headers);
```

参数说明

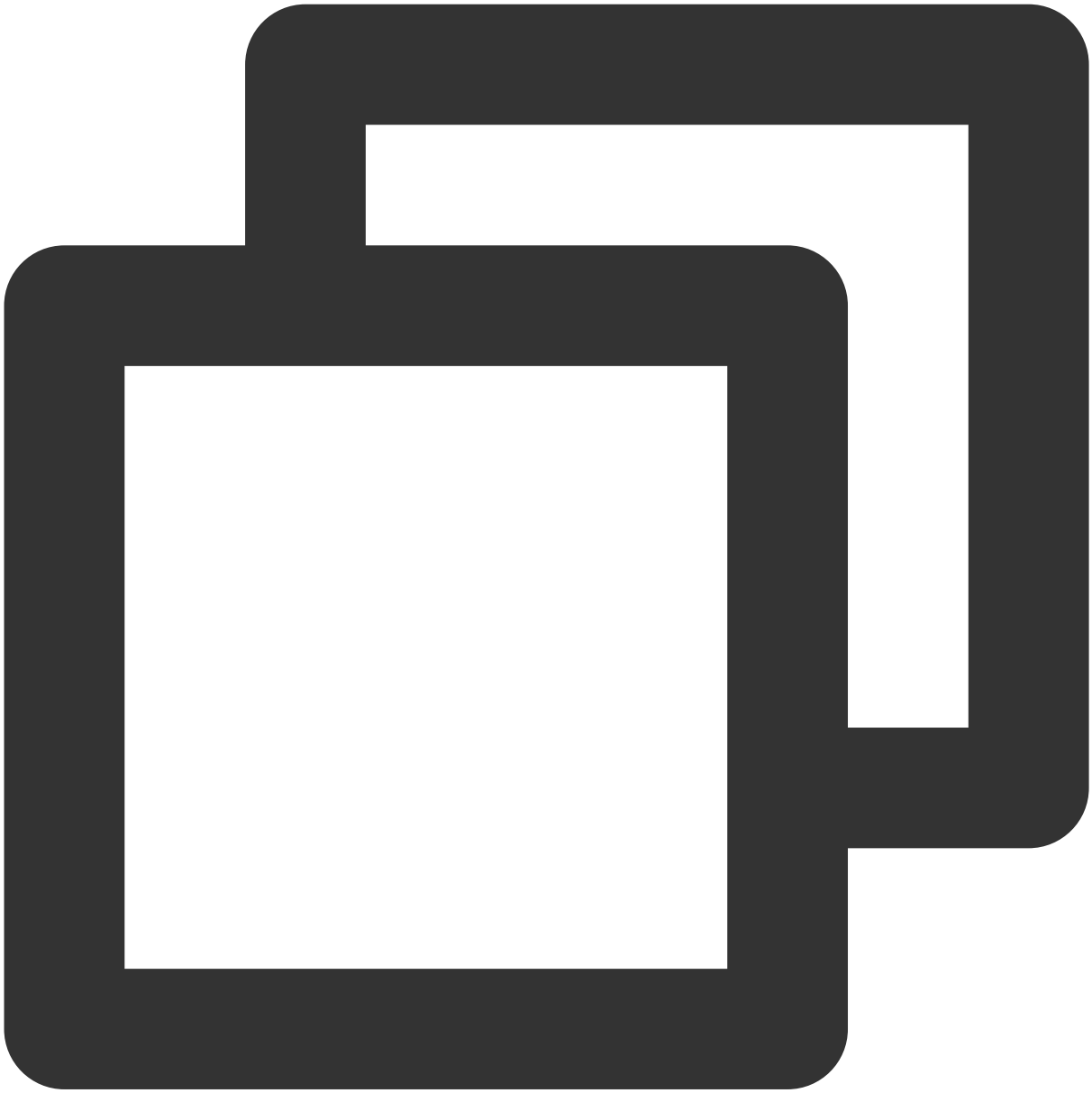
参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，Bucket 的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
referer_params	referer配置参数	Struct
status	是否开启防盗链，枚举值：Enabled、Disabled	String
referer_type	防盗链类型，枚举值：Black-List、White-List	String
empty_refer_config	是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny	String
domain_list	生效域名列表， 支持多个域名且为前缀匹配， 支持带端口的域名和 IP， 支持通配符*，做二级域名或多级域名的通配	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

请求示例

完整代码请参见 cos_demo.c 中的 test_referer() 函数。



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

static void log_get_referer(cos_referer_params_t *result)
{
    int index = 0;
    cos_referer_domain_t *domain;

    cos_warn_log("status: %s", result->status.data);
    cos_warn_log("referer_type: %s", result->referer_type.data);
    cos_warn_log("empty_refer_config: %s", result->empty_refer_config.data);

    cos_list_for_each_entry(cos_referer_domain_t, domain, &result->domain_list, nod
        cos_warn_log("domain index:%d", ++index);
        cos_warn_log("domain: %s", domain->domain.data);
    }
}

void test_get_referer()
{
    cos_pool_t *pool = NULL;
    int is_cname = 0;
    cos_status_t *status = NULL;
```

```
    cos_request_options_t *options = NULL;
    cos_table_t *resp_headers = NULL;
    cos_string_t bucket;
    cos_referer_params_t *result = NULL;

    //创建内存池
    cos_pool_create(&pool, NULL);

    //初始化请求选项
    options = cos_request_options_create(pool);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    // get referer
    result = cos_create_referer_params(pool);
    status = cos_get_bucket_referer(options, &bucket, result, &resp_headers);
    log_status(status);
    if (status->code == 200) {
        log_get_referer(result);
    }

    cos_pool_destroy(pool);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_get_referer();

    cos_http_io_deinitialize();

    return 0;
}
```

版权所有：腾讯云计算（北京）有限责任公司

数据校验
CRC64 校验

最近更新时间：2024-01-04 17:06:09

简介

数据在客户端和服务器间传输时可能会出现错误，COS 除了可以通过 MD5 和自定义属性 验证数据完整性外，还可以通过 CRC64 检验码来进行数据校验。
COS 会对新上传的对象进行 CRC64 计算，并将结果作为对象的属性进行存储，随后在返回的响应头部中携带 x-cos-hash-crc64ecma，该头部表示上传对象的 CRC64 值，根据 ECMA-182标准 计算得到。对于 CRC64 特性上线前就已经存在于 COS 的对象，COS 不会对其计算 CRC64 值，所以获取此类对象时不会返回其 CRC64 值。

操作说明

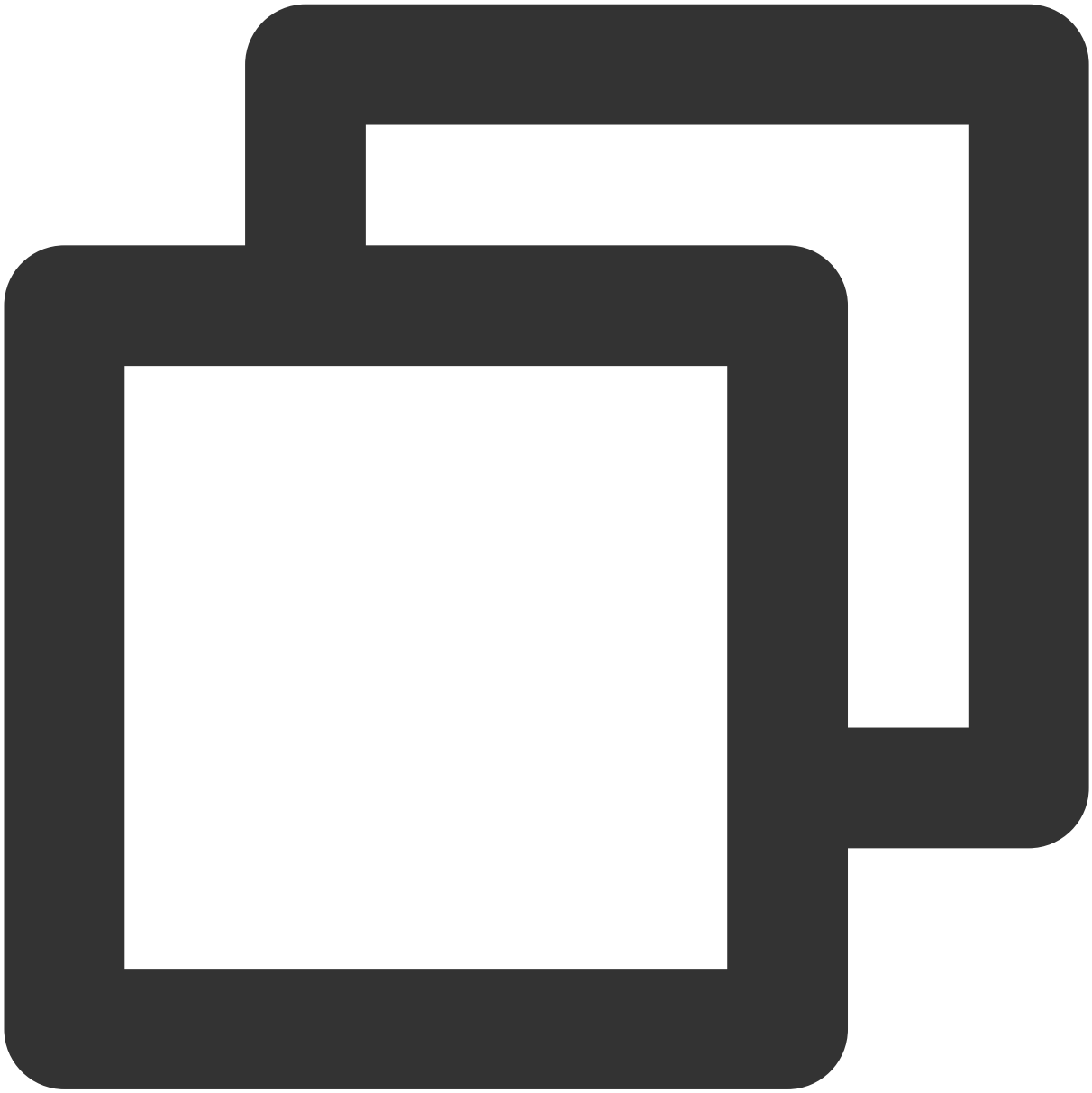
目前支持 CRC64 的 API 如下：
简单上传接口
PUT Object 和 POST Object：用户可在返回的响应头中获得文件 CRC64 校验值。
分块上传接口
Upload Part：用户可以根据 COS 返回的 CRC64 值与本地计算的数值进行比较验证。
Complete Multipart Upload：如果每个分块都有 CRC64 属性，则会返回整个对象的 CRC64 值，如果某些分块不具备 CRC64 值，则不返回。
执行 Upload Part - Copy 时，会返回对应的 CRC64 值。
执行 PUT Object - Copy 时，如果源对象存在 CRC64 值，则返回 CRC64，否则不返回。
执行 HEAD Object 和 GET Object 时，如果对象存在 CRC64，则返回。用户可以根据 COS 返回的 CRC64 值和本地计算的 CRC64 进行比较验证。

API 接口示例

分块上传响应

下面为用户发出 Upload Part 请求后得到的响应示例。x-cos-hash-crc64ecma 头部表示分块的 CRC64 值，用户可以通过该值与本地计算的 CRC64 值进行比较，从而校验分块完整性。

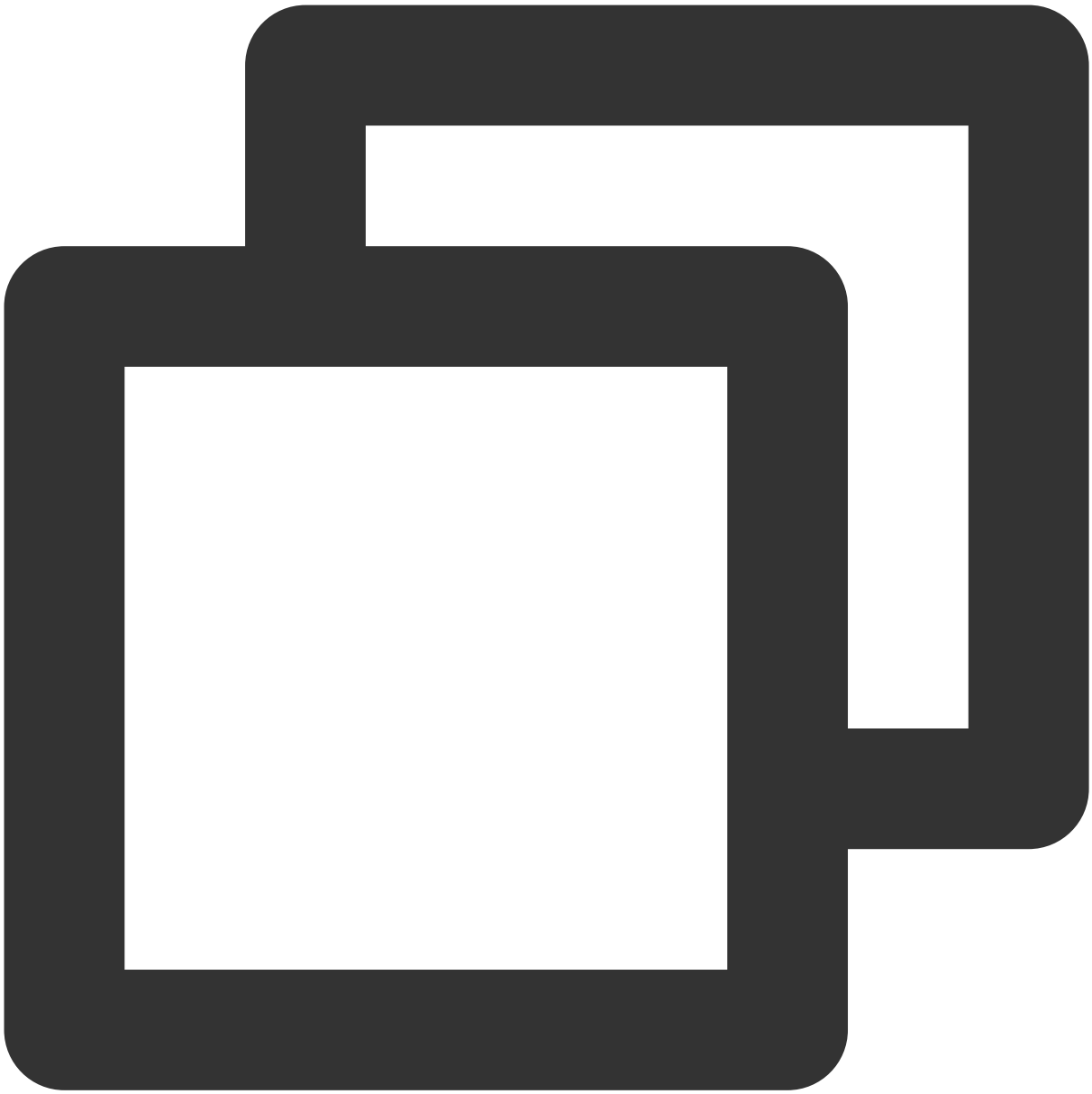
版权所有：腾讯云计算（北京）有限责任公司



```
HTTP/1.1 200 OK
content-length: 0
connection: close
date: Thu, 05 Dec 2019 01:58:03 GMT
etag: "358e8c8b1bfa35ee3bd44cb3d2cc416b"
server: tencent-cos
x-cos-hash-crc64ecma: 15060521397700495958
x-cos-request-id: NWRlODY0MmJfMjBiNDU4NjRfNjkyZl80ZjZi****
```

完成分块上传响应

下面为用户发出 Complete Multipart Upload 请求后得到的响应示例。x-cos-hash-crc64ecma 头部表示整个对象的 CRC64 值，用户可以通过该值与本地计算的 CRC64 值进行比较，从而校验对象完整性。



```
HTTP/1.1 200 OK
content-type: application/xml
transfer-encoding: chunked
connection: close
date: Thu, 05 Dec 2019 02:01:17 GMT
server: tencent-cos
x-cos-hash-crc64ecma: 15060521397700495958
x-cos-request-id: NWRlODY0ZWRfMjNiMjU4NjRfOGQ4Ml81MDEw****
```

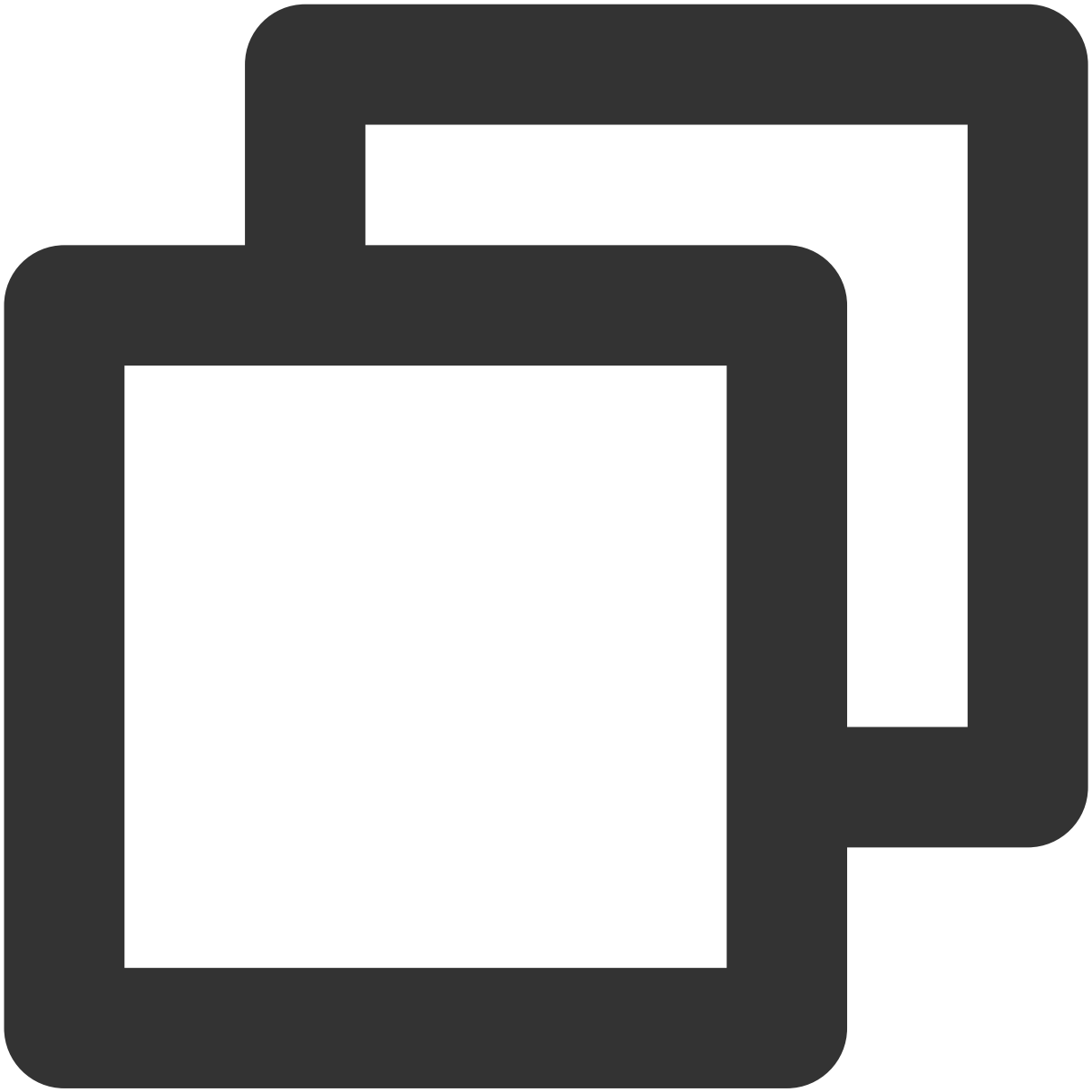
```
[Object Content]
```

SDK 示例

功能说明

用于在对象上传和下载的时候对对象数据做 crc64 一致性校验。

方法原型



```
cos_status_t *cos_upload_part_from_buffer(const cos_request_options_t *options,
                                          const cos_string_t *bucket,
                                          const cos_string_t *object,
                                          const cos_string_t *upload_id,
                                          int part_num,
                                          cos_list_t *buffer,
                                          cos_table_t **resp_headers);

cos_status_t *cos_upload_part_from_file(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        const cos_string_t *object,
```

```
                                        const cos_string_t *upload_id,
                                        int part_num,
                                        cos_upload_file_t *upload_file,
                                        cos_table_t **resp_headers);

cos_status_t *cos_download_part_to_file(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        const cos_string_t *object,
                                        cos_upload_file_t *download_file,
                                        cos_table_t **resp_headers);

cos_status_t *cos_put_object_from_buffer(const cos_request_options_t *options,
                                         const cos_string_t *bucket,
                                         const cos_string_t *object,
                                         cos_list_t *buffer,
                                         cos_table_t *headers,
                                         cos_table_t **resp_headers);

cos_status_t *cos_put_object_from_file(const cos_request_options_t *options,
                                       const cos_string_t *bucket,
                                       const cos_string_t *object,
                                       const cos_string_t *filename,
                                       cos_table_t *headers,
                                       cos_table_t **resp_headers);

cos_status_t *cos_get_object_to_buffer(const cos_request_options_t *options,
                                       const cos_string_t *bucket,
                                       const cos_string_t *object,
                                       cos_table_t *headers,
                                       cos_table_t *params,
                                       cos_list_t *buffer,
                                       cos_table_t **resp_headers);

cos_status_t *cos_get_object_to_file(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     const cos_string_t *object,
                                     cos_table_t *headers,
                                     cos_table_t *params,
                                     cos_string_t *filename,
                                     cos_table_t **resp_headers);

cos_status_t *cos_append_object_from_buffer(const cos_request_options_t *options,
                                            const cos_string_t *bucket,
                                            const cos_string_t *object,
                                            int64_t position,
                                            cos_list_t *buffer,
                                            cos_table_t *headers,
```

```
                                            cos_table_t **resp_headers);

cos_status_t *cos_append_object_from_file(const cos_request_options_t *options,
                                          const cos_string_t *bucket,
                                          const cos_string_t *object,
                                          int64_t position,
                                          const cos_string_t *append_file,
                                          cos_table_t *headers,
                                          cos_table_t **resp_headers);

cos_status_t *cos_resumable_download_file(cos_request_options_t *options,
                                            cos_string_t *bucket,
                                            cos_string_t *object,
                                            cos_string_t *filepath,
                                            cos_table_t *headers,
                                            cos_table_t *params,
                                            cos_resumable_clt_params_t *clt_params,
                                            cos_progress_callback progress_callback
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
options.ctl.options.enable_crc	crc 校验选项，默认开启 crc 校验	Int

说明

其他参数说明则参见 对象操作。

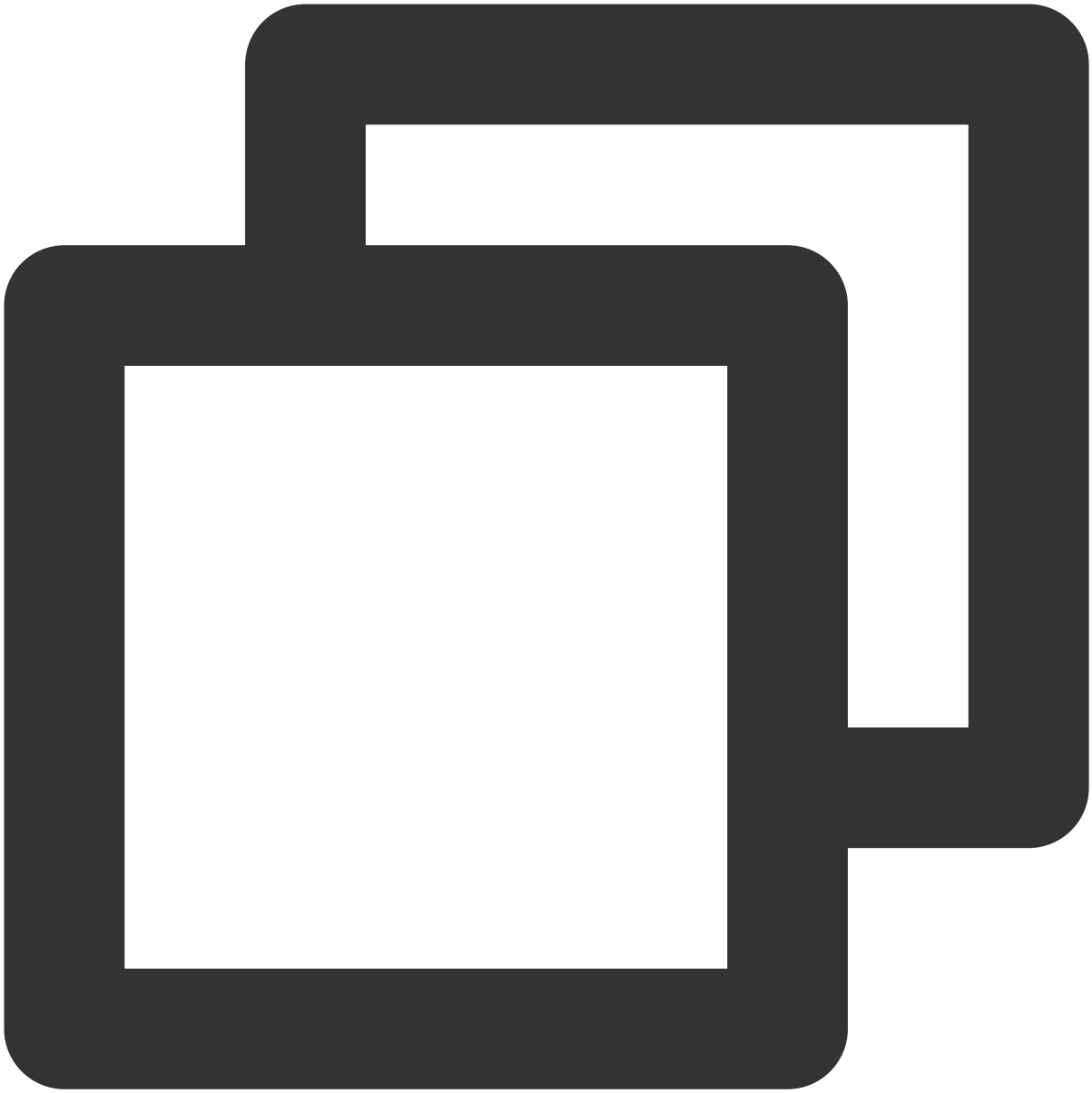
返回结果说明

返回结果	描述	类型
code	错误码（crc 校验不通过返回：COSE_CRC_INCONSISTENT_ERROR）	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

这里只用了简单上传和下载两个接口来举例，其它接口也是同样使用方式，只需要将 **options** > **ctl** > **options** > **enable_crc** 设置为 COS_TRUE 即可（默认值也是 COS_TRUE）。

版权所有：腾讯云计算（北京）有限责任公司



```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <unistd.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
```

```
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_data_crc64()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers;
    cos_table_t *headers = NULL;
    cos_list_t buffer;
    cos_list_t download_buffer;
    cos_string_t object;
    cos_buf_t *content = NULL;
    char *str = "This is my test data.";

    // 基本配置
    cos_pool_create(&p, NULL);
```

```
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    options->ctl->options->enable_crc = COS_TRUE;    // 开启crc校验，默认也是开启的
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_OBJECT_NAME1);

    // 上传对象，开启crc校验
    cos_list_init(&buffer);
    content = cos_buf_pack(options->pool, str, strlen(str));
    cos_list_add_tail(&content->node, &buffer);
    s = cos_put_object_from_buffer(options, &bucket, &object,
                    &buffer, headers, &resp_headers);
    log_status(s);

    // 下载对象，开启crc校验
    cos_list_init(&download_buffer);
    s = cos_get_object_to_buffer(options, &bucket, &object,
                        headers, NULL, &download_buffer, &resp_headers);
    log_status(s);

    // 销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_data_crc64();

    cos_http_io_deinitialize();

    return 0;
}
```

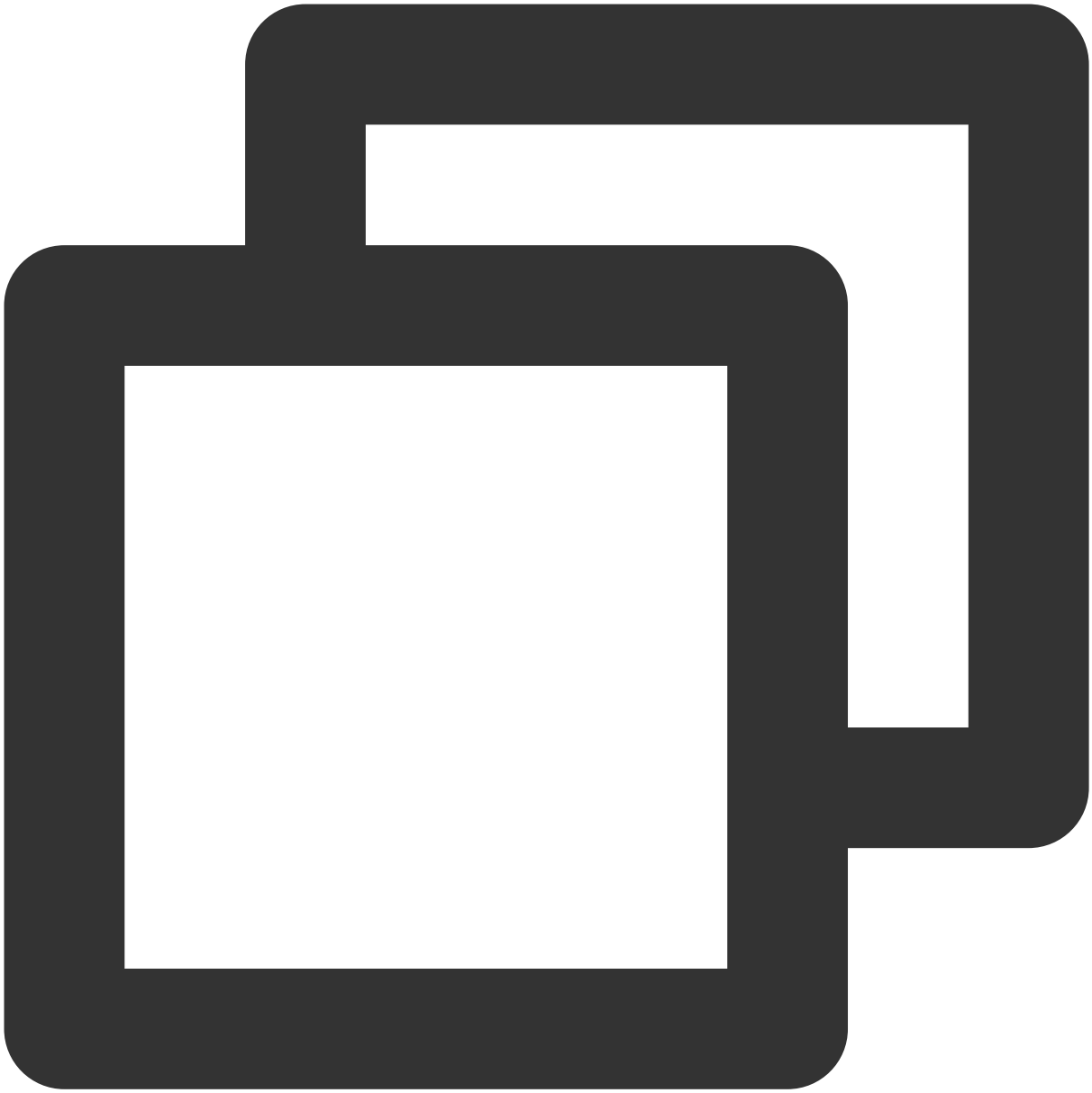
异常处理

最近更新时间：2024-01-04 17:06:08

简介

SDK 调用失败时，结果信息包含在 API 返回的 cos_status_t 结构中。
SDK 中使用每一个 API 的正确做法如下所示，为了简要，文档中范例不再给出具体异常的处理，仅给出 API 的使用范例。

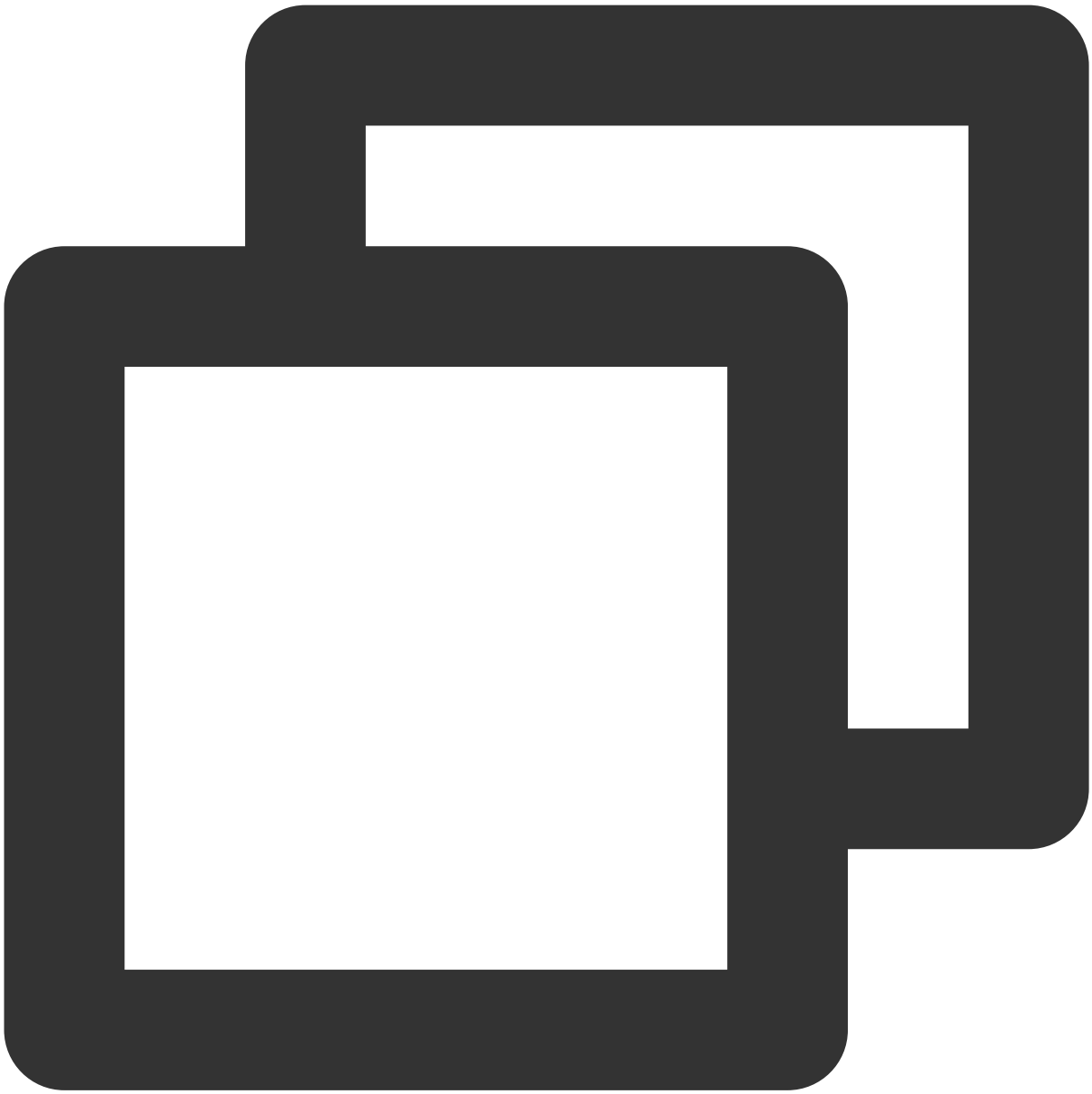
版权所有：腾讯云计算（北京）有限责任公司



```
cos_status_t *s = NULL;
s = cos_put_object_from_file(options, &bucket, &object, &file, &headers, &resp_head
if (!s && !cos_status_is_ok(s)) {
        // 按需要进行异常场景的日志输出和处理
        cos_warn_log("failed to put object from file", buf);
        if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
        if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
        if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}
```

客户端异常

当 cos_status_t 结构中 code 成员值小于0时，表明发生 SDK 本地客户端错误，错误码信息参考枚举 cos_error_code_e 定义。



```
typedef enum {
        COSE_OK = 0,
        COSE_OUT_MEMORY = -1000,
        COSE_OVER_MEMORY = -999,
        COSE_FAILED_CONNECT = -998,
```

```
        COSE_ABORT_CALLBACK = -997,
        COSE_INTERNAL_ERROR = -996,
        COSE_REQUEST_TIMEOUT = -995,
        COSE_INVALID_ARGUMENT = -994,
        COSE_INVALID_OPERATION = -993,
        COSE_CONNECTION_FAILED = -992,
        COSE_FAILED_INITIALIZE = -991,
        COSE_NAME_LOOKUP_ERROR = -990,
        COSE_FAILED_VERIFICATION = -989,
        COSE_WRITE_BODY_ERROR = -988,
        COSE_READ_BODY_ERROR = -987,
        COSE_SERVICE_ERROR = -986,
        COSE_OPEN_FILE_ERROR = -985,
        COSE_FILE_SEEK_ERROR = -984,
        COSE_FILE_INFO_ERROR = -983,
        COSE_FILE_READ_ERROR = -982,
        COSE_FILE_WRITE_ERROR = -981,
        COSE_XML_PARSE_ERROR = -980,
        COSE_UTF8_ENCODE_ERROR = -979,
        COSE_CRC_INCONSISTENT_ERROR = -978,
        COSE_FILE_FLUSH_ERROR = -977,
        COSE_FILE_TRUNC_ERROR = -976,
        COSE_UNKNOWN_ERROR = -100
} cos_error_code_e;
```

服务端异常

当 cos_status_t 结构中 code 成员值大于0时，表明发生网络侧错误。

以下是 cos_status_t 结构的描述：

cos_status_t 成员	描述	类型
code	response 的 status 状态码， 4xx 是指请求因客户端而失败， 5xx 是服务端异常导致的失败，详情请参见 错误码 文档	Int
error_code	请求失败时 body 返回的 Error Code，详情请参见 错误码 文档	String
error_msg	请求失败时 body 返回的 Error Message，详情请参见 错误码 文档	String
req_id	请求 ID，用于标识用户唯一请求	String

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**。
4. 稍侯片刻，便能看到相应的智能诊断结果。

版权所有：腾讯云计算（北京）有限责任公司

设置访问域名（CDN/全球加速）

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于如何使用非默认域名请求对象存储（Cloud Object Storage，COS）服务。

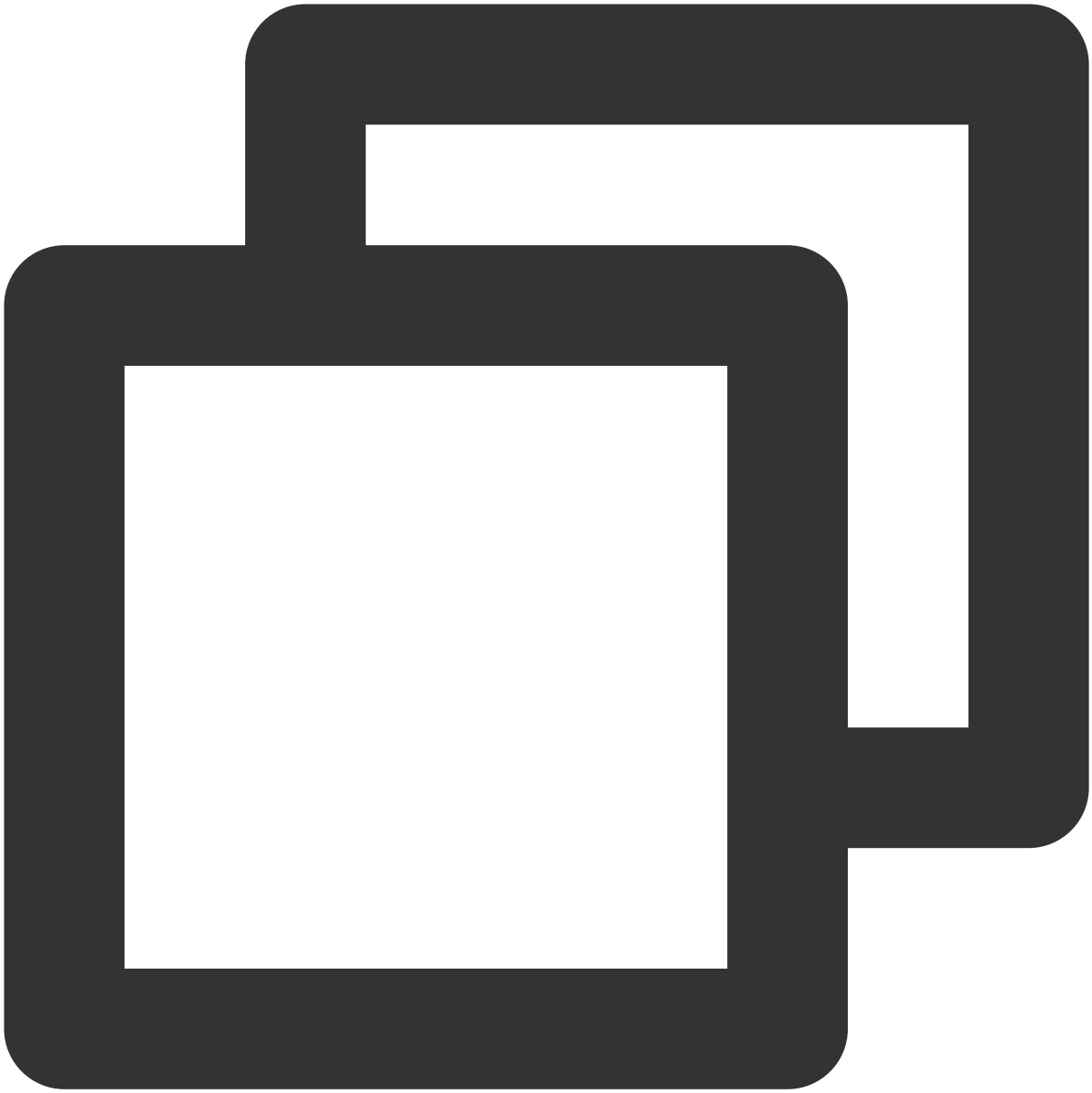
默认 CDN 加速域名

功能说明

在 COS 控制台为存储桶开启默认 CDN 加速域名后，可在 SDK 代码中设置对应的域名，以便实现默认加速功能。关于如何开启默认加速域名请参见 开启默认 CDN 加速域名。

请求示例

版权所有：腾讯云计算（北京）有限责任公司



```
cos_pool_t *p = NULL;
cos_request_options_t *options = NULL;
cos_pool_create(&p, NULL);
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
//请求域名格式为 <bucketname-appid>.file.myqcloud.com
cos_str_set(&options->config->endpoint, "<bucketname-appid>.file.myqcloud.com");
```

参数说明

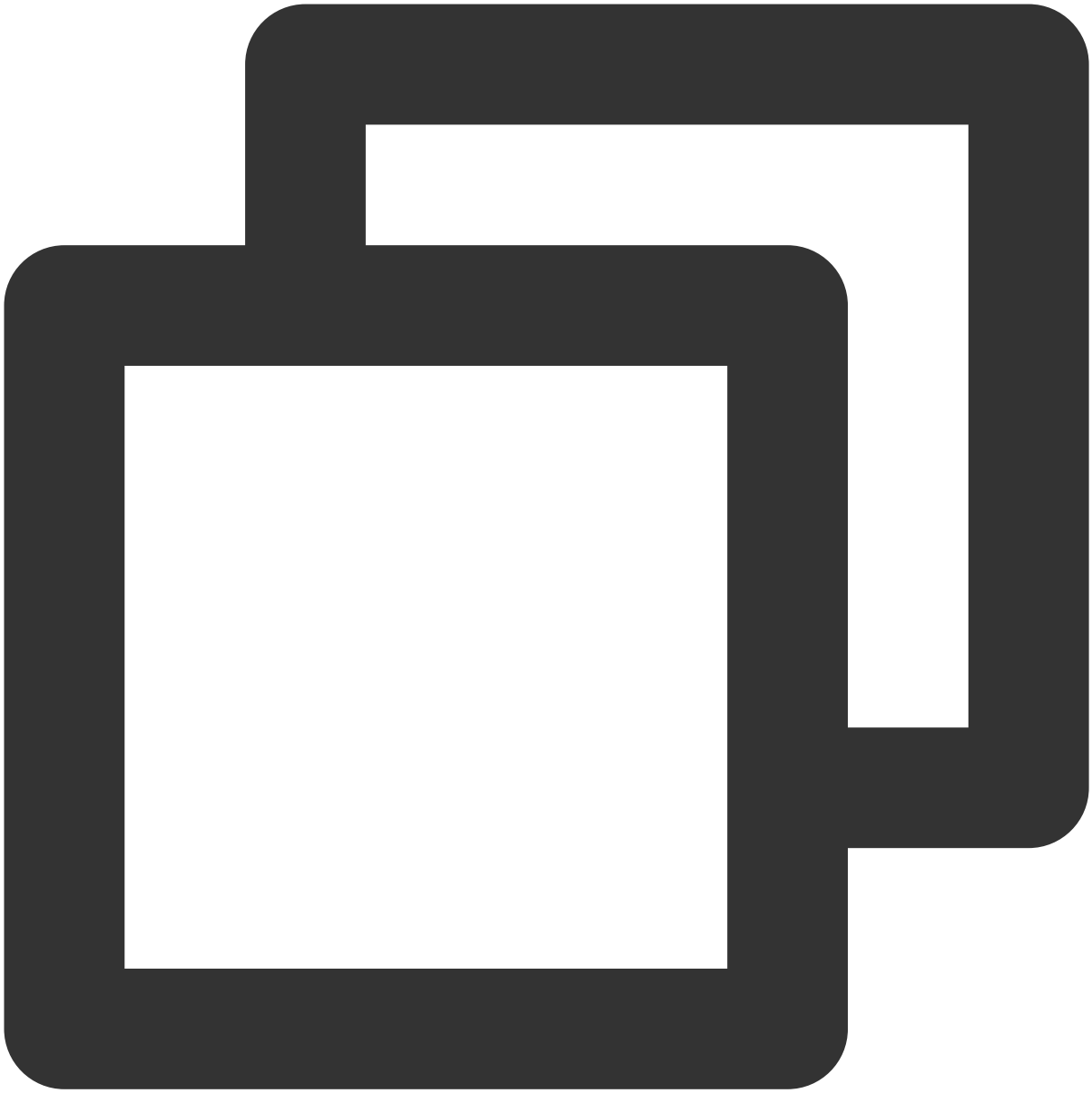
参数名称	描述	类型
endpoint	通过设置域名后缀，使用 CDN 加速域名	String

自定义 CDN 加速域名

功能说明

在 COS 控制台为存储桶设置自定义 CDN 加速域名后，可在 SDK 代码中设置对应的域名，以便实现自定义域名的加速功能。关于如何开启自定义 CDN 加速域名，请参见 开启自定义 CDN 加速域名。

请求示例



```
cos_pool_t *p = NULL;
cos_request_options_t *options = NULL;
cos_pool_create(&p, NULL);
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
//请求域名为 your.domain.com
cos_str_set(&options->config->endpoint, "your.domain.com"); //自定义CDN域名
// is_cname设置为1
options->config->is_cname = 1;
```

参数说明

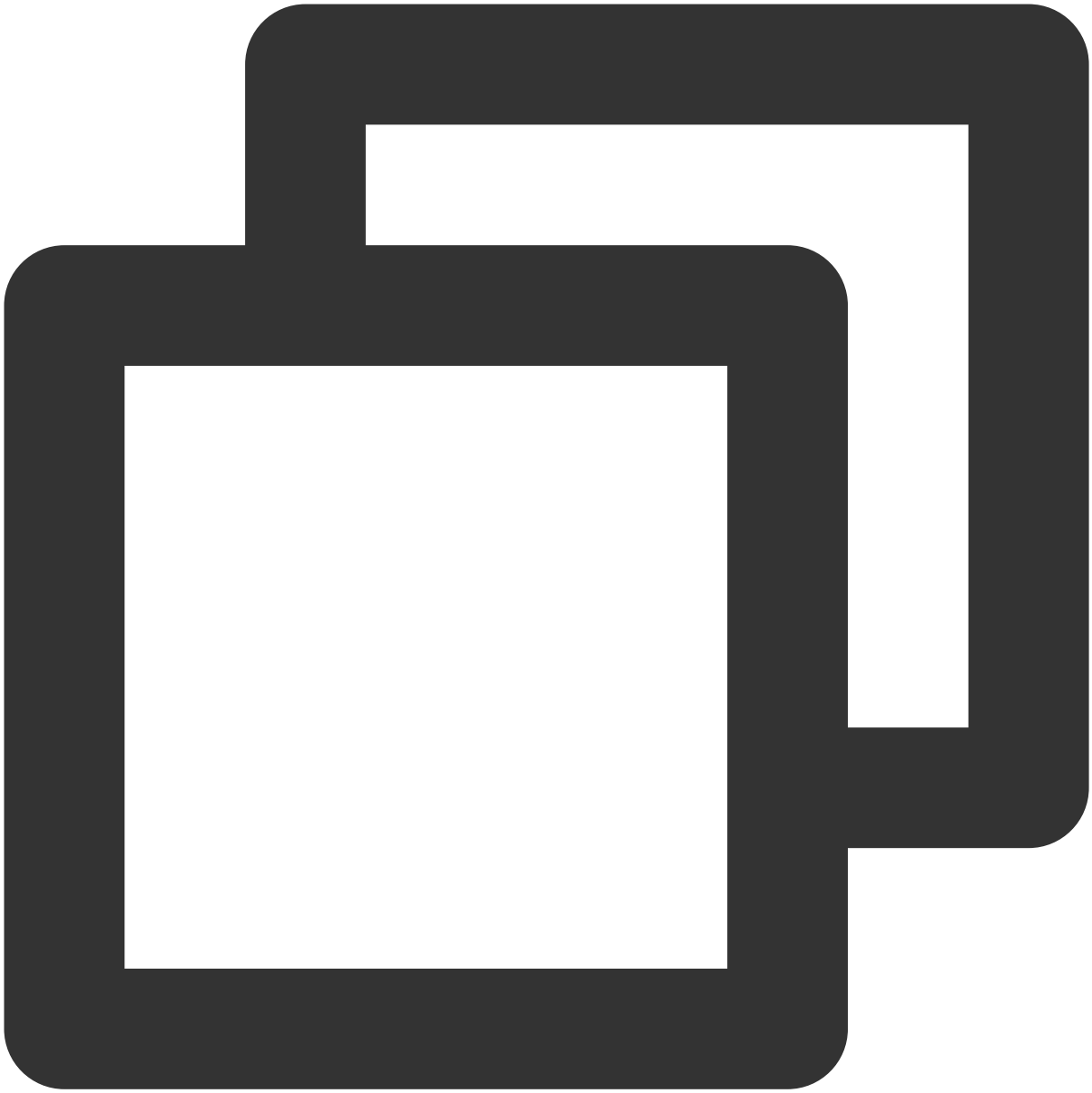
参数名称	描述	类型
endpoint	通过设置域名后缀，使用自定义 CDN 域名	String
is_cname	设置为1，设置为自定义 CDN 域名模式	Int

自定义源站域名

功能说明

在 COS 控制台为存储桶设置自定义源站域名后，可在 SDK 代码中设置对应的域名，以便实现访问 COS 资源。关于如何设置自定义源站域名请参见 自定义源站域名。

示例代码



```
cos_pool_t *p = NULL;
cos_request_options_t *options = NULL;
cos_pool_create(&p, NULL);
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
//请求域名为 your.domain.com
cos_str_set(&options->config->endpoint, "your.domain.com"); //自定义域名
// is_cname设置为1
options->config->is_cname = 1;
```

参数说明

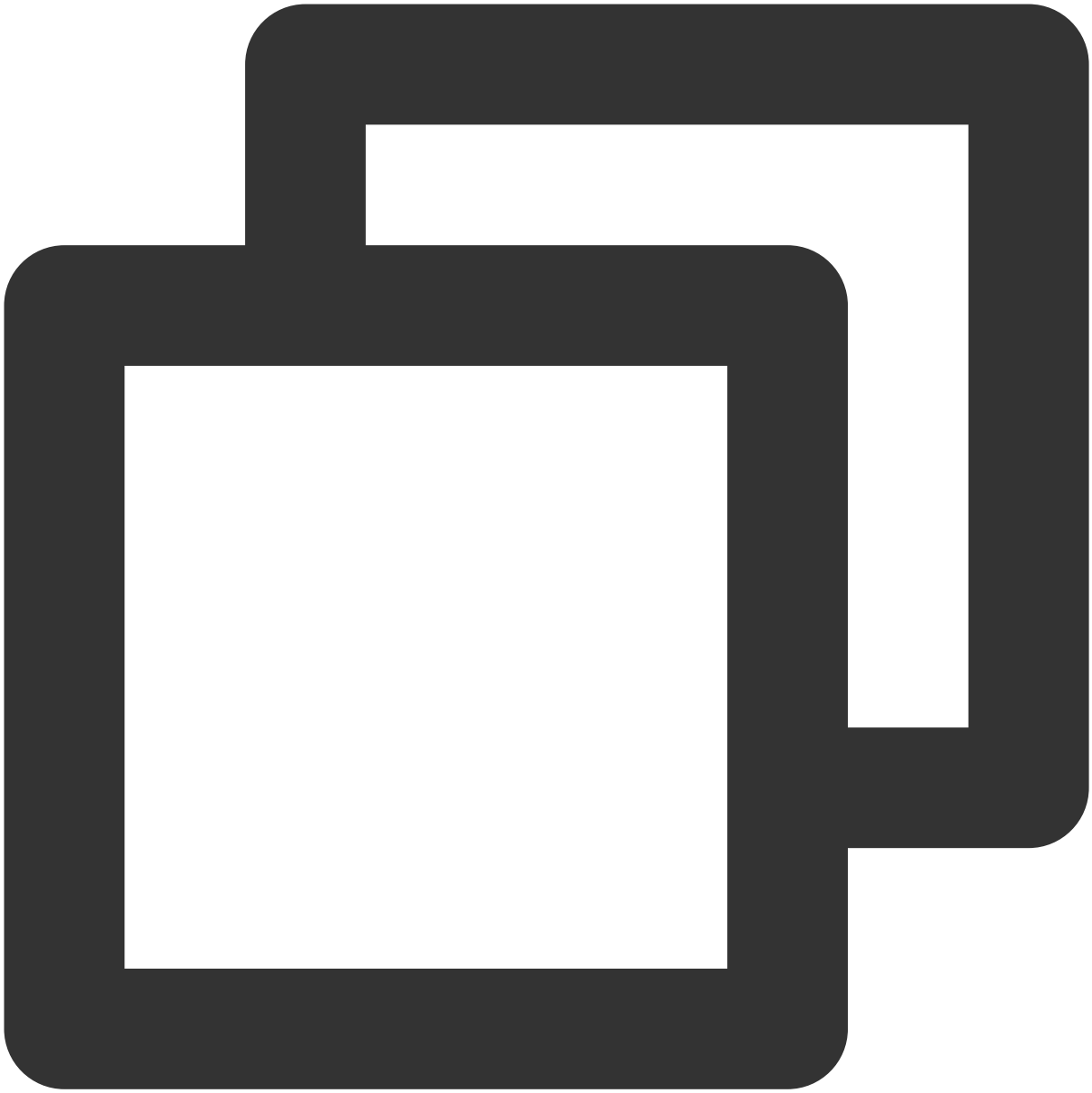
参数名称	描述	类型
endpoint	通过设置域名后缀，使用自定义源站域名	String
is_cname	设置为1，设置为自定义源站域名模式	Int

全球加速域名

功能说明

在 COS 控制台为存储桶开启全球加速域名后，可在 SDK 代码中设置对应的域名，以便实现全球加速功能。关于全球加速的功能介绍，请参见 全球加速功能概述。

示例代码



```
cos_pool_t *p = NULL;
cos_request_options_t *options = NULL;
cos_pool_create(&p, NULL);
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
//请求域名为 cos.accelerate.myqcloud.com
cos_str_set(&options->config->endpoint, "cos.accelerate.myqcloud.com");
```

参数说明

参数名称	描述	类型
endpoint	通过设置域名后缀，使用全球加速域名	String

C++ SDK
快速入门

最近更新时间：2023-04-12 15:21:57

下载与安装

相关资源

- 对象存储 COS 的 XML C++ SDK 源码下载地址：
  Linux 版本/Windows 版本/macOS 版本： XML Linux C++ SDK。
- SDK 快速下载地址：XML C++ SDK。
- 示例 Demo 下载地址：COS XML C++ SDK 示例。
- SDK 更新日志请参见 ChangeLog。

> 说明：
> 如果您在使用 XML 版本 SDK 时遇到函数或方法不存在等错误，请先将 XML 版本 SDK 升级到最新版再重试。

已编译（推荐）

下载 XML C++ SDK 源码，libs 目录中有编译完成的库，可以直接使用，使用时请选择对应的系统版本。

```
libs/linux/libcossdk.a #linux 的静态库，libcossdk.a 是基于 gcc version 4.8.5版本编译的，如果客户编译环境的 gcc 版本不同，需要重新编译 libcossdk.a
libs/linux/libcossdk-shared.so #linux 动态库
libs/Win32/cossdk.lib #Win32库
libs/x64/cossdk.lib #Win64库
libs/macOS/libcossdk.a #macOS 静态库
libs/macOS/libcossdk-shared.dylib #macOS 动态库
```

> 说明：
> 使用时，请将对应系统的库文件和 SDK include 头文件拷贝至您的工程中。

版权所有：腾讯云计算（北京）有限责任公司

Third-party 目录下有第三方依赖库，如下：

```
third_party/lib/linux/poco/ #linux下依赖的 poco 动态库，poco 库是基于 OpenSSL 1.0.2版本编译的，如果客户编译环境的 openssl 版本不同，需要重新编译 poco
third_party/lib/Win32/openssl/ #Win32依赖的 openssl 库
third_party/lib/Win32/poco/ #Win32依赖的 poco 库
third_party/lib/x64/openssl/ #Win64依赖的 openssl 库
third_party/lib/x64/poco/ #Win64依赖的 poco 库
third_party/lib/macOS/poco/ #macOS依赖的 poco 库
```

> 说明：
> 使用时，请将对应系统的依赖库拷贝至您的工程中。

手动编译

编译选项

在根目录下的 CMakeLists.txt 文件可以配置编译选项，有以下编译选项：

```
option(BUILD_UNITTEST "Build unittest" OFF) #配置编译单元测试
option(BUILD_DEMO "Build demo" ON) #配置编译 demo 测试代码
option(BUILD_SHARED_LIB "Build shared library" OFF) #配置编译动态库
```

依赖库

依赖动态库：poco、openssl、crypto。

编译 Linux 版本 SDK

1. 安装编译工具及依赖库

```
yum install -y gcc gcc-c++ make cmake openssl
#cmake 版本要求大于2.8
```

2. 编译 SDK

下载 XML C++ SDK 源码 到您的开发环境，并执行以下命令：

```
cd ${cos-cpp-sdk}
mkdir -p build
cd build
cmake ..
make
```

3. 安装 Poco 库

```
cd ${cos-cpp-sdk}
sh install-libpoco.sh
```

说明：
该脚本将 Poco 动态库安装到 /usr/lib64 目录中，并创建软链接，如果要在生产环境中使用 COS SDK，也需要安装 Poco 库到生产环境中。

4. 测试 demo

说明：
如果不需要测试 demo，可跳过此步骤。

```
cd ${cos-cpp-sdk}
vim demo/cos_demo.cpp #修改 demo 中的存储桶名以及测试代码
vim CMakeLists.txt #修改根目录下的 CMakeLists.txt 中的 BUILD_DEMO 为 ON，开启编译 demo
cd build && make #编译 demo
ls bin/cos_demo #生成的可执行文件在 bin 目录
vim bin/config.json #修改密钥和园区
cd bin && ./cos_demo #运行 demo
```

5. 使用 SDK

编译生成的库文件在 build/lib 目录中，静态库名称为 `libcossdk.a`，动态库名称为 `libcossdk-shared.so`。使用时，请将库拷贝至您的工程中，同时将 include 目录拷贝至您的工程中的 include 路径下。

编译 Windows 版本 SDK

1. 安装 visual studio 2017
   安装 visual studio 2017 开发环境。

2. 安装 CMake 工具
   从 CMake官网 下载 Windows 版本的 CMake 编译工具，并将 `${CMake的安装路径}\bin` 配置在 Windows 的系统环境变量 Path 中。

3. 编译 SDK
   i. 下载 XML Windows C++ SDK 源码 到您的开发环境。
   ii. 打开 Windows 的命令行，cd 到 C++ SDK 的源码目录下，执行以下命令：

```
mkdir build
cd build
cmake .. #生成 Win32 makefile
cmake -G "Visual Studio 15 Win64" .. #生成 Win64 makefile
```

   iii. 使用 visual studio 2017 打开解决方案资源管理器，进行编译。

4. 测试 demo

> 说明：
> 如果不需要测试 demo，可跳过此步骤。

   修改 demo 代码并编译，生成的 cos_demo.exe 在 bin 目录中，修改 bin/config.json 可运行 cos_demo.exe。

5. 使用 SDK
   编译生成的库文件在 build/Release 目录中，静态库名称为 `cossdk.lib` 。使用时，请将库拷贝至您的工程中，同时将 include 目录拷贝至您的工程中的 include 路径下。

编译 Mac 版本 SDK

1. 安装编译工具及依赖库

```
brew install gcc make cmake openssl
```

2. 编译 SDK

下载 XML C++ SDK 源码 到您的开发环境，并执行以下命令：

```
cd ${cos-cpp-sdk}
mkdir -p build
cd build
cmake ..
make
```

3. 安装 Poco 库

Poco 库在 third_party/lib/macOS/poco 目录下，请自行安装。

4. 测试 demo

> 说明：
> 如果不需要测试 demo，可跳过此步骤。

修改 demo 代码并编译，生成的 cos_demo 在 bin 目录中，拷贝 cos-cpp-sdk-v5/demo/config.json 到 bin 目录下，修改 bin/config.json 可运行 cos_demo。

5. 使用 SDK

编译生成的库文件在 build/lib 目录中，静态库名称为 `libcossdk.a`，动态库名称为 `libcossdk-shared.dylib`。使用时，请将库拷贝至您的工程中，同时将 include 目录拷贝至您的工程中的 include 路径下。

> 说明：
> 如果遇到编译问题，可参见C++ SDK 常见问题。

开始使用

下面为您介绍如何使用 COS C++ SDK 完成一个基础操作，如初始化客户端、创建存储桶、查询存储桶列表、上传对象、查询对象列表、下载对象和删除对象。

> 说明：

关于文章中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参见 COS 术语信息。

初始化

注意：

- 建议用户使用子账号密钥 + 环境变量的方式调用 SDK，提高 SDK 使用的安全性。为子账号授权时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

使用临时密钥访问 COS

如果您需要使用临时密钥访问 COS，请参见如下代码：

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"
int main(int argc, char *argv[]) {
// 这里不推荐用配置文件初始化，因为临时密钥 secret_id 和 secret_key 会变化
qcloud_cos::CosConfig config(appid, "secret_id", "secret_key", "region");
// 临时密钥的 secret_id 和 secret_key，临时密钥生成和使用指引参见 https://cloud.tencent.com/document/product/436/14048
config.SetTmpToken("xxx");
qcloud_cos::CosAPI cos(config);
}
```

使用永久密钥访问 COS(不推荐)

如果您需要使用永久密钥访问 COS，请参见如下代码：

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"
int main(int argc, char *argv[]) {
qcloud_cos::CosConfig config("./config.json"); // 用配置文件方式初始化
// 或者直接用构造函数参数初始化
//qcloud_cos::CosConfig config(appid, "secret_id", "secret_key", "region"); // 永久密钥
qcloud_cos::CosAPI cos(config);
}
```

配置文件各字段介绍：

版权所有：腾讯云计算（北京）有限责任公司

```
{
"SecretId":"**********", // sercret_id 替换为用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
"SecretKey":"**********", // sercret_key 替换为用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
"Region":"ap-guangzhou", // 存储桶地域，替换为客户存储桶所在地域，可以在COS控制台指定存储桶的概览页查看存储桶地域信息，参见 https://console.cloud.tencent.com/cos5/bucket/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224
"SignExpiredTime":360, // 签名超时时间，单位s
"ConnectTimeoutInms":6000, // connect 超时时间，单位ms
"ReceiveTimeoutInms":60000, // recv 超时时间，单位ms
"UploadPartSize":10485760, // 上传文件分块大小，1M~5G，默认为10M
"UploadCopyPartSize":20971520, // 上传复制文件分块大小，5M~5G，默认为20M
"UploadThreadPoolSize":5, // 单文件分块上传线程池大小
"DownloadSliceSize":4194304, // 下载文件分块大小
"DownloadThreadPoolSize":5, // 单文件下载线程池大小
"AsynThreadPoolSize":2, // 异步上传下载线程池大小
"LogoutType":1, // 日志输出类型,0:不输出,1:输出到屏幕,2输出到 syslog
"LogLevel":3, // 日志级别:1: ERR, 2: WARN, 3:INFO, 4:DBG
"IsDomainSameToHost":false, // 是否使用专有的 host
"DestDomain":"", // 特定 host
"IsUseIntranet":false, // 是否使用特定ip和端口号
"IntranetAddr":"" // 特定 ip 和端口号,例如“127.0.0.1:80”
}
```

将 SDK 内部日志打印到自定义日志文件

如果您需要将 SDK 内部日志打印到自定义日志文件，特别是 Windows 系统，则参考如下代码：

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"
void TestLogCallback(const std::string& log) {
std::ofstream ofs;
ofs.open("test.log", std::ios_base::app);
ofs << log;
ofs.close();
}
int main(int argc, char** argv) {
qcloud_cos::CosConfig config("./config.json");
config.SetLogCallback(&TestLogCallback);
qcloud_cos::CosAPI cos(config);
}
```

创建存储桶

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"

int main(int argc, char *argv[]) {
// 1. 指定配置文件路径，初始化 CosConfig
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 2. 构造创建存储桶的请求
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 https://console.intl.cloud.tencent.com/cos5/bucket
qcloud_cos::PutBucketReq req(bucket_name);
qcloud_cos::PutBucketResp resp;

// 3. 调用创建存储桶接口
qcloud_cos::CosResult result = cos.PutBucket(req, &resp);

// 4. 处理调用结果
if (result.IsSucc()) {
// 创建成功
} else {
// 创建存储桶失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
}
```

查询存储桶列表

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"

int main(int argc, char *argv[]) {
// 1. 指定配置文件路径，初始化 CosConfig
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
```

```
// 2. 构造查询存储桶列表的请求
qcloud_cos::GetServiceReq req;
qcloud_cos::GetServiceResp resp;
qcloud_cos::CosResult result = cos.GetService(req, &resp);

// 3. 获取响应信息
const qcloud_cos::Owner& owner = resp.GetOwner();
const std::vector<qcloud_cos::Bucket>& buckets = resp.GetBuckets();
std::cout << "owner.m_id=" << owner.m_id << ", owner.display_name=" << owner.m_di
splay_name << std::endl;

for (std::vector<qcloud_cos::Bucket>::const_iterator itr = buckets.begin(); itr !
= buckets.end(); ++itr) {
const qcloud_cos::Bucket& bucket = *itr;
std::cout << "Bucket name=" << bucket.m_name << ", location="
<< bucket.m_location << ", create_date=" << bucket.m_create_date << std::endl;
}

// 4. 处理调用结果
if (result.IsSucc()) {
// 查询存储桶列表成功
} else {
// 查询存储桶列表失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
}
```

上传对象

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"

int main(int argc, char *argv[]) {
// 1. 指定配置文件路径，初始化 CosConfig
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 2. 构造上传文件的请求
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由 buc
```

```
ketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "exampleobject"; //exampleobject 即为对象键（Key），是对象在存储桶中的唯一标识。例如，在对象的访问域名 examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg，替换为用户指定的对象名。
qcloud_cos::PutObjectByFileReq req(bucket_name, object_name, "/path/to/local/file"); // 替换为用户指定的文件路径
//req.SetXCosStorageClass("STANDARD_IA"); // 默认为 STANDARD，可以调用 Set 方法设置存储类型
qcloud_cos::PutObjectByFileResp resp;

// 3. 调用上传文件接口
qcloud_cos::CosResult result = cos.PutObject(req, &resp);

// 4. 处理调用结果
if (result.IsSucc()) {
// 上传文件成功
} else {
// 上传文件失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
}
```

查询对象列表

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"

int main(int argc, char *argv[]) {
// 1. 指定配置文件路径，初始化 CosConfig
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 2. 构造查询对象列表的请求
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 https://console.intl.cloud.tencent.com/cos5/bucket
qcloud_cos::GetBucketReq req(bucket_name);
qcloud_cos::GetBucketResp resp;
qcloud_cos::CosResult result = cos.GetBucket(req, &resp);
```

```
std::vector<qcloud_cos::Content> cotents = resp.GetContents();
for (std::vector<qcloud_cos::Content>::const_iterator itr = cotents.begin(); itr
!= cotents.end(); ++itr) {
const qcloud_cos::Content& content = *itr;
std::cout << "key name=" << content.m_key << ", lastmodified ="
<< content.m_last_modified << ", size=" << content.m_size << std::endl;
}

// 3. 处理调用结果
if (result.IsSucc()) {
// 查询对象列表成功
} else {
// 查询对象列表失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
}
```

下载对象

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"

int main(int argc, char *argv[]) {
// 1. 指定配置文件路径，初始化 CosConfig
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 2. 构造下载对象的请求
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由 buc
ketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 https://console.in
tl.cloud.tencent.com/cos5/bucket
std::string object_name = "exampleobject"; // exampleobject 即为对象键（Key），是对
象在存储桶中的唯一标识。例如，在对象的访问域名 examplebucket-1250000000.cos.ap-guangzho
u.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg，替换为用户指定的对象名。
std::string local_path = "/tmp/exampleobject";
// request 需要提供 appid、bucketname、object,以及本地的路径（包含文件名）
qcloud_cos::GetObjectByFileReq req(bucket_name, object_name, local_path);
qcloud_cos::GetObjectByFileResp resp;
```

```
// 3. 调用下载对象接口
qcloud_cos::CosResult result = cos.GetObject(req, &resp);

// 4. 处理调用结果
if (result.IsSucc()) {
// 下载文件成功
} else {
// 下载文件失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
}
```

删除对象

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"

int main(int argc, char *argv[]) {
// 1. 指定配置文件路径，初始化 CosConfig
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 2. 构造删除对象的请求
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "exampleobject"; // exampleobject 即为对象键（Key），是对象在存储桶中的唯一标识。例如，在对象的访问域名 examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg，替换为用户指定的对象名。。
// 3. 调用删除对象接口
qcloud_cos::DeleteObjectReq req(bucket_name, object_name);
qcloud_cos::DeleteObjectResp resp;
qcloud_cos::CosResult result = cos.DeleteObject(req, &resp);

// 4. 处理调用结果
if (result.IsSucc()) {
// 删除对象成功
} else {
// 删除对象失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
```

```
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
}
```

版权所有：腾讯云计算（北京）有限责任公司

存储桶操作

最近更新时间：2023-04-25 15:26:18

简介

本文档提供关于存储桶基本操作的相关 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Service（List Buckets）	查询存储桶列表	查询指定账号下所有的存储桶列表
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

查询存储桶列表

功能说明

用于查询指定账号下所有的存储桶列表。

方法原型

```
CosResult GetService(const GetServiceReq& request, GetServiceResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket

qcloud_cos::GetServiceReq req;
qcloud_cos::GetServiceResp resp;
qcloud_cos::CosResult result = cos.GetService(req, &resp);
```

```
// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
const qcloud_cos::Owner& owner = resp.GetOwner();
const std::vector<qcloud_cos::Bucket>& buckets = resp.GetBuckets();
std::cout << "owner.m_id=" << owner.m_id << ", owner.display_name=" << owner.m_di
splay_name << std::endl;
for (std::vector<qcloud_cos::Bucket>::const_iterator itr = buckets.begin(); itr !
= buckets.end(); ++itr) {
const qcloud_cos::Bucket& bucket = *itr;
std::cout << "Bucket name=" << bucket.m_name << ", location="
<< bucket.m_location << ", create_date=" << bucket.m_create_date << std::endl;
}
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetService 操作的请求	GetServiceReq	是
resp	GetService 操作的响应	GetServiceResp	是

GetServiceResp 提供以下成员函数，用于获取 Get Service 返回的 XML 格式中的具体内容。

```
// 获取 Bucket 持有者的信息
Owner GetOwner() const;
// 获取所有 Bucket 列表信息
std::vector<Bucket> GetBuckets() const
```

其中 Owner 的定义如下：

```
struct Owner {
std::string m_id; // 存储桶所有者的 ID
std::string m_display_name; // 存储桶所有者的名字信息
};
```

其中 Bucket 的定义如下：

```
// 描述单个 Bucket 的信息
struct Bucket {
std::string m_name; // Bucket 名称
std::string m_location; // Bucket 所在地域
```

版权所有：腾讯云计算（北京）有限责任公司

```
std::string m_create_date; // Bucket 创建时间。ISO8601 格式，例如 2016-11-09T08:46:32.000Z
};
```

创建存储桶

功能说明

创建一个存储桶（PUT Bucket）。

方法原型

```
CosResult PutBucket(const PutBucketReq& req, PutBucketResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket

qcloud_cos::PutBucketReq req(bucket_name);
//创建MAZ存储桶使用下方进行设置
//req.SetMAZBucket();
qcloud_cos::PutBucketResp resp;
qcloud_cos::CosResult result = cos.PutBucket(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutBucket 操作的请求	PutBucketReq	是

参数	参数描述	类型	是否必填
resp	PutBucket 操作的响应	PutBucketResp	是

PutBucketReq 提供以下成员函数：

```
// 定义 Bucket 的 ACL 属性,有效值：private,public-read-write,public-read
// 默认值：private
void SetXCosAcl(const std::string& str);

// 赋予被授权者读的权限.格式：x-cos-grant-read: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXCosGrantRead(const std::string& str);

// 赋予被授权者写的权限,格式：x-cos-grant-write: id=" ",id=" "./
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>",
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXCosGrantWrite(const std::string& str);

// 赋予被授权者读写权限.格式：x-cos-grant-full-control: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>",
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXCosGrantFullControl(const std::string& str);
```

检索存储桶及其权限

功能说明

检索存储桶是否存在且是否有权限访问。

方法原型

```
CosResult HeadBucket(const HeadBucketReq& req, HeadBucketResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由 buc
```

```
ketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket

qcloud_cos::HeadBucketReq req(bucket_name);
qcloud_cos::HeadBucketResp resp;
qcloud_cos::CosResult result = cos.HeadBucket(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	HeadBucket 操作的请求	HeadBucketReq	是
resp	HeadBucket 操作的响应	HeadBucketResp	是

删除存储桶

功能说明

删除指定账号下的空存储桶。

方法原型

```
CosResult DeleteBucket(const DeleteBucketReq& req, DeleteBucketResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket

// DeleteBucketReq 的构造函数需要传入 bucket_name
qcloud_cos::DeleteBucketReq req(bucket_name);
```

```
qcloud_cos::DeleteBucketResp resp;
qcloud_cos::CosResult result = cos.DeleteBucket(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	DeleteBucket 操作的请求	DeleteBucketReq	是
resp	DeleteBucket 操作的响应	DeleteBucketResp	是

C++ SDK 常见问题

最近更新时间：2023-04-25 15:26:19

编译可执行程序的时候提示错误： PocoCrypto.so.64: undefined reference ， 该如何处理？

```
PocoCrypto.so.64: undefined reference to `PEM_write_bio_PrivateKey@libcrypto.so.1
0'
libPocoNetSSL.so.64: undefined reference to `X509_check_host@libcrypto.so.10'
ibPocoCrypto.so.64: undefined reference to `ECDSA_sign@OPENSSL_1.0.1_EC'
libPocoCrypto.so.64: undefined reference to `CRYPTO_set_id_callback@libcrypto.so.
10'
ibPocoCrypto.so.64: undefined reference to `EVP_PKEY_id@libcrypto.so.10'
libPocoNetSSL.so.64: undefined reference to `SSL_get1_session@libssl.so.10'
libPocoNetSSL.so.64: undefined reference to `SSL_get_shutdown@libssl.so.10'
libPocoCrypto.so.64: undefined reference to `EVP_PKEY_set1_RSA@libcrypto.so.10'
libPocoCrypto.so.64: undefined reference to `SSL_load_error_strings@libssl.so.10'
```

这种情况一般是工程里自带的 poco 库的编译依赖的 SSL 版本与客户机器上的版本不一致导致的，需要用户重新编译 poco 库，并替换掉 third_party 里的poco库。

> 注意：
> 此处建议执行以下命令从github下载源码。如果从poco官网下载，请下载COMPLETE版本。

```
wget https://github.com/pocoproject/poco/archive/refs/tags/poco-1.9.4-release.zip
cd poco-poco-1.9.4-release/
./configure --omit=Data/ODBC,Data/MySQL
mkdir my_build
cd my_build
cmake ..
make -j5
```

编译 poco 库的时候无法编译出 PocoNetSSL 库，该如何处理？

一般是因为机器没装 openssl-devel 库。使用如下命令安装。

```
yum install -y openssl-devel
```

编译可执行程序的时候提示错误： undefined reference to qcloud_cos ， 该如何处理？

```
undefined reference to `qcloud_cos::CosConfig::CosConfig(std::__cxx11::basic_stri
ng<char, std::char_traits<char>, std::allocator<char> > const&)
```

这种情况一般是因为工程自带的 libcossdk.a 编译使用的 gcc 版本与客户机器上的 gcc 版本不一致导致的，需要客户重新编译 poco 库和 libcossdk。

C++ SDK 报错请求过期报错请求过期：Request has expired ， 该如何处理？

由于签名过期导致，重新生成签名即可解决；若重新生成签名仍报相同的错误，可以再检查下机器的本地时间是否已设置为当地标准时间。

对象操作

最近更新时间：2023-06-14 14:57:11

简介

本文档提供关于对象的简单操作、分块操作等其他操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象
PUT Object	简单上传对象	上传一个对象至存储桶
HEAD Object	查询对象元数据	查询对象元数据信息
GET Object	下载对象	下载一个对象至本地
PUT Object - Copy	复制对象	复制文件到目标路径
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象
POST Object restore	恢复归档对象	将归档类型的对象取回访问

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传任务
Upload Part	上传分块	分块上传文件
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

版权所有：腾讯云计算（北京）有限责任公司

简单操作

查询对象列表

功能说明

查询存储桶下的部分或者全部对象 。

方法原型

```
CosResult GetBucket(const GetBucketReq& req, GetBucketResp* resp)
```

请求示例1：列举目录下的对象

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket

qcloud_cos::GetBucketReq req(bucket_name);
req.SetPrefix("image/"); // 列举image/目录下的对象
qcloud_cos::GetBucketResp resp;
qcloud_cos::CosResult result = cos.GetBucket(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
std::vector<qcloud_cos::Content> contents = resp.GetContents();
// 遍历列举的对象
for (auto &content: contents) {
// do something
}
} else {
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
```

请求示例2：列举存储桶所有对象

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"

int main(int argc, char *argv[]) {
// 1. 指定配置文件路径，初始化 CosConfig
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 2. 构造查询对象列表的请求
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
qcloud_cos::GetBucketReq req(bucket_name);
qcloud_cos::CosResult result;
bool is_truncated = false;

// 3. 遍历所有对象
do {
qcloud_cos::GetBucketResp resp;
result = cos.GetBucket(req, &resp);
if (result.IsSucc()) {
std::vector<qcloud_cos::Content> contents = resp.GetContents();
for (auto &content: contents) {
// do something
}
req.SetMarker(resp.GetNextMarker()); // 设置下次列举的起始key
is_truncated = resp.IsTruncated();
}
} while (result.IsSucc() && is_truncated);
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetBucket 操作的请求	GetBucketReq	是
resp	GetBucket 操作的响应	GetBucketResp	是

GetBucketReq提供以下成员函数：

```
/// \@brief 设置前缀，用来规定返回的文件前缀地址
void SetPrefix(const std::string& prefix);
/// \brief 设置定界符,限制返回的是以 prefix 开头，并以 delimiter 第一次出现的结束的路径
```

版权所有：腾讯云计算（北京）有限责任公司

```
void SetDelimiter(const std::string& delimiter);
/// \brief 规定返回值的编码方式，可选值：url
void SetEncodingType(const std::string& encoding_type);
/// \brief 默认以 UTF-8 二进制顺序列出条目，所有列出条目从marker开始
void SetMarker(const std::string& marker);
/// \brief 单次返回最大的条目数量，默认1000
void SetMaxKeys(uint64_t max_keys);
```

GetBucketResp 提供以下成员函数:

```
/// \brief 获取Bucket中Object对应的元信息
std::vector<Content> GetContents() const;
/// \brief Bucket名称
std::string GetName() const;
/// \brief 获取定界符
std::string GetDelimiter() const;
/// \brief 编码格式
std::string GetEncodingType() const;
/// \brief 返回的文件前缀
std::string GetPrefix() const;
/// \brief 默认以UTF-8二进制顺序列出条目，所有列出条目从marker开始
std::string GetMarker() const;
/// \brief 单次响应请求内返回结果的最大的条目数量
uint64_t GetMaxKeys() const;
/// \brief 响应请求条目是否被截断，布尔值：true，false
bool IsTruncated() const;
/// \brief 假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点
std::string GetNextMarker() const;
```

其中 Content 的定义如下：

```
struct Content {
std::string m_key; // Object 的 Key
std::string m_last_modified; // Object 最后被修改时间
std::string m_etag; // 文件的 MD-5 算法校验值
std::string m_size; // 文件大小，单位是 Byte
std::vector<std::string> m_owner_ids; // Bucket 持有者信息
std::string m_storage_class; // Object 的存储类别，枚举值：STANDARD，STANDARD_IA
};
```

简单上传对象

功能说明

上传对象到指定的存储桶中。

方法原型

```
/// 通过 Stream 进行上传
CosResult PutObject(const PutObjectByStreamReq& req, PutObjectByStreamResp* resp)

/// 上传本地文件
CosResult PutObject(const PutObjectByFileReq& req, PutObjectByFileResp* resp)
```

请求示例1：流式上传

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "object_name";
std::istringstream iss("put object");
// request 的构造函数中需要传入 istream
qcloud_cos::PutObjectByStreamReq req(bucket_name, object_name, iss);
// 调用 Set 方法设置元数据或者 ACL 等
req.SetXCosStorageClass("STANDARD_IA");
// 关闭MD5校验，开启使用req.TurnOnComputeConentMd5()，默认情况开启
req.TurnOffComputeConentMd5();
qcloud_cos::PutObjectByStreamResp resp;
qcloud_cos::CosResult result = cos.PutObject(req, &resp);

if (result.IsSucc()) {
// 调用成功，调用 resp 的成员函数获取返回内容
} else {
// 调用失败，调用 result 的成员函数获取错误信息
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
```

请求示例2：上传文件

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
```

版权所有：腾讯云计算（北京）有限责任公司

```
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由buck
etname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cl
oud.tencent.com/cos5/bucket
std::string object_name = "object_name";
// request 的构造函数中需要传入本地文件路径
qcloud_cos::PutObjectByFileReq req(bucket_name, object_name, "/path/to/local/fil
e");
// 调用 Set 方法设置元数据或者 ACL 等
req.SetXCosStorageClass("STANDARD_IA");
// 关闭 MD5 校验，开启使用 req.TurnOnComputeConentMd5()，默认情况开启
req.TurnOffComputeConentMd5();
qcloud_cos::PutObjectByFileResp resp;
qcloud_cos::CosResult result = cos.PutObject(req, &resp);
if (result.IsSucc()) {
// 调用成功，调用 resp 的成员函数获取返回内容
} else {
// 调用失败，调用 result 的成员函数获取错误信息
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
```

请求示例3：上传时限速

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由buck
etname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cl
oud.tencent.com/cos5/bucket
std::string object_name = "object_name";
// request 的构造函数中需要传入本地文件路径
qcloud_cos::PutObjectByFileReq req(bucket_name, object_name, "/path/to/local/fil
e");
// 限速值设置范围为819200-838860800，即100KB/s-100MB/s,单位为bit/s
req.SetTrafficLimitByHeader(1048576);
qcloud_cos::PutObjectByFileResp resp;
qcloud_cos::CosResult result = cos.PutObject(req, &resp);
if (result.IsSucc()) {
// 调用成功，调用 resp 的成员函数获取返回内容
} else {
// 调用失败，调用 result 的成员函数获取错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutObject 操作的请求	PutObjectByStreamReq/PutObjectByFileReq	是
resp	PutObject 操作的响应	PutObjectByStreamResp/PutObjectByFileResp	是

参数 Req 包括如下成员函数：

```
// Cache-Control RFC 2616 中定义的缓存策略，将作为 Object 元数据保存
void SetCacheControl(const std::string& str);

// Content-Disposition RFC 2616 中定义的文件名称，将作为 Object 元数据保存
void SetContentDisposition(const std::string& str);

// Content-Encoding RFC 2616 中定义的编码格式，将作为 Object 元数据保存-
void SetContentEncoding(const std::string& str);

// Content-Type RFC 2616 中定义的内容类型（MIME），将作为 Object 元数据保存
void SetContentType(const std::string& str);

// Expect 当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容
void SetExpect(const std::string& str);

// Expires RFC 2616 中定义的过期时间，将作为 Object 元数据保存
void SetExpires(const std::string& str);

// 允许用户自定义的头部信息,将作为 Object 元数据返回.大小限制2K
void SetXCosMeta(const std::string& key, const std::string& value);

// x-cos-storage-class 设置 Object 的存储级别，枚举值：STANDARD,STANDARD_IA, ARCHIVE
// 默认值：STANDARD
void SetXCosStorageClass(const std::string& storage_class);

// 定义 Object 的 ACL 属性,有效值：private,public-read
// 默认值：private
void SetXcosAcl(const std::string& str);

// 赋予被授权者读的权限.格式：x-cos-grant-read: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXcosGrantRead(const std::string& str);

// 赋予被授权者读写权限.格式：x-cos-grant-full-control: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>",
```

```
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXcosGrantFullControl(const std::string& str);

/// 设置Server端加密使用的算法, 目前支持AES256
void SetXCosServerSideEncryption(const std::string& str);
```

参数 Resp 包括如下成员函数：

```
/// 获取Object的版本号, 如果Bucket未开启多版本, 返回空字符串
std::string GetVersionId();

/// Server端加密使用的算法
std::string GetXCosServerSideEncryption();
```

查询对象元数据

功能说明

查询对象元数据信息。

方法原型

```
CosResult HeadObject(const HeadObjectReq& req, HeadObjectResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名, 由bucketname-appid 组成, appid必须填入, 可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "object_name";
qcloud_cos::HeadObjectReq req(bucket_name, object_name);
qcloud_cos::HeadObjectResp resp;
qcloud_cos::CosResult result = cos.HeadObject(req, &resp);
if (result.IsSucc()) {
// 下载成功, 可以调用 HeadObjectResp 的成员函数
} else {
// 下载失败, 可以调用 CosResult 的成员函数输出错误信息, 例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	HeadObject 操作的请求	HeadObjectReq	是
resp	HeadObject 操作的响应	HeadObjectResp	是

HeadObjectResp 除了读取公共头部的成员函数外，还提供以下成员函数：

```
std::string GetXCosObjectType();

std::string GetXCosStorageClass();

// 获取自定义的 meta，参数可以为 x-cos-meta-* 中的 *
std::string GetXCosMeta(const std::string& key);

// 以 map 形式返回所有自定义的 meta，map 的 key 均不包含"x-cos-meta-"前缀
std::map<std::string, std::string> GetXCosMetas();

// 获取 Server 端加密使用的算法
std::string GetXCosServerSideEncryption();
```

下载对象

功能说明

下载对象到本地（Get Object）。

方法原型

```
// 将 Object 下载到本地文件中
CosResult GetObject(const GetObjectByFileReq& req, GetObjectByFileResp* resp)

// 将 Object 下载到流中
CosResult GetObject(const GetObjectByStreamReq& req, GetObjectByStreamResp* resp)

// 将 Object 下载到本地文件中（多线程）
CosResult GetObject(const MultiGetObjectReq& req, MultiGetObjectResp* resp)
```

请求示例1：下载文件到本地

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
```

```
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "object_name";
std::string local_path = "/tmp/object_name";

// request 需要提供 appid、bucketname、object,以及本地的路径（包含文件名）
qcloud_cos::GetObjectByFileReq req(bucket_name, object_name, local_path);
qcloud_cos::GetObjectByFileResp resp;
qcloud_cos::CosResult result = cos.GetObject(req, &resp);
if (result.IsSucc()) {
// 下载成功，可以调用 GetObjectByFileResp 的成员函数
} else {
// 可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

请求示例2：下载文件到流

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "object_name";
std::string local_path = "/tmp/object_name";
// request 需要提供 appid、bucketname、object，以及输出流
std::ostringstream os;
qcloud_cos::GetObjectByStreamReq req(bucket_name, object_name, os);
qcloud_cos::GetObjectByStreamResp resp;
qcloud_cos::CosResult result = cos.GetObject(req, &resp);
if (result.IsSucc()) {
// 下载成功，可以调用 GetObjectByStreamResp 的成员函数
} else {
// 下载失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

请求示例3：多线程下载文件到本地

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "object_name";
std::string local_path = "/tmp/object_name";
```

版权所有：腾讯云计算（北京）有限责任公司

```
// request需要提供 appid、bucketname、object以及本地的路径（包含文件名）
qcloud_cos::MultiGetObjectReq req(bucket_name, object_name, local_path);
qcloud_cos::MultiGetObjectResp resp;
qcloud_cos::CosResult result = cos.GetObject(req, &resp);
if (result.IsSucc()) {
// 下载成功，可以调用 MultiGetObjectResp 的成员函数
} else {
// 下载失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetObject 操作的请求	GetObjectByFileReq/GetObjectByStreamReq/MultiGetObjectReq	是
resp	GetObject 操作的响应	GetObjectByFileResp/GetObjectByStreamResp/MultiGetObjectResp	是

成员函数如下：

```
// 设置响应头部中的 Content-Type 参数
void SetResponseContentType(const std::string& str);

// 设置响应头部中的 Content-Language 参数
void SetResponseContentLang(const std::string& str);

// 设置响应头部中的 Content-Expires 参数
void SetResponseExpires(const std::string& str);

// 设置响应头部中的 Cache-Control 参数
void SetResponseCacheControl(const std::string& str);

// 设置响应头部中的 Content-Disposition 参数
void SetResponseContentDisposition(const std::string& str);

// 设置响应头部中的 Content-Encoding 参数
void SetResponseContentEncoding(const std::string& str);
```

GetObjectResp 除了读取公共头部的成员函数外，还提供以下成员函数：

```
// 获取 Object 最后被修改的时间，字符串格式 Date，类似"Wed, 28 Oct 2014 20:30:00 GMT"
std::string GetLastModified();

// 获取 Object type，表示 Object 是否可以被追加上传，枚举值：normal 或者 appendable
std::string GetXCosObjectType();

// 获取 Object 的存储类别，枚举值：STANDARD，STANDARD_IA
std::string GetXCosStorageClass();

// 以 map 形式返回所有自定义的 meta，map 的 key 均不包含"x-cos-meta-"前缀
std::map<std::string, std::string> GetXCosMetas();

// 获取自定义的 meta，参数可以为 x-cos-meta-*中的*
std::string GetXCosMeta(const std::string& key);

// 获取Server端加密使用的算法
std::string GetXCosServerSideEncryption();
```

复制对象

复制文件到目标路径。

方法原型

```
CosResult PutObjectCopy(const PutObjectCopyReq& req, PutObjectCopyResp* resp)
```

请求示例1：复制对象

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "sevenyou";
qcloud_cos::PutObjectCopyReq req(bucket_name, object_name);
req.SetXCosCopySource("sevenyousouthtest-12345656.cn-south.myqcloud.com/sevenyou_source_obj");
qcloud_cos::PutObjectCopyResp resp;
qcloud_cos::CosResult result = cos.PutObjectCopy(req, &resp);
```

请求示例2：修改对象元数据

```
qcloud_cos::PutObjectCopyReq req(bucket_name, object_name);
req.SetXCosMeta("key1", "val1"); // 自定义元数据
req.SetXCosMeta("key2", "val2"); // 自定义元数据
req.SetXCosMetadataDirective("Replaced"); // 必须加这行
qcloud_cos::PutObjectCopyResp resp;
qcloud_cos::CosResult result = cos.PutObjectCopy(req, &resp);
```

请求示例3：修改对象存储类型

```
qcloud_cos::PutObjectCopyReq req(bucket_name, object_name);
req.SetXCosStorageClass("STANDARD_IA");
req.SetXCosMetadataDirective("Replaced"); // 必须加这行
qcloud_cos::PutObjectCopyResp resp;
qcloud_cos::CosResult result = cos.PutObjectCopy(req, &resp);
```

参数说明

参数	参数描述	类型	是否必填
req	PutObjectCopy 操作的请求	PutObjectCopyReq	是
resp	PutObjectCopy 操作的响应	PutObjectCopyResp	是

PutObjectCopyReq 包含以下成员函数：

```
// 源文件 URL 路径，可以通过 versionid 子资源指定历史版本
void SetXCosCopySource(const std::string& str);

// 是否拷贝元数据，枚举值：Copy，Replaced，默认值 Copy。
// 假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制；
// 假如标记为 Replaced，按 Header 信息修改元数据。
// 当目标路径和原路径一致，即用户试图修改元数据时，必须为 Replaced
void SetXCosMetadataDirective(const std::string& str);

// 当 Object 在指定时间后被修改，则执行操作，否则返回 412。
// 可与 x-cos-copy-source-If-None-Match 一起使用，与其他条件联合使用返回冲突。
void SetXCosCopySourceIfModifiedSince(const std::string& str);

// 当 Object 在指定时间后未被修改，则执行操作，否则返回 412。
// 可与 x-cos-copy-source-If-Match 一起使用，与其他条件联合使用返回冲突。
void SetXCosCopySourceIfUnmodifiedSince(const std::string& str);

// 当 Object 的 Etag 和给定一致时，则执行操作，否则返回 412。
// 可与x-cos-copy-source-If-Unmodified-Since 一起使用，与其他条件联合使用返回冲突
```

```
void SetXCosCopySourceIfMatch(const std::string& str);

// 当 Object 的 Etag 和给定不一致时，则执行操作，否则返回 412。
// 可与 x-cos-copy-source-If-Modified-Since 一起使用，与其他条件联合使用返回冲突。
void SetXCosCopySourceIfNoneMatch(const std::string& str);

// x-cos-storage-class 设置 Object 的存储级别，枚举值：STANDARD,STANDARD_IA
// 默认值：STANDARD
void SetXCosStorageClass(const std::string& storage_class);

// 定义 Object 的 ACL 属性,有效值：private,public-read
// 默认值：private
void SetXCosAcl(const std::string& str);

// 赋予被授权者读的权限。格式：id="[OwnerUin]"
void SetXCosGrantRead(const std::string& str);

// 赋予被授权者所有的权限。格式：id="[OwnerUin]"
void SetXCosGrantFullControl(const std::string& str);

// 允许用户自定义的头部信息,将作为 Object 元数据返回.大小限制2K
void SetXCosMeta(const std::string& key, const std::string& value);

/// 设置 Server 端加密使用的算法，目前支持 AES256
void SetXCosServerSideEncryption(const std::string& str);
```

PutObjectCopyResp 包含以下成员函数：

```
// 返回文件的 MD5 算法校验值。ETag 的值可以用于检查 Object 的内容是否发生变化。
std::string GetEtag();

// 返回文件最后修改时间，GMT 格式
std::string GetLastModified();

// 返回版本号
std::string GetVersionId();

/// Server端加密使用的算法
std::string GetXCosServerSideEncryption();
```

删除单个对象

功能说明

在存储桶中删除指定对象。

方法原型

```
CosResult DeleteObject(const DeleteObjectReq& req, DeleteObjectResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "test_object";

qcloud_cos::DeleteObjectReq req(bucket_name, object_name);
qcloud_cos::DeleteObjectResp resp;
qcloud_cos::CosResult result = cos.DeleteObject(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	DeleteObject 操作的请求	DeleteObjectReq	是
resp	DeleteObject 操作的响应	DeletObjectResp	是

删除多个对象

功能说明

在存储桶中批量删除对象。

方法原型

```
CosResult DeleteObjects(const DeleteObjectsReq& req, DeleteObjectsResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket

std::vector<std::string> objects;
std::vector<ObjectVersionPair> to_be_deleted;
objects.push_back("batch_delete_test_00");
objects.push_back("batch_delete_test_01");
objects.push_back("batch_delete_test_02");
objects.push_back("batch_delete_test_03");
for (size_t idx = 0; idx < objects.size(); ++idx) {
ObjectVersionPair pair;
pair.m_object_name = objects[idx];
to_be_deleted.push_back(pair);
}
qcloud_cos::DeleteObjectsReq req(bucket_name, to_be_deleted);
qcloud_cos::DeleteObjectsResp resp;
qcloud_cos::CosResult result = cos.DeleteObjects(req, &resp);
// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	DeleteObjects 操作的请求	DeleteObjectsReq	是
resp	DeleteObjects 操作的响应	DeletObjectsResp	是

DeleteObjectsReq 包含以下成员函数：

```
// 添加对象，并指定版本
void AddObjectVersion(const std::string& object, const std::string& version)
// 添加对象，非多版本
void AddObject(const std::string& object)
```

DeleteObjectsResp 包含以下成员函数：

```
// 获取删除成功的 objects 信息
std::vector<DeletedInfo> GetDeletedInfos() const

// 获取删除失败的 objects 信息
std::vector<ErrorInfo> GetErrorMsgs() const
```

对应 DeletedInfo 和 ErrorInfo 的结构如下：

```
struct DeletedInfo{
std::string m_key; // object key
std::string m_version_id; //version_id
bool m_delete_marker; // is delete marker
std::string m_delete_marker_version_id; // if delete marker, so version id
}
struct ErrorInfo{
std::string m_key; // object key
std::string m_code; // error code
std::string m_message; // error message
std::string m_version_id; // version id
}
```

恢复归档对象

功能说明

将归档类型的对象取回访问。

方法原型

```
CosResult PostObjectRestore(const PostObjectRestoreReq& req, PostObjectRestoreRes
p* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "sevenyou";

{
qcloud_cos::PostObjectRestoreReq req(bucket_name, object_name);
req.SetExiryDays(30);
req.SetTier("Standard");
qcloud_cos::PostObjectRestoreResp resp;
qcloud_cos::CosResult result = cos.PostObjectRestore(req, &resp);
// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
}
```

参数说明

参数	参数描述	类型	是否必填
req	PostObjectRestore 操作的请求	PostObjectRestoreReq	是
resp	PostObjectRestore 操作的响应	PostObjectRestoreResp	是

PostObjectRestoreReq 包含以下成员函数：

```
// 设置临时副本的过期时间
void SetExiryDays(uint64_t days);

// 枚举值： Expedited , Standard , Bulk；默认值：Standard
void SetTier(const std::string& tier);
```

分块操作

关于分块上传的更多说明请参见 分块上传。分块上传对象可包括的操作：

- 分块上传对象：初始化分块上传，上传分块，完成所有分块上传。
- 分块续传：查询已上传的分块，上传分块，完成所有分块上传。
- 删除已上传分块。

查询分块上传

功能说明

查询指定存储桶中正在进行的分块上传（List Multipart Uploads）。

方法原型

```
CosResult CosAPI::ListMultipartUpload(const ListMultipartUploadReq& request, List
MultipartUploadResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由buck
etname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cl
oud.tencent.com/cos5/bucket
std::string object_name = "test_object";

qcloud_cos::ListMultipartUploadReq req(bucket_name, object_name);
qcloud_cos::ListMultipartUploadResp resp;
qcloud_cos::CosResult result = cos.ListMultipartUpload(req, &resp);
std::vector<qcloud_cos::Upload> rst = resp.GetUpload();

for (std::vector<qcloud_cos::Upload>::const_iterator itr = rst.begin(); itr != rs
t.end(); ++itr) {
const qcloud_cos::Upload& upload = *itr;
std::cout << "key = " << upload.m_key << ", uploadid= " << upload.m_uploadid <<
", storagen class = " << upload.m_storage_class << ", m_initiated= " << upload.m_
initiated << std::endl;
}

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	ListMultipartUpload 操作的请求	ListMultipartUploadReq	是
resp	ListMultipartUpload 操作的响应	ListMultipartUploadResp	是

ListMultipartUploadReq 成员函数：

```
// 限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix。
void SetPrefix(const std::string& prefix);

// 定界符为一个符号，对 Object 名字包含指定前缀且第一次出现 delimiter 字符之间的 Object 作为一组元素：common prefix。如果没有 prefix，则从路径起点开始
void SetDelimiter(const std::string& delimiter);

// 规定返回值的编码格式，合法值：url
void SetEncodingType(const std::string& encoding_type);

// 与 upload-id-marker 一起使用当 upload-id-marker 未被指定时，ObjectName 字母顺序大于 key-marker 的条目将被列出，当 upload-id-marker 被指定时，ObjectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出。
void SetKeyMarker(const std::string& marker);

// 设置最大返回的 multipart 数量，合法取值从1到1000，默认1000
void SetMaxUploads(const std::string& max_uploads);

// 与 key-marker 一起使用，当 key-marker 未被指定时，upload-id-marker 将被忽略，当 key-marker 被指定时，ObjectName字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出。
void SetUploadIdMarker(const std::string& upload_id_marker);
```

ListMultipartUploadResp 成员函数：

```
// 获取Bucket中Object对应的元信息
std::vector<Upload> GetUpload();
// Bucket 名称
std::string GetName();
// 编码格式
std::string GetEncodingType() const;
// 默认以UTF-8二进制顺序列出条目，所有列出条目从marker开始
std::string GetMarker() const;
// 列出条目从该 UploadId 值开始
```

```
std::string GetUploadIdMarker() const;
// 假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点
std::string GetNextKeyMarker() const;
// 假如返回条目被截断，则返回 UploadId 就是下一个条目的起点
std::string GetNextUploadIdMarker() const;
// 最大返回的 multipart 数量，合法取值从0到1000
std::string GetMaxUploads () const;
// 响应请求条目是否被截断，布尔值：true，false
bool IsTruncated();
// 返回的文件前缀
std::string GetPrefix() const;
// 获取定界符
std::string GetDelimiter() const;
// 将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix
std::vector<std::string> GetCommonPrefixes() const
```

初始化分块上传

功能说明

初始化分块上传，获取对应的 uploadId（Initiate Multipart Upload）。

方法原型

```
CosResult InitMultiUpload(const InitMultiUploadReq& req, InitMultiUploadResp* res
p)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由buck
etname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cl
oud.tencent.com/cos5/bucket
std::string object_name = "object_name";

qcloud_cos::InitMultiUploadReq req(bucket_name, object_name);
qcloud_cos::InitMultiUploadResp resp;
qcloud_cos::CosResult result = cos.InitMultiUpload(req, &resp);

std::string upload_id = "";
if (result.IsSucc()) {
upload_id = resp.GetUploadId();
}
```

参数说明

参数	参数描述	类型	是否必填
req	InitMultiUpload 操作的请求	InitMultiUploadReq	是
resp	InitMultiUpload 操作的响应	InitMultiUploadResp	是

InitMultiUploadReq 的成员函数如下：

```
// Cache-Control RFC 2616 中定义的缓存策略，将作为 Object 元数据保存
void SetCacheControl(const std::string& str);

// Content-Disposition RFC 2616 中定义的文件名称，将作为 Object 元数据保存
void SetContentDisposition(const std::string& str);

// Content-Encoding RFC 2616 中定义的编码格式，将作为 Object 元数据保存-
void SetContentEncoding(const std::string& str);

// Content-Type RFC 2616 中定义的内容类型（MIME），将作为 Object 元数据保存
void SetContentType(const std::string& str);

// Expires RFC 2616 中定义的过期时间，将作为 Object 元数据保存
void SetExpires(const std::string& str);

// 允许用户自定义的头部信息,将作为 Object 元数据返回.大小限制2K
void SetXCosMeta(const std::string& key, const std::string& value);

// x-cos-storage-class 设置 Object 的存储级别，枚举值：STANDARD,STANDARD_IA, ARCHIVE
// 默认值：STANDARD
void SetXCosStorageClass(const std::string& storage_class);

// 定义 Object 的 ACL 属性,有效值：private,public-read
// 默认值：private
void SetXcosAcl(const std::string& str);

// 赋予被授权者读的权限.格式：x-cos-grant-read: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXcosGrantRead(const std::string& str);

// 赋予被授权者读写权限.格式：x-cos-grant-full-control: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>",
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXcosGrantFullControl(const std::string& str);
```

```
/// 设置 Server 端加密使用的算法，目前支持 AES256
void SetXCosServerSideEncryption(const std::string& str);
```

当成功执行此请求后，返回的 response 中会包含 bucket、key、uploadId， 分别表示分块上传的目标 Bucket、Object 名称以及后续分块上传所需的编号。

InitMultiUploadResp 的成员函数如下:

```
std::string GetBucket();
std::string GetKey();
std::string GetUploadId();

// Server端加密使用的算法
std::string GetXCosServerSideEncryption();
```

上传分块

上传分块（Upload Part）。

方法原型

```
CosResult UploadPartData(const UploadPartDataReq& request, UploadPartDataResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "test_object";

// 上传第一个分块
{
std::fstream is("demo_5M.part1");
qcloud_cos::UploadPartDataReq req(bucket_name, object_name, upload_id, is);
req.SetPartNumber(1);
// 关闭 MD5 校验，开启使用 req.TurnOnComputeConentMd5()，默认情况开启
req.TurnOffComputeConentMd5();
qcloud_cos::UploadPartDataResp resp;
qcloud_cos::CosResult result = cos.UploadPartData(req, &resp);
```

```
// 上传成功需要记录分块编号以及返回的 ETag
if (result.IsSucc()) {
etags.push_back(resp.GetEtag());
part_numbers.push_back(1);
}
is.close();
}

// 上传第二个分块
{
std::fstream is("demo_5M.part2");
qcloud_cos::UploadPartDataReq req(bucket_name, object_name,
upload_id, is);
req.SetPartNumber(2);
qcloud_cos::UploadPartDataResp resp;
qcloud_cos::CosResult result = cos.UploadPartData(req, &resp);

// 上传成功需要记录分块编号以及返回的 ETag
if (result.IsSucc()) {
etags.push_back(resp.GetEtag());
part_numbers.push_back(2);
}
is.close();
}
```

参数说明

参数	参数描述	类型	是否必填
req	UploadPartData 操作的请求	UploadPartDataReq	是
resp	UploadPartData 操作的响应	UploadPartDataResp	是

UploadPartDataReq 在构造时，需要指明请求的 APPID、Bucket、Object、初始化成功后获取的 UploadId，以及上传的数据流（调用完成后，流由调用方自己负责关闭）。

```
UploadPartDataReq(const std::string& bucket_name,
const std::string& object_name, const std::string& upload_id,
std::istream& in_stream);
```

此外，请求还需要设置分块编号, 这个分块在完成分块上传时也会用到。

```
void SetPartNumber(uint64_t part_number);
```

UploadPartDataResp 的成员函数如下：

```
/// Server 端加密使用的算法
std::string GetXCosServerSideEncryption();
```

复制分块

将其他对象复制为一个分块。

方法原型

```
CosResult UploadPartCopyData(const UploadPartCopyDataReq& request,UploadPartCopyD
ataResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由buck
etname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cl
oud.tencent.com/cos5/bucket
std::string object_name = "test_object";

std::string upload_id;
std::vector<uint64_t> numbers;
std::vector<std::string> etags;
std::string etag1 = "", etag2 = "";
InitMultiUpload(cos, bucket_name, object_name, &upload_id);

// First part
qcloud_cos::UploadPartCopyDataReq req(bucket_name, object_name, upload_id, 1);
req.SetXCosCopySource("sevenyousouth-1251668577.cos.ap-guangzhou.myqcloud.com/sev
en_10G.tmp");
req.SetXCosCopySourceRange("bytes=0-1048576000");
qcloud_cos::UploadPartCopyDataResp resp;
qcloud_cos::CosResult result = cos.UploadPartCopyData(req, &resp);
if (result.IsSucc()) {
etag1 = resp.GetEtag();
```

版权所有：腾讯云计算（北京）有限责任公司

```
}
numbers.push_back(1);
etags.push_back(etag1);

// Second part
qcloud_cos::UploadPartCopyDataReq req2(bucket_name, object_name, upload_id, 2);
req2.SetXCosCopySource("sevenyoutest-7319456.cos.cn-north.myqcloud.com/sevenyou_2G_part");
req2.SetXCosCopySourceRange("bytes=1048576000-2097152000");
qcloud_cos::UploadPartCopyDataResp resp2;
qcloud_cos::CosResult result = cos.UploadPartCopyData(req2, &resp2);
if (result.IsSucc()) {
etag2 = resp2.GetEtag();
}
numbers.push_back(2);
etags.push_back(etag2);

CompleteMultiUpload(cos, bucket_name, object_name, upload_id, etags, numbers);
```

参数说明

参数	参数描述	类型	是否必填
req	UploadPartCopyData 操作的请求	UploadPartCopyDataReq	是
resp	UploadPartCopyData 操作的响应	UploadPartCopyDataResp	是

```
/// 设置本次分块复制的 ID
void SetUploadId(const std::string& upload_id)
/// 设置本次分块复制的编号
void SetPartNumber(uint64_t part_number)
/// 设置本次分块复制的源文件 URL 路径，可以通过 versionid 子资源指定历史版本
void SetXCosCopySource(const std::string& src)
/// 设置源文件的字节范围，范围值必须使用 bytes=first-last 格式。
void SetXCosCopySourceRange(const std::string& range)
/// 当 Object 在指定时间后被修改，则执行操作，否则返回 412
void SetXCosCopySourceIfModifiedSince(const std::string& date)
/// 当 Object 在指定时间后未被修改，则执行操作，否则返回 412
void SetXCosCopySourceIfUnmodifiedSince(const std::string& date)
/// 当 Object 的 Etag 和给定一致时，则执行操作，否则返回 412
void SetXCosCopySourceIfMatch(const std::string& etag)
/// 当 Object 的 Etag 和给定不一致时，则执行操作，否则返回 412
void SetXCosCopySourceIfNoneMatch(const std::string& etag)
```

```
/// 获取返回文件的MD5算法校验值。
std::string GetEtag() const
/// 返回文件最后修改时间，GMT 格式
std::string GetLastModified() const
/// Server端加密使用的算法
std::string GetXCosServerSideEncryption() const
```

查询已上传块

功能说明

查询特定分块上传操作中的已上传的块。

方法原型

```
CosResult ListParts(const ListPartsReq& req, ListPartsResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "test_object";

// uploadId 是调用 InitMultiUpload 后获取的
qcloud_cos::ListPartsReq req(bucket_name, object_name, upload_id);
req.SetMaxParts(1);
req.SetPartNumberMarker("1");
qcloud_cos::ListPartsResp resp;
qcloud_cos::CosResult result = cos.ListParts(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	ListParts 操作的请求	ListPartsReq	是
resp	ListParts 操作的响应	ListPartsResp	是

ListPartsReq 包含以下成员函数：

```
// 构造函数，Bucket 名、Object 名、分块上传的 ID
ListPartsReq(const std::string& bucket_name,
const std::string& object_name,
const std::string& upload_id);

// \brief 规定返回值的编码方式
void SetEncodingType(const std::string& encoding_type);

// \brief 单次返回最大的条目数量，若不设置，默认1000
void SetMaxParts(uint64_t max_parts);

// \brief 默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始
void SetPartNumberMarker(const std::string& part_number_marker);
```

ListPartsResp 包含以下成员函数：

```
// 分块上传的目标 Bucket
std::string GetBucket();

// 规定返回值的编码方式
std::string GetEncodingType();

// Object 的名称
std::string GetKey();

// 标识本次分块上传的 ID
std::string GetUploadId();

// 用来表示本次上传发起者的信息
Initiator GetInitiator();

// 用来表示这些分块所有者的信息
Owner GetOwner();

// 默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始
uint64_t GetPartNumberMarker();
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 返回每一个块的信息
std::vector<Part> GetParts();

// 假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点
uint64_t GetNextPartNumberMarker();

// 用来表示这些分块的存储级别，枚举值：Standard, Standard_IA, ARCHIVE
std::string GetStorageClass();

// 单次返回最大的条目数量
uint64_t GetMaxParts();

// 返回条目是否被截断，布尔值：TRUE, FALSE
bool IsTruncated();
```

其中 Part、Owner、Initiator 的定义如下：

```
struct Initiator {
std::string m_id; // 创建者的一个唯一标识
std::string m_display_name; // 创建者的用户名描述
};

struct Owner {
std::string m_id; // 用户的一个唯一标识
std::string m_display_name; // 用户名描述
};

struct Part {
uint64_t m_part_num; // 块的编号
uint64_t m_size; // 块大小，单位 Byte
std::string m_etag; // Object 块的 MD5 算法校验值
std::string m_last_modified; // 块最后修改时间
};
```

完成分块上传

功能说明

完成整个文件的分块上传。

方法原型

```
CosResult CompleteMultiUpload(const CompleteMultiUploadReq& request, CompleteMult
iUploadResp* response)
```

请求示例

```
qcloud_cos::CompleteMultiUploadReq req(bucket_name, object_name, upload_id);
qcloud_cos::CompleteMultiUploadResp resp;
req.SetEtags(etags);
req.SetPartNumbers(part_numbers);

qcloud_cos::CosResult result = cos.CompleteMultiUpload(req, &resp);
```

参数说明

参数	参数描述	类型	是否必填
req	CompleteMultiUpload 操作的请求	CompleteMultiUploadReq	是
resp	CompleteMultiUpload 操作的响应	CompleteMultiUploadResp	是

CompleteMultiUploadReq 在构造时，需要指明请求的 APPID、Bucket、Object、初始化成功后获取的 UploadId。

```
CompleteMultiUploadReq(const std::string& bucket_name,
const std::string& object_name, const std::string& upload_id)
```

此外，request 还需要设置所有上传的分块编号和 ETag。

```
// 调用下列方法时，应注意编号和 ETag 的顺序必须一一对应
void SetPartNumbers(const std::vector<uint64_t>& part_numbers);
void SetEtags(const std::vector<std::string>& etags) ;

// 添加 part_number 和 ETag 对
void AddPartEtagPair(uint64_t part_number, const std::string& etag);

/// 设置 Server 端加密使用的算法，目前支持 AES256
void SetXCosServerSideEncryption(const std::string& str);
```

CompleteMultiUploadResp 的返回内容中包括 Location、Bucket、Key、ETag，分别表示创建的 Object 的外网访问域名、分块上传的目标 Bucket、Object 的名称、合并后文件的 MD5 算法校验值。可以调用下列成员函数对 response 中的内容进行访问。

版权所有：腾讯云计算（北京）有限责任公司

```
std::string GetLocation();
std::string GetKey();
std::string GetBucket();
std::string GetEtag();

// Server端加密使用的算法
std::string GetXCosServerSideEncryption();
```

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块。

方法原型

```
CosResult AbortMultiUpload(const AbortMultiUploadReq& request, AbortMultiUploadResp* response)
```

请求示例

```
qcloud_cos::AbortMultiUploadReq req(bucket_name, object_name, upload_id);
qcloud_cos::AbortMultiUploadResp resp;
qcloud_cos::CosResult result = cos.AbortMultiUpload(req, &resp);
```

参数说明

参数	参数描述	类型	是否必填
req	AbortMultiUpload 操作的请求	AbortMultiUploadReq	是
resp	AbortMultiUpload 操作的响应	AbortMultiUploadResp	是

AbortMultiUploadReq 需要在构造的时候指明 Bucket、Object 以及 Upload_id。

```
AbortMultiUploadReq(const std::string& bucket_name,
const std::string& object_name, const std::string& upload_id);
```

无特殊方法，可调用 BaseResp 的成员函数来获取公共头部内容。

高级接口（推荐）

复合上传

功能说明

封装分块上传各接口，多线程并发上传。

方法原型

```
CosResult MultiUploadObject(const MultiUploadObjectReq& request, MultiUploadObjectResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "exampleobject";
std::string local_file = "./test"

qcloud_cos::MultiUploadObjectReq req(bucket_name, object_name, local_file);
// Complete 接口内部 chunk 保活，建议设置较长时间的 timeout。
req.SetRecvTimeoutInms(1000 * 60);
qcloud_cos::MultiUploadObjectResp resp;
qcloud_cos::CosResult result = cos.MultiUploadObject(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	MultiUploadObject 操作的请求	MultiUploadObjectReq	是
resp	MultiUploadObject 操作的响应	MultiUploadObjectResp	是

版权所有：腾讯云计算（北京）有限责任公司

MultiUploadObjectReq 包含以下成员函数：

```
// 设置分块大小，若小于1M，则按1M计算；若大于5G，则按5G计算
void SetPartSize(uint64_t bytes)
// 允许用户自定义的头部信息，将作为 Object 元数据返回，大小限制2K
void SetXCosMeta(const std::string& key, const std::string& value)
// 设置 Server 端加密使用的算法，目前支持 AES256
void SetXCosServerSideEncryption(const std::string& str)
// 设置内部线程池大小
void SetThreadPoolSize(int size)
```

MultiUploadObjectResp 包含以下成员函数：

```
std::string GetRespTag()
/// Server 端加密使用的算法
std::string GetXCosServerSideEncryption() const
```

复合下载

功能说明

多线程并发 Range 下载。

方法原型

```
CosResult GetObject(const MultiGetObjectReq& request, MultiGetObjectResp* respons
e)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由buck
etname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cl
oud.tencent.com/cos5/bucket
std::string object_name = "exampleobject";
std::string file_path = "./test";

qcloud_cos::MultiGetObjectReq req(bucket_name, object_name, file_path);
qcloud_cos::MultiGetObjectResp resp;
qcloud_cos::CosResult result = cos.GetObject(req, &resp);
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	MultiGetObject 操作的请求	MultiGetObjectReq	是
resp	MultiGetObject 操作的响应	MultiGetObjectResp	是

MultiGetObjectReq 包含以下成员函数：

```
// 设置分块大小
void SetSliceSize(uint64_t bytes)
// 设置线程池大小
void SetThreadPoolSize(int size)
```

MultiGetObjectResp 包含以下成员函数：

```
/// Server 端加密使用的算法
std::string GetXCosServerSideEncryption() const
```

断点下载

功能说明

并发 Range 下载，支持断点下载。

方法原型

```
CosResult ResumableGetObject(const MultiGetObjectReq& request, MultiGetObjectResp
* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "exampleobject";
std::string file_path = "./test";

qcloud_cos::MultiGetObjectReq req(bucket_name, object_name, file_path);
qcloud_cos::MultiGetObjectResp resp;
qcloud_cos::CosResult result = cos.ResumableGetObject(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	MultiGetObject 操作的请求	MultiGetObjectReq	是
resp	MultiGetObject 操作的响应	MultiGetObjectResp	是

批量下载（从 COS 下载目录）

功能说明

该示例展示通过组合 SDK 的基本接口，完成批量下载 COS 目录中的文件到本地磁盘。

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存
std::string server_dir_name = "image/"; // cos目录
std::string local_dir_name = "/data/"; // 本地目录

qcloud_cos::GetBucketReq get_bucket_req(bucket_name);
```

```
get_bucket_req.SetPrefix(server_dir_name);
qcloud_cos::CosResult get_bucket_result;
bool is_truncated = false;

do {
qcloud_cos::GetBucketResp get_bucket_resp;
// 列举对象
get_bucket_result = cos.GetBucket(get_bucket_req, &get_bucket_resp);
if (get_bucket_result.IsSucc()) {
std::vector<Content> contents = get_bucket_resp.GetContents();
for (auto& content : contents) {
std::cout << "====process " << content.m_key << std::endl;
std::string local_file_name = local_dir_name + content.m_key;

if (StringUtil::StringEndsWith(local_file_name, "/")) {
continue;
}

// 创建本地目录
size_t found_dir = local_file_name.find_last_of("/");
if (found_dir) {
std::string dirname = local_file_name.substr(0, found_dir);
char* p_dirname = const_cast<char*>(dirname.c_str());
struct stat buffer;
if (stat(p_dirname, &buffer) != 0) {
std::cout << "====mkdir " << dirname << std::endl;
std::string mkdir_cmd = "mkdir -p " + dirname;
system(mkdir_cmd.c_str());
}
}
// 下载对象
GetObjectByFileReq get_req(bucket_name, content.m_key, local_file_name);
GetObjectByFileResp get_resp;
CosResult get_result = cos.GetObject(get_req, &get_resp);
if (get_result.IsSucc()) {
std::cout << "====download " << content.m_key << " to "
<< local_file_name << " succeed" << std::endl;
} else {
std::cout << "====download " << content.m_key << " to "
<< local_file_name << " failed" << std::endl;
}
}
get_bucket_req.SetMarker(
get_bucket_resp.GetNextMarker()); // 设置下次列举的起始key
is_truncated = get_bucket_resp.IsTruncated();
}
} while (get_bucket_result.IsSucc() && is_truncated);
```

异地容灾
版本控制

最近更新时间：2021-01-13 19:32:53

简介

本文档提供关于版本控制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

设置版本控制

功能说明

设置指定存储桶的版本控制功能（PUT Bucket versioning）。

方法原型

```
CosResult BucketOp::PutBucketVersioning(const PutBucketVersioningReq& req, PutBuc
ketVersioningResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::PutBucketVersioningReq req(bucket_name);
qcloud_cos::PutBucketVersioningResp resp;

// 开启版本控制
req.SetStatus(true);

qcloud_cos::CosResult result = cos.PutBucketVersioning(req, &resp);

if (result.IsSucc()) {
// 请求成功
```

```
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutBucketVersioning 操作的请求	PutBucketVersioningReq	是
resp	PutBucketVersioning 操作的响应	PutBucketVersioningResp	是

PutBucketVersioningReq 提供了如下方法开启或暂停版本控制：

```
void SetStatus(bool is_enable);
```

查询版本控制

功能说明

查询指定存储桶的版本控制信息（GET Bucket versioning）。

方法原型

```
CosResult CosAPI::GetBucketVersioning(const GetBucketVersioningReq& request, GetB
ucketVersioningResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::GetBucketVersioningReq req(bucket_name);
qcloud_cos::GetBucketVersioningResp resp;

qcloud_cos::CosResult result = cos.GetBucketVersioning(req, &resp);

if (result.IsSucc()) {
// 请求成功，通过resp的方法获取版本控制状态
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

版权所有：腾讯云计算（北京）有限责任公司

参数说明

参数	参数描述	类型	是否必填
req	GetBucketVersioning 操作的请求	GetBucketVersioningReq	是
resp	GetBucketVersioning 操作的响应	GetBucketVersioningResp	是

GetBucketVersioningResp 提供以下方法获取版本控制状态：

```
int GetStatus() const;
```

存储桶复制

最近更新时间：2021-01-05 16:04:02

简介

本文档提供关于存储桶复制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket replication	设置存储桶复制	对已启用版本控制的存储桶配置存储桶复制规则
GET Bucket replication	查询存储桶复制	查询存储桶的存储桶复制配置信息
DELETE Bucket replication	删除存储桶复制	删除存储桶的存储桶复制配置信息

设置存储桶复制

功能说明

设置指定存储桶的存储桶复制规则（PUT Bucket replication）。

方法原型

```
CosResult PutBucketReplication(const PutBucketReplicationReq& request, PutBucketReplicationResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::PutBucketReplicationReq req(bucket_name);
qcloud_cos::PutBucketReplicationResp resp;

req.SetRole("qcs::cam::uin/100000000001:uin/100000000001"); // 设置发起者身份
qcloud_cos::ReplicationRule rule("", "qcs::cos:ap-guangzhou:uid/1250000000:destinationbucket-1250000000",
"", "", true); // 设置存储桶复制规则
req.AddReplicationRule(rule);

qcloud_cos::CosResult result = cos.PutBucketReplication(req, &resp);
```

```
if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutBucketReplication 操作的请求	PutBucketReplicationReq	是
resp	PutBucketReplication 操作的响应	PutBucketReplicationResp	是

PutBucketReplicationReq 提供以下方法设置存储桶复制相关信息：

```
void SetRole(const std::string& role)
void AddReplicationRule(const ReplicationRule& rule)
void SetReplicationRule(const std::vector<ReplicationRule>& rules)
```

ReplicationRule 的定义如下：

```
struct ReplicationRule {
bool m_is_enable;
std::string m_id; // 用来标注具体 Rule 的名称
std::string m_prefix; // 前缀匹配策略，不可重叠，重叠返回错误
std::string m_dest_bucket; // 目标存储桶信息
std::string m_dest_storage_class; // 存储类型，枚举值：STANDARD，STANDARD_IA
```

查询存储桶复制

功能说明

查询指定存储桶的存储桶复制规则（GET Bucket replication）。

方法原型

```
CosResult GetBucketReplication(const GetBucketReplicationReq& request, GetBucketR
eplicationResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::GetBucketReplicationReq req(bucket_name);
qcloud_cos::GetBucketReplicationResp resp;

qcloud_cos::CosResult result = cos.GetBucketReplication(req, &resp);

if (result.IsSucc()) {
// 请求成功，调用 resp 方法获取存储桶复制规则
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetBucketReplication 操作的请求	GetBucketReplicationReq	是
resp	GetBucketReplication 操作的响应	GetBucketReplicationResp	是

GetBucketReplicationResp 提供以下方法获取存储桶复制规则：

```
std::string GetRole() const;
std::vector<ReplicationRule> GetRules() const;
```

删除存储桶复制

功能说明

删除指定存储桶的存储桶复制规则（DELETE Bucket replication）。

方法原型

```
CosResult DeleteBucketReplication(const DeleteBucketReplicationReq& request, Dele
teBucketReplicationResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
```

```
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::DeleteBucketReplicationReq req(bucket_name);
qcloud_cos::DeleteBucketReplicationResp resp;

qcloud_cos::CosResult result = cos.DeleteBucketReplication(req, &resp);

if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	DeleteBucketReplication 操作的请求	DeleteBucketReplicationReq	是
resp	DeleteBucketReplication 操作的响应	DeleteBucketReplicationResp	是

数据管理
生命周期

最近更新时间：2021-01-05 16:04:01

简介

本文档提供关于生命周期的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket lifecycle	设置生命周期	设置存储桶生命周期管理的配置
GET Bucket lifecycle	查询生命周期	查询存储桶生命周期管理的配置
DELETE Bucket lifecycle	删除生命周期	删除存储桶生命周期管理的配置

设置生命周期

功能说明

设置指定存储桶的生命周期配置信息（PUT Bucket lifecycle）。

方法原型

```
CosResult CosAPI::PutBucketLifecycle(const PutBucketLifecycleReq& request, PutBucketLifecycleResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::PutBucketLifecycleReq req(bucket_name);
qcloud_cos::PutBucketLifecycleResp resp;

qcloud_cos::LifecycleRule rule;
rule.SetIsEnable(true); // 指明规则是否启用，枚举值：Enabled，Disabled
rule.SetId("lifecycle_rule00"); // 用于唯一地标识规则，长度不能超过255个字符

qcloud_cos::LifecycleFilter filter;
```

```
filter.SetPrefix("test");
rule.SetFilter(filter); // 指定规则所适用的前缀。匹配前缀的对象受该规则影响，Prefix最多只能有一个

qcloud_cos::LifecycleTransition transition; // 规则转换属性
transition.SetDays(30);
transition.SetStorageClass("Standard_IA");
rule.AddTransition(transition);
req.AddRule(rule);

qcloud_cos::CosResult result = cos.GetBucketWebsite(req, &resp);

if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutBucketLifecycle 操作的请求	PutBucketLifecycleReq	是
resp	PutBucketLifecycle 操作的响应	PutBucketLifecycleResp	是

PutBucketLifecycleReq 提供了如下方法设置生命周期规则：

```
void AddRule(const LifecycleRule& rule);
void SetRule(const std::vector<LifecycleRule>& rules);
```

说明：

LifecycleRule，LifecycleFilter，LifecycleTransition，LifecycleExpiration，LifecycleNonCurrTransition，LifecycleNonCurrExpiration，AbortIncompleteMultipartUpload 等类的定义，请参考 SDK 头文件 `include/cos_defines.h`

查询生命周期

功能说明

查询存储桶的生命周期管理配置（GET Bucket lifecycle）。

方法原型

```
CosResult CosAPI::GetBucketLifecycle(const GetBucketLifecycleReq& request, GetBucketLifecycleResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::GetBucketLifecycleReq req(bucket_name);
qcloud_cos::GetBucketLifecycleResp resp;

qcloud_cos::CosResult result = cos.GetBucketLifecycle(req, &resp);

if (result.IsSucc()) {
// 请求成功，通过resp获取生命周期规则
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetBucketLifecycle 操作的请求	GetBucketLifecycleReq	是
resp	GetBucketLifecycle 操作的响应	GetBucketLifecycleResp	是

GetBucketLifecycleResp 提供如下方法获取生命周期规则：

```
std::vector<LifecycleRule> GetRules() const;
```

删除生命周期

功能说明

删除存储桶生命周期管理的配置（DELETE Bucket lifecycle）。

方法原型

```
CosResult CosAPI::DeleteBucketLifecycle(const DeleteBucketLifecycleReq& request,
DeleteBucketLifecycleResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::DeleteBucketLifecycleReq req(bucket_name);
qcloud_cos::DeleteBucketLifecycleResp resp;

qcloud_cos::CosResult result = cos.DeleteBucketLifecycle(req, &resp);

if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	DeletBucketLifecycle 操作的请求	DelettBucketLifecycleReq	是
resp	DeletBucketLifecycle 操作的响应	DeletBucketLifecycleResp	是

日志管理

最近更新时间：2021-01-05 16:04:01

简介

本文档提供关于日志管理的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket logging	设置日志管理	为源存储桶开启日志记录
GET Bucket logging	查询日志管理	查询源存储桶的日志配置信息

设置日志管理

功能说明

PUT Bucket logging 用于为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中。

方法原型

```
CosResult CosAPI::PutBucketLogging(const PutBucketLoggingReq& request, PutBucketLoggingResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::PutBucketLoggingReq req(bucket_name);
qcloud_cos::PutBucketLoggingResp resp;

qcloud_cos::LoggingEnabled rules;
rules.SetTargetBucket(TargetBucketname); // 存放日志的目标存储桶
rules.SetTargetPrefix(TargetPrefix); // 日志存放在目标存储桶的指定路径
req.SetLoggingEnabled(rules);

qcloud_cos::CosResult result = cos.PutBucketLogging(req, &resp);

if (result.IsSucc()) {
// 请求成功，通过resp获取静态网站配置
```

```
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutBucketLogginge 操作的请求	PutBucketLoggingReq	是
resp	PutBucketLogging 操作的响应	PutBucketLoggingResp	是

PutBucketLoggingReq 提供了如下方法设置存储桶日志的配置信息：

```
void SetLoggingEnabled(const LoggingEnabled& rules);
```

LoggingEnabledt 提供如下方法设置配置信息：

```
class LoggingEnabled {
public:
void SetTargetBucket(const std::string &targetbucket);
void SetTargetPrefix(const std::string &targetprefix);
```

查询日志管理

功能说明

GET Bucket logging 用于查询指定存储桶的日志配置信息。

方法原型

```
CosResult CosAPI::GetBucketLogging(const GetBucketLoggingReq& request, GetBucketLoggingResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::GetBucketLoggingReq req(bucket_name);
qcloud_cos::GetBucketLoggingResp resp;
```

版权所有：腾讯云计算（北京）有限责任公司

```
qcloud_cos::CosResult result = cos.GetBucketLogging(req, &resp);

if (result.IsSucc()) {
// 请求成功，通过resp获取存储桶日志信息
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetBucketLogginge 操作的请求	GetBucketLoggingReq	是
resp	GetBucketLogging 操作的响应	GetBucketLoggingResp	是

GetBucketLoggingResp 提供如下方法获取存储桶日志配置信息：

```
LoggingEnabled GetLoggingEnabled() const;
```

存储桶标签

最近更新时间：2021-01-05 16:04:01

简介

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket tagging	设置存储桶标签	为已存在的存储桶设置标签
GET Bucket tagging	查询存储桶标签	查询指定存储桶下已有的存储桶标签
DELETE Bucket tagging	删除存储桶标签	删除指定的存储桶标签

设置存储桶标签

功能说明

PUT Bucket tagging 用于为已存在的存储桶设置标签。

方法原型

```
CosResult PutBucketTagging(const PutBucketTaggingReq& request, PutBucketTaggingResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::PutBucketTaggingReq req(bucket_name);
qcloud_cos::PutBucketTaggingResp resp;

std::vector<Tag> tagset;
Tag tag1;
tag1.SetKey("age");
tag1.SetValue("19");

Tag tag2;
tag2.SetKey("name");
```

版权所有：腾讯云计算（北京）有限责任公司

```
tag2.SetValue("xiaoming");
tagset.push_back(tag1);
tagset.push_back(tag2);
req.SetTagSet(tagset);

qcloud_cos::CosResult result = cos.PutBucketTagging(req, &resp);

if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutBucketTagging 操作的请求	PutBucketTaggingReq	是
resp	PutBucketTagging 操作的响应	PutBucketTaggingResp	是

PutBucketTaggingReq 提供以下方法：

```
void SetTagSet(std::vector<Tag>& tagset) // 设置 tagging
```

Tag 提供以下方法：

```
class Tag {
void SetKey(const std::string key); // 设置 key
void SetValue(const std::string value); // 设置 value
```

查询存储桶标签

功能说明

GET Bucket tagging 用于查询指定存储桶下已有的存储桶标签。

方法原型

```
CosResult CosAPI::GetBucketTagging(const GetBucketTaggingReq& request, GetBucketT
aggingResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::GetBucketTaggingReq req(bucket_name);
qcloud_cos::GetBucketTaggingResp resp;

qcloud_cos::CosResult result = cos.GetBucketTagging(req, &resp);

if (result.IsSucc()) {
// 请求成功，通过 resp 的方法获取 Tagging
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

GetBucketTaggingResp 提供以下方法获取 Tagging

```
std::vector<Tag> GetTagSet() const;
```

参数说明

参数	参数描述	类型	是否必填
req	GetBucketTagging 操作的请求	GetBucketTaggingReq	是
resp	GetBucketTagging 操作的响应	GetBucketTaggingResp	是

删除存储桶标签

功能说明

DELETE Bucket tagging 用于删除指定存储桶下已有的存储桶标签。

方法原型

```
CosResult CosAPI::DeleteBucketTagging(const DeleteBucketTaggingReq& request, Dele
teBucketTaggingResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
```

```
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::DeleteBucketTaggingReq req(bucket_name);
qcloud_cos::DeleteBucketTaggingResp resp;

qcloud_cos::CosResult result = cos.DeleteBucketTagging(req, &resp);

if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	DeleteBucketTagging 操作的请求	DeleteBucketTaggingReq	是
resp	DeleteBucketTagging 操作的响应	DeleteBucketTaggingResp	是

静态网站

最近更新时间：2021-01-05 16:04:01

简介

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket website	设置静态网站	设置存储桶的静态网站配置
GET Bucket website	查询静态网站配置	查询存储桶的静态网站配置
DELETE Bucket website	删除静态网站配置	删除存储桶的静态网站配置

设置静态网站

功能说明

PUT Bucket website 用于为存储桶配置静态网站。

方法原型

```
CosResult CosAPI::PutBucketWebsite(const PutBucketWebsiteReq& request,PutBucketWebsiteResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::PutBucketWebsiteReq req(bucket_name);
qcloud_cos::PutBucketWebsiteResp resp;

req.SetSuffix("index.xml"); //必选项
req.SetProtocol("https");
req.SetKey("Error.html");

//设置重定向规则，最多设置100条

// 设置第一条规则
```

```
qcloud_cos::RoutingRule routerule1;
qcloud_cos::Condition temp_condtion1;
temp_condtion1.SetHttpErrorCodeReturnedEquals(404);//需要设置，默认404
routerule1.SetCondition(temp_condtion1);
qcloud_cos::Redirect temp_redirect1;
temp_redirect1.SetProtocol("https");
temp_redirect1.SetReplaceKeyWith("404.htmp");
routerule1.SetRedirect(temp_redirect1);

// 设置第二条规则
qcloud_cos::RoutingRule routerule2;
qcloud_cos::Condition temp_condtion2;
temp_condtion2.SetHttpErrorCodeReturnedEquals(403);//需要设置，默认404
routerule2.SetCondition(temp_condtion2);
qcloud_cos::Redirect temp_redirect2;
temp_redirect2.SetProtocol("https");
temp_redirect2.SetReplaceKeyWith("403.htmp");
routerule2.SetRedirect(temp_redirect2);

// 设置第三条规则
qcloud_cos::RoutingRule routerule3;
qcloud_cos::Condition temp_condtion3;
temp_condtion3.SetKeyPrefixEquals("img/");
temp_condtion3.SetHttpErrorCodeReturnedEquals(402);
routerule3.SetCondition(temp_condtion3);
qcloud_cos::Redirect temp_redirect3;
temp_redirect3.SetProtocol("https");
temp_redirect3.SetReplaceKeyWith("401.htmp");
routerule3.SetRedirect(temp_redirect3);

// 设置第四条规则
qcloud_cos::RoutingRule routerule4;
qcloud_cos::Condition temp_condtion4;
temp_condtion4.SetKeyPrefixEquals("img1/");
routerule4.SetCondition(temp_condtion4);
qcloud_cos::Redirect temp_redirect4;
temp_redirect4.SetProtocol("https");
temp_redirect4.SetReplaceKeyPrefixWith("402.htmp");
routerule4.SetRedirect(temp_redirect4);

req.AddRoutingRule(routerule1);
req.AddRoutingRule(routerule2);
req.AddRoutingRule(routerule3);
req.AddRoutingRule(routerule4);

qcloud_cos::CosResult result = cos.PutBucketWebsite(req, &resp);
```

版权所有：腾讯云计算（北京）有限责任公司

```
if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutBucketWebsite 操作的请求	PutBucketWebsiteReq	是
resp	PutBucketWebsite 操作的响应	PutBucketWebsiteResp	是

查询静态网站配置

功能说明

GET Bucket website 用于查询与存储桶关联的静态网站配置信息。

方法原型

```
CosResult CosAPI::GetBucketWebsite(const GetBucketWebsiteReq& request, GetBucketWebsiteResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::GetBucketWebsiteReq req(bucket_name);
qcloud_cos::GetBucketWebsiteResp resp;

qcloud_cos::CosResult result = cos.GetBucketWebsite(req, &resp);

if (result.IsSucc()) {
// 请求成功，通过 resp 获取静态网站配置
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetBucketWebsite 操作的请求	GetBucketWebsiteReq	是
resp	GetBucketWebsite 操作的响应	GetBucketWebsiteResp	是

GetBucketWebsiteResp 提供如下方法获取静态网站配置：

```
std::vector<RoutingRule> GetRoutingRules() const;
```

RoutingRule 的定义可以参考 SDK 头文件。

删除静态网站配置

功能说明

DELETE Bucket website 用于删除存储桶中的静态网站配置。

方法原型

```
CosResult CosAPI::DeleteBucketWebsite(const DeleteBucketWebsiteReq& request, Dele
teBucketWebsiteResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::DeleteBucketWebsiteReq req(bucket_name);
qcloud_cos::DeleteBucketWebsiteResp resp;

qcloud_cos::CosResult result = cos.DeleteBucketWebsite(req, &resp);

if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填

req	DeleteBucketWebsite 操作的请求	DeleteBucketWebsiteReq	是
resp	DeleteBucketWebsite 操作的响应	DeleteBucketWebsiteResp	是

版权所有：腾讯云计算（北京）有限责任公司

清单

最近更新时间：2021-01-05 16:04:02

简介

本文档提供关于清单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket inventory	设置清单任务	设置存储桶的清单任务
GET Bucket inventory	查询清单任务	查询存储桶的清单任务
LIST Bucket inventory	列出清单任务	列出存储桶的清单任务
DELETE Bucket inventory	删除清单任务	删除存储桶的清单任务

设置清单任务

功能说明

PUT Bucket inventory 用于在存储桶中创建清单任务。

方法原型

```
CosResult BucketOp::PutBucketInventory(const PutBucketInventoryReq& req, PutBucketInventoryResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::PutBucketInventoryReq req(bucket_name);
qcloud_cos::PutBucketInventoryResp resp;

// 设置清单任务
req.SetId("list");

COSBucketDestination destination;
destination.SetFormat("CSV");
```

```
destination.SetAccountId("100010974959");

destination.SetBucket("qcs::cos:ap-guangzhou::loggtest-1234567890");
destination.SetPrefix("/");
destination.SetEncryption(true);

OptionalFields fields;
fields.SetIsSize(true);
fields.SetIsLastModified(true);
fields.SetIsStorageClass(true);
fields.SetIsMultipartUploaded(true);
fields.SetIsReplicationStatus(true);
fields.SetIsEtag(true);

Inventory inventory;
inventory.SetIsEnable(true);
inventory.SetIncludedObjectVersions("All");
inventory.SetFilter("/");
inventory.SetId(req.GetId());
inventory.SetFrequency("Daily");
inventory.SetCOSBucketDestination(destination);
inventory.SetOptionalFields(fields);

req.SetInventory(inventory);

qcloud_cos::CosResult result = cos.PutBucketInventory(req, &resp);
if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutBucketInventory 操作的请求	PutBucketInventoryReq	是
resp	PutBucketInventory 操作的响应	PutBucketInventoryResp	是

PutBucketInventoryReq 提供以下成员函数：

```
// 设置清单规则
void SetInventory(Inventory& inventory);
```

该请求涉及到的类定义如下：

```
// 清单配置参数
class Inventory {
void SetId(const std::string& id) // 设置清单的名称
void SetIsEnable(bool is_enabled) // 清单是否启用
void SetIncludedObjectVersions(const std::string& included_objectversions) // 是否在清单中包含对象版本：
void SetFilter(const std::string& filter) // 筛选待分析对象（前缀）
void SetCOSBucketDestination(const COSBucketDestination& destination) // 设置清单结果导出后存放的存储桶信息
void SetOptionalFields(const OptionalFields& fields) // 设置清单结果中应包含的分析项目
void SetFrequency(const std::string& frequency) // 设置清单任务的周期，支持 Daily 和 Weekly
}

// 清单结果中应包含的分析项目
class OptionalFields {
void SetIsSize(const bool size); // 设置是否需要文件大小
void SetIsLastModified(const bool last_modified) // 设置是否需要文件修改时间
void SetIsStorageClass(const bool storage_class) // 设置是否需要存储类型
void SetIsMultipartUploaded(const bool ismultipart_uploaded) // 设置是否需要分块上传
void SetIsReplicationStatus(const bool replication_status) // 设置是否需要跨区域复制状态
}

// 清单结果导出后存放的存储桶信息
class COSBucketDestination {
public:
void SetFormat(const std::string& format) // 设置清单生成文件格式，目前支持 CSV 格式
void SetAccountId(const std::string& accountId) // 存储桶的所有者 ID
void SetBucket(const std::string& bucket) // 设置存储桶完整 ID
void SetPrefix(const std::string& prefix) // 设置清单分析结果的存储桶名
void SetEncryption(const bool encryption) // 设置清单结果是否加密
```

错误码说明

该请求可能会发生的一些常见的特殊错误如下：

错误码	描述	状态码
InvalidArgument	不合法的参数值	HTTP 400 Bad Request
TooManyConfigurations	清单数量已经达到1000条的上限	HTTP 400 Bad Request

AccessDenied	未授权的访问。您可能不具备访问该存储桶的权限	HTTP 403 Forbidden

查询清单任务

功能说明

GET Bucket inventory 用于查询存储桶中用户的清单任务信息。

方法原型

```
CosResult CosAPI::GetBucketInventory(const GetBucketInventoryReq& request, GetBucketInventoryResp* response) {
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::GetBucketInventoryReq req(bucket_name);
qcloud_cos::GetBucketInventoryResp resp;

// 设置清单任务
req.SetId("list");

qcloud_cos::CosResult result = cos.GetBucketInventoryResp(req, &resp);
if (result.IsSucc()) {
// 请求成功，获取清单信息
//const Inventory inventory = resp.GetInventory();
//std::cout << "inventory isenabled:" << inventory.GetIsEnable() << std::endl;
//std::cout << "inventory IncludedObjectVersions:" << inventory.GetIncludedObjectVersions() << std::endl;
//std::cout << "inventory filter:" << inventory.GetFilter() << std::endl;
//std::cout << "inventory id:" << inventory.GetId() << std::endl;
//std::cout << "inventory Frequency:" << inventory.GetFrequency() << std::endl;

//std:: cout << "==================================" << std::endl;
//COSBucketDestination destination = inventory.GetCOSBucketDestination();
//std::cout << "destination Format:" << destination.GetFormat() << std::endl;
//std::cout << "destination AccountID:" << destination.GetAccountId() << std::endl;
//std::cout << "destination Bucket:" << destination.GetBucket() << std::endl;
//std::cout << "destination Encryption:" << destination.GetEncryption() << std::endl;
```

```
//std:: cout << "==================================" << std::endl;
//
//OptionalFields fields = inventory.GetOptionalFields();

//std::cout << "fields Size:" << fields.GetIsSize() << std::endl;
//std::cout << "fields LastModified:" << fields.GetIsLastModified() << std::endl;
//std::cout << "fields StorageClass:" << fields.GetIsStorageClass() << std::endl;
//std::cout << "fields Region:" << fields.GetIsMultipartUploaded() << std::endl;
//std::cout << "fields ReplicationStatus:" << fields.GetIsReplicationStatus() <<
std::endl;
//std::cout << "fields Tag:" << fields.GetIsETag() << std::endl;
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetBucketInventory 操作的请求	GetBucketInventoryReq	是
resp	GetBucketInventory 操作的响应	GetBucketInventoryResp	是

GetBucketInventoryResp 提供以下成员函数：

```
// 获取清单规则
const Inventory& GetInventory() const;
```

查询所有清单

功能说明

List Bucket Inventory Configurations 用于请求返回一个存储桶中的所有清单任务。

方法原型

```
CosResult CosAPI::ListBucketInventoryConfigurations(const ListBucketInventoryConf
igurationsReq& request, ListBucketInventoryConfigurationsResp* response) {
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::ListBucketInventoryConfigurationsReq req(bucket_name);
qcloud_cos::ListBucketInventoryConfigurationsResp resp;

qcloud_cos::CosResult result = cos.ListBucketInventoryConfigurations(req, &resp);
if (result.IsSucc()) {
// 请求成功，获取清单信息
// std::cout << "==================ListBucketInventoryConfigurations============
=========" << std::endl;
//
// std::vector<Inventory> inventory_vec = resp.GetInventory();
//
// std::cout << resp.GetIsTruncated() << std::endl;
// std::cout << resp.GetContinuationToken() << std::endl;
// std::cout << resp.GetNextContinuationToken() << std::endl;
//
// std::vector<Inventory>::iterator itr = inventory_vec.begin();
// for(; itr != inventory_vec.end(); ++itr) {
//
// std:: cout << "==============Inventory==============================" << std::e
ndl;
// std::cout << "inventory id:" << itr->GetId() << std::endl;
// std::cout << "inventory isenabled:" << itr->GetIsEnable() << std::endl;
// std::cout << "inventory IncludedObjectVersions:" << itr->GetIncludedObjectVers
ions() << std::endl;
// std::cout << "inventory filter:" << itr->GetFilter() << std::endl;
// std::cout << "inventory Frequency:" << itr->GetFrequency() << std::endl;
// std:: cout << "==============GetCOSBucketDestination=================" << st
d::endl;
// COSBucketDestination destination = itr->GetCOSBucketDestination();
// std::cout << "destination Format:" << destination.GetFormat() << std::endl;
// std::cout << "destination AccountID:" << destination.GetAccountId() << std::en
dl;
// std::cout << "destination Bucket:" << destination.GetBucket() << std::endl;
// std::cout << "destination Encryption:" << destination.GetEncryption() << std::
endl;
//
// std:: cout << "==================OptionalFields======================" << st
d::endl;
// OptionalFields fields = itr->GetOptionalFields();
// std::cout << "fields Size:" << fields.GetIsSize() << std::endl;
// std::cout << "fields LastModified:" << fields.GetIsLastModified() << std::end
l;
// std::cout << "fields StorageClass:" << fields.GetIsStorageClass() << std::end
```

```
l;
// std::cout << "fields Region:" << fields.GetIsMultipartUploaded() << std::endl;
// std::cout << "fields ReplicationStatus:" << fields.GetIsReplicationStatus() <<
std::endl;
// std::cout << "fields Tag:" << fields.GetIsETag() << std::endl;
// }
// std::cout << "===============ListBucketInventoryConfigurations end============
====" << std::endl;
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	ListBucketInventoryConfigurations 操作的请求	ListBucketInventoryConfigurationsReq	是
resp	ListBucketInventoryConfigurations 操作的响应	ListBucketInventoryConfigurationsResp	是

ListBucketInventoryConfigurationsReq 提供以下成员函数：

```
// 当 COS 响应体中 IsTruncated 为 true，且 NextContinuationToken 节点中存在参数值时，您
可以将这个参数作为 continuation-token 参数值，以获取下一页的清单任务信息。
void SetContinuationToken(const std::string continuation_token);
```

删除清单任务

功能说明

DELETE Bucket inventory 用于删除存储桶中指定的清单任务。

方法原型

```
CosResult CosAPI::DeleteBucketInventory(const DeleteBucketInventoryReq& request,
DeleteBucketInventoryResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::DeleteBucketInventoryReq req(bucket_name);
qcloud_cos::DeleteBucketInventoryResp resp;

qcloud_cos::CosResult result = cos.DeleteBucketInventory(req, &resp);
if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
```

参数说明

参数	参数描述	类型	是否必填
req	DeletBucketInventory 操作的请求	DeletBucketInventoryReq	是
resp	DeletBucketInventory 操作的响应	DeletBucketInventoryResp	是

DeleteBucketInventoryReq 提供以下成员函数：

```
void SetId(const std::string id) // 设置要删除清单的 ID
```

访问管理
跨域访问

最近更新时间：2021-01-05 16:04:02

简介

本文档提供关于跨域访问的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域名访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域名访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域名访问配置信息

设置跨域配置

功能说明

设置指定存储桶的跨域名访问配置信息（PUT Bucket cors）。

方法原型

```
CosResult PutBucketCORS(const PutBucketCORSReq& request, PutBucketCORSResp* respo
nse);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::PutBucketCORSReq req(bucket_name);
qcloud_cos::PutBucketCORSResp resp;

// 设置跨域请求内容
qcloud_cos::CORSRule rule;
rule.m_id = "123"; // 设置跨域配置 ID
rule.m_allowed_headers.push_back("x-cos-meta-test"); // 设置跨域允许的 HTTP 请求头部
rule.m_allowed_origins.push_back("http://www.qq.com"); // 设置跨域允许的来源域名
```

版权所有：腾讯云计算（北京）有限责任公司

```
rule.m_allowed_methods.push_back("PUT"); // 设置跨域允许的 HTTP 方法
rule.m_allowed_methods.push_back("GET");
rule.m_max_age_secs = "600"; // 设置有效配置的时间
rule.m_expose_headers.push_back("x-cos-expose"); // 允许浏览器获取的 CORS 请求响应中的头部
req.AddRule(rule);

qcloud_cos::CosResult result = cos.PutBucketCORS(req, &resp);
if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutBucketCORS 操作的请求	PutBucketCORSReq	是
resp	PutBucketCORS 操作的响应	PutBucketCORSResp	是

PutBucketCORSReq 提供以下成员函数：

```
// 添加跨域规则
void AddRule(const CORSRule& rule);
// 添加跨域规则
void SetRules(const std::vector<CORSRule>& rules)
```

该请求涉及到的类定义如下：

```
struct CORSRule {
std::string m_id;
std::string m_max_age_secs;
std::vector<std::string> m_allowed_headers;
std::vector<std::string> m_allowed_methods;
std::vector<std::string> m_allowed_origins;
std::vector<std::string> m_expose_headers;
};
```

查询跨域配置

功能说明

查询指定存储桶的跨域名访问配置信息（GET Bucket cors）。

方法原型

```
CosResult CosAPI::GetBucketCORS(const GetBucketCORSReq& request, GetBucketCORSRes
p* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::GetBucketCORSReq req(bucket_name);
qcloud_cos::GetBucketCORSResp resp;

qcloud_cos::CosResult result = cos.GetBucketCORS(req, &resp);
if (result.IsSucc()) {
// 请求成功，调用 resp 方法获取跨域规则
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetBucketCORS 操作的请求	GetBucketCORSReq	是
resp	GetBucketCORS 操作的响应	GetBucketCORSResp	是

GetBucketCORSReq 提供以下成员函数：

```
// 从响应中获取跨域规则
std::vector<CORSRule> GetCORSRules() const;
```

删除跨域配置

功能说明

删除指定存储桶的跨域名访问配置（DELETE Bucket cors）。

方法原型

```
CosResult BucketOp::DeleteBucketCORS(const DeleteBucketCORSReq& req, DeleteBucket
CORSResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::DeleteBucketCORSReq req(bucket_name);
qcloud_cos::DeleteBucketCORSResp resp;

qcloud_cos::CosResult result = cos.DeleteBucketCORS(req, &resp);
if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	DeleteBucketCORS 操作的请求	DeleteBucketCORSReq	是
resp	DeleteBucketCORS 操作的响应	DeleteBucketCORSResp	是

存储桶策略

最近更新时间：2023-04-25 15:29:35

简介

本文档提供关于存储桶策略的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket policy	设置存储桶策略	设置指定存储桶的权限策略
GET Bucket policy	查询存储桶策略	查询指定存储桶的权限策略
DELETE Bucket policy	删除存储桶策略	删除指定存储桶的权限策略

设置存储桶策略

功能说明

PUT Bucket policy 请求可以向 Bucket 写入权限策略，当存储桶已存在权限策略时，该请求上传的策略将覆盖原有的权限策略。

方法原型

```
CosResult PutBucketPolicy(const PutBucketPolicyReq& req, PutBucketPolicyResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket

qcloud_cos::PutBucketPolicyReq req(bucket_name);
qcloud_cos::PutBucketPolicyResp resp;
std::string bucket_policy =
" {"
```

```
" \"Statement\": ["
" {"
" \"Principal\": {"
" \"qcs\": ["
" \"qcs::cam::uin/100000000001:uin/100000000011\"" //替换成您想授予权限的账户 uin
" ]\n"
" },\n"
" \"Effect\": \"allow\","
" \"Action\": ["
" \"cos:PutObject\""
" ],\n"
" \"Resource\": [" //这里改成允许的路径前缀，可以根据自己网站的用户登录态判断允许上传的具体路径，例子： a.jpg 或者 a/* 或者 * (使用通配符*存在重大安全风险，请谨慎评估使用)
" \"qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/exampleobject\""
" ],\n"
" \"Condition\": {"
" \"string_equal\": {"
" \"cos:x-cos-mime-limit\": \"image/jpeg\""
" }"
" }"
" }"
" ],"
" \"Version\": \"2.0\""
" }";

req.SetBody(bucket_policy);
qcloud_cos::CosResult result = cos.PutBucketPolicy(req, &resp);

if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数名称	描述	必填
Statement	描述一条或多条权限的详细信息	是
Version	策略语法版本，默认为2.0	是
Principal	描述策略授权的实体，详情请参见 访问策略语言概述	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	必填
Action	此处是指 COS API，根据需求指定一个或者一序列操作的组合或所有操作(*)，例如 action 为 name/cos:GetService，**请注意区分英文大小写**	是
Effect	有 allow（允许）和 deny（显式拒绝）两种情况	是
Resource	授权操作的具体数据，可以是任意资源、指定路径前缀的资源、指定绝对路径的资源或它们的组合	是
Condition	约束条件，可以不填，具体说明请参见 condition 说明	否

查询存储桶策略

功能说明

GET Bucket policy 请求可以向 Bucket 读取权限策略。

方法原型

```
CosResult GetBucketPolicy(const GetBucketPolicyReq& req, GetBucketPolicyResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket

qcloud_cos::GetBucketPolicyReq req(bucket_name);
qcloud_cos::GetBucketPolicyResp resp;
qcloud_cos::CosResult result = cos.GetBucketPolicy(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

版权所有：腾讯云计算（北京）有限责任公司

返回结果说明

GetBucketPolicyResp 提供以下成员函数，用于获取 Get Bucket Policy 返回的 Policy 内容。

```
std::string resp.GetPolicy()
```

删除存储桶策略

功能说明

DELETE Bucket policy 请求可以向 Bucket 删除权限策略。

方法原型

```
CosResult DeleteBucketPolicy(const DeleteBucketPolicyReq& req, DeleteBucketPolicy
Resp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由 buc
ketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称：https://console.int
l.cloud.tencent.com/cos5/bucket

qcloud_cos::DeleteBucketPolicyReq req(bucket_name);
qcloud_cos::DeleteBucketPolicyResp resp;
qcloud_cos::CosResult result = cos.DeleteBucketPolicy(req, &resp);

if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

访问控制

最近更新时间：2021-10-08 15:04:50

简介

本文档提供关于存储桶、对象的访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

存储桶 ACL

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶的访问权限控制列表（ACL）
GET Bucket acl	查询存储桶 ACL	查询指定存储桶的访问权限控制列表（ACL）

对象 ACL

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表
GET Object acl	查询对象 ACL	查询对象的访问控制列表

存储桶 ACL

设置存储桶 ACL

功能说明

设置指定存储桶的访问权限控制列表（PUT Bucket acl）。该操作是覆盖操作，会覆盖已有的权限设置。ACL 包括预定义权限策略（CannedAccessControlList）或者自定义的权限控制（AccessControlList）。两类权限当同时设置时将忽略预定义策略，以自定义策略为主。

方法原型

```
CosResult PutBucketACL(const PutBucketACLReq& request, PutBucketACLResp* respons
e);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
// PutBucketACLReq 的构造函数需要传入 bucket_name
qcloud_cos::PutBucketACLReq req(bucket_name);
qcloud_cos::PutBucketACLResp resp;
// 设置 ACL 配置(可以通过 Body、Header 两种方式，但只能二选一，否则会有冲突)
// 1.通过 header 设置 ACL
req.SetXCosAcl("public-read-write");
// 2.通过 body 设置 ACL
qcloud_cos::Owner owner = {"qcs::cam::uin/00000000001:uin/00000000001", "qcs::ca
m::uin/00000000001:uin/00000000001" };
qcloud_cos::Grant grant;
req.SetOwner(owner);
grant.m_grantee.m_type = "Group";
grant.m_grantee.m_uri = "http://cam.qcloud.com/groups/global/AllUsers";
grant.m_perm = "READ";
req.AddAccessControlList(grant);
qcloud_cos::PutBucketACLResp resp;
qcloud_cos::CosResult result = cos.PutBucketACL(req, &resp);
if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutBucketACL 操作的请求	PutBucketACLReq	是
resp	PutBucketACL 操作的响应	PutBucketACLResp	是

PutBucketACLReq 提供以下成员函数：

```
// 定义 Bucket 的 ACL 属性,有效值：private,public-read-write,public-read
// 默认值：private
void SetXCosAcl(const std::string& str);
// 赋予被授权者读的权限.格式：x-cos-grant-read: id=" ",id=" ".
// 当需要给子账号授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"
// 当需要给主账号授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXCosGrantRead(const std::string& str);
// 赋予被授权者写的权限,格式：x-cos-grant-write: id=" ",id=" "./
// 当需要给子账号授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>",
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 当需要给主账号授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXCosGrantWrite(const std::string& str);
// 赋予被授权者读写权限.格式：x-cos-grant-full-control: id=" ",id=" ".
// 当需要给子账号授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>",
// 当需要给主账号授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXCosGrantFullControl(const std::string& str);
// Bucket 持有者 ID
void SetOwner(const Owner& owner);
// 设置被授权者信息与权限信息
void SetAccessControlList(const std::vector<Grant>& grants);
// 添加单个 Bucket 的授权信息
void AddAccessControlList(const Grant& grant);
```

注意：

SetXCosAcl、SetXCosGrantRead、SetXCosGrantWrite、SetXCosGrantFullControl 这类接口与 SetAccessControlList、AddAccessControlList 不可同时使用。因为前者实际是通过设置 HTTP Header 实现，而后者是在 Body 中添加了 XML 格式的内容，二者只能二选一。 SDK 内部优先使用第一类。

该请求涉及到的类定义如下：

```
struct Owner {
// 存储桶持有者信息
std::string m_id; // owner 的完整 ID，格式为：qcs::cam::uin/<OwnerUin>:uin/<SubUin>
std::string m_display_name; // owner 的描述
};
struct Grantee {
// type 类型可以为 RootAccount， SubAccount
// 当 type 类型为 RootAccount 时，可以在 id 中 uin 填写账号 ID，也可以用 anyone（指代所有类型用户）代替 uin/<OwnerUin> 和 uin/<SubUin>
// 当 type 类型为 RootAccount 时，uin 代表主账号，Subaccount 代表子账号
std::string m_type;
std::string m_id; // qcs::cam::uin/<OwnerUin>:uin/<SubUin>
std::string m_display_name; // 非必选
std::string m_uri;
};
struct Grant {
Grantee m_grantee; // 被授权者资源信息
std::string m_perm; // 指明授予被授权者的权限信息，可选值"READ", "WRITE", "FULL_CONTROL"
};
```

查询存储桶 ACL

功能说明

查询指定存储桶的访问权限控制列表。

方法原型

```
CosResult GetBucketACL(const GetBucketACLReq& req, GetBucketACLResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::GetBucketACLReq req(bucket_name);
qcloud_cos::GetBucketACLResp resp;
qcloud_cos::CosResult result = cos.GetBucketACL(req, &resp);
if (result.IsSucc()) {
// 请求成功，调用 resp 的成员函数获取返回内容
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetBucketACL 操作的请求	GetBucketACLReq	是
resp	GetBucketACL 操作的响应	GetBucketACLResp	是

GetBucketACLResp 提供以下成员函数：

```
std::string GetOwnerID();
std::string GetOwnerDisplayName();
std::vector<Grant> GetAccessControlList();
```

对象 ACL

设置对象 ACL

功能说明

设置存储桶中某个对象的访问控制列表。

注意：
当前访问策略条目限制为1000条，如果您不需要进行对象 ACL 控制，请不要设置，默认继承 Bucket 权限。

ACL 包括预定义权限策略（CannedAccessControlList）或者自定义的权限控制（AccessControlList）。两类权限当同时设置时将忽略预定义策略，以自定义策略为主。

方法原型

```
CosResult PutObjectACL(const PutObjectACLReq& req, PutObjectACLResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "test";
// 1 设置 ACL 配置（通过 Body，设置 ACL 可以通过 Body、Header 两种方式，但只能二选一，否则会有冲突）
{
qcloud_cos::PutObjectACLReq req(bucket_name, object_name);
qcloud_cos::Owner owner = {"qcs::cam::uin/xxxxx:uin/xxx", "qcs::cam::uin/xxxxxx:uin/xxxxx" };
qcloud_cos::Grant grant;
req.SetOwner(owner);
grant.m_grantee.m_type = "Group";
grant.m_grantee.m_uri = "http://cam.qcloud.com/groups/global/AllUsers";
grant.m_perm = "READ";
req.AddAccessControlList(grant);
qcloud_cos::PutObjectACLResp resp;
qcloud_cos::CosResult result = cos.PutObjectACL(req, &resp);
if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
}
// 2 设置 ACL 配置（通过 Header，设置 ACL 可以通过 Body、Header 两种方式，但只能二选一，否则会有冲突）
{
qcloud_cos::PutObjectACLReq req(bucket_name, object_name);
req.SetXCosAcl("public-read-write");
qcloud_cos::PutObjectACLResp resp;
qcloud_cos::CosResult result = cos.PutObjectACL(req, &resp);
```

```
if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutObjectACL 操作的请求	PuttObjectACLReq	是
resp	PutObjectACL 操作的响应	PutObjectACLResp	是

PutObjectACLReq 包含以下成员函数：

```
// 定义 Object 的 ACL 属性,有效值：private,public-read
// 默认值：private
void SetXCosAcl(const std::string& str);
// 赋予被授权者读的权限。格式：id="[OwnerUin]"
void SetXCosGrantRead(const std::string& str);
// 赋予被授权者所有的权限。格式：id="[OwnerUin]"
void SetXCosGrantFullControl(const std::string& str);
// Object 持有者 ID
void SetOwner(const Owner& owner);
// 设置被授权者信息与权限信息
void SetAccessControlList(const std::vector<Grant>& grants);
// 添加单个 Object 的授权信息
void AddAccessControlList(const Grant& grant);
```

注意：
SetXCosAcl/SetXCosGrantRead/SetXCosGrantWrite/SetXCosGrantFullControl 这类接口与 SetAccessControlList/AddAccessControlList 不可同时使用。因为前者实际是通过设置 HTTP Header 实现，而后者是在Body 中添加了 XML 格式的内容，二者只能二选一。SDK 内部优先使用第一类。

ACLRule 定义如下：

```
struct Grantee {
// type 类型可以为 RootAccount,  SubAccount
// 当 type 类型为 RootAccount 时，可以在 id 中 uin 填写账号 ID，也可以用 anyone（指代所有
```

版权所有：腾讯云计算（北京）有限责任公司

```
类型用户）代替 uin/<OwnerUin> 和 uin/<SubUin>
// 当 type 类型为 RootAccount 时，uin 代表主账号账号，Subaccount 代表子账号账号
std::string m_type;
std::string m_id; // qcs::cam::uin/<OwnerUin>:uin/<SubUin>
std::string m_display_name; // 非必选
std::string m_uri;
};
struct Grant {
Grantee m_grantee; // 被授权者资源信息
std::string m_perm; // 指明授予被授权者的权限信息，可选值"READ", "WRITE", "FULL_CONTRO
L"
};
```

查询对象 ACL

功能说明

查询对象的访问控制列表。

方法原型

```
CosResult GetObjectACL(const GetObjectACLReq& req, GetObjectACLResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "exampleobject";
// GetObjectACLReq 的构造函数需要传入 Object_name
qcloud_cos::GetObjectACLReq req(bucket_name, object_name);
qcloud_cos::GetObjectACLResp resp;
qcloud_cos::CosResult result = cos.GetObjectACL(req, &resp);
if (result.IsSucc()) {
// 请求成功，调用 resp 的成员函数获取返回内容
} else {
// 请求失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetObjectACL 操作的请求	GetObjectACLReq	是

参数	参数描述	类型	是否必填
resp	GetObjectACL 操作的响应	GetObjectACLResp	是

GetObjectACLResp 包含以下成员函数：

```
std::string GetOwnerID();
std::string GetOwnerDisplayName();
std::vector<Grant> GetAccessControlList();
```

防盗链

最近更新时间：2022-01-04 12:30:50

简介

本文档提供关于存储桶 Referer 白名单或者黑名单的 API 概览以及 SDK 示例代码。

> 注意：
> 需要v5.5.0 及以上版本。

API	操作名	操作描述
PUT Bucket referer	设置存储桶 Referer	设置存储桶 Referer 白名单或者黑名单
GET Bucket referer	查询存储桶 Referer	查询存储桶 Referer 白名单或者黑名单

设置存储桶 Referer

功能说明

设置指定存储桶的 Referer 白名单或者黑名单（PUT Bucket referer）。

方法原型

```
CosResult PutBucketReferer(const PutBucketRefererReq& request, PutBucketRefererResp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名
qcloud_cos::PutBucketRefererReq req(bucket_name);
qcloud_cos::PutBucketRefererResp resp;
// 根据实际情况构造请求
req.SetStatus("Enabled");
req.SetRefererType("White-List");
req.AddDomain("test1.com");
```

```
req.AddDomain("test2.com");
qcloud_cos::CosResult result = cos.PutBucketReferer(req, &resp);
if (result.IsSucc()) {
// 请求成功
} else {
// 请求失败，调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	PutBucketReferer操作的请求	PutBucketRefererReq	是
resp	PutBucketReferer操作的响应	PutBucketRefererResp	是

返回结果说明

无。

查询存储桶 Referer

功能说明

查询指定存储桶 Referer 白名单或者黑名单（GET Bucket referer）。

方法原型

```
CosResult GetBucketReferer(const GetBucketRefererReq& request, GetBucketRefererRe
sp* response);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名
qcloud_cos::GetBucketRefererReq req(bucket_name);
qcloud_cos::GetBucketRefererResp resp;
qcloud_cos::CosResult result = cos.GetBucketReferer(req, &resp);
if (result.IsSucc()) {
// 请求成功,获取referer配置
std::cout << "Status:" << resp.GetStatus() << std::endl;
std::cout << "RefererType:" << resp.GetRefererType() << std::endl;
for (auto& domain : resp.GetDomainList()) {
```

```
std::cout << "Domain:" << domain << std::endl;
}
std::cout << "EmptyReferConfiguration:" << resp.GetEmptyReferConf() << std::endl;
} else {
// 请求失败，调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetBucketReferer操作的请求	GetBucketRefererReq	是
resp	GetBucketReferer操作的响应	GetBucketRefererResp	是

返回结果说明

无。

预签名 URL

最近更新时间：2023-06-16 16:02:05

简介

C++ SDK 提供生成签名和获取请求预签名 URL 接口，详细操作请查看本文说明和示例。

关于使用预签名 URL 上传的说明请参见 预签名授权上传， 使用预签名 URL 下载的说明请参见 预签名授权下载。

说明：

- 建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄露目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。

生成签名

功能说明

计算并生成签名。

方法原型一

```
static std::string Sign(const std::string& secret_id,
const std::string& secret_key,
const std::string& http_method,
const std::string& in_uri,
const std::map<std::string, std::string>& headers,
const std::map<std::string, std::string>& params);
```

参数说明

参数名称	参数描述	类型
secret_id	开发者拥有的项目身份识别 ID，用以身份认证	String
secret_key	开发者拥有的项目身份密钥	String

参数名称	参数描述	类型
http_method	HTTP 方法，如 POST/GET/HEAD/PUT 等， 传入大小写不敏感	String
in_uri	HTTP uri	String
headers	HTTP header 的键值对	map<string,string>
params	HTTP params 的键值对	map<string,string>

返回结果说明

返回签名字符串，可以在指定的有效期内（通过 CosSysConfig 设置，默认60s）使用，返回空串表示计算签名失败。

方法原型二

```
static std::string Sign(const std::string& secret_id,
const std::string& secret_key,
const std::string& http_method,
const std::string& in_uri,
const std::map<std::string, std::string>& headers,
const std::map<std::string, std::string>& params,
uint64_t start_time_in_s,
uint64_t end_time_in_s);
```

参数说明

参数名称	参数描述	类型
secret_id	开发者拥有的项目身份识别 ID，用以身份认证	String
secret_key	开发者拥有的项目身份密钥	String
http_method	HTTP 方法，如 POST/GET/HEAD/PUT 等，传入大小写不敏感	String
in_uri	HTTP uri	String
headers	HTTP header 的键值对	map <string,string>
params	HTTP params 的键值对	map <string,string>
start_time_in_s	签名生效的开始时间	uint64_t
end_time_in_s	签名生效的截止时间	uint64_t

版权所有：腾讯云计算（北京）有限责任公司

返回结果说明

返回签名字符串，可以在指定的有效期内（通过 CosSysConfig 设置，默认60s）使用，返回空串表示计算签名失败。

获取请求预签名 URL

```
std::string GeneratePresignedUrl(const GeneratePresignedUrlReq& req)
```

参数说明

参数	参数描述
req	GeneratePresignedUrlReq，GeneratePresignedUrl 操作的请求

HTTP_METHOD 枚举定义如下：

```
typedef enum {
HTTP_HEAD,
HTTP_GET,
HTTP_PUT,
HTTP_POST,
HTTP_DELETE,
HTTP_OPTIONS
} HTTP_METHOD;
```

预签名请求示例

可根据 CosConfig 类设置永久密钥或临时密钥发起预签名请求，具体配置文件内容请参见 快速入门 文档。

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "exampleobject";

// 添加存储桶名称和对象键，以及 HTTP 请求方法。
// 注意：用户无需对 object_name 进行编码操作
qcloud_cos::GeneratePresignedUrlReq req(bucket_name, object_name, qcloud_cos::HTT
P_GET);
```

版权所有：腾讯云计算（北京）有限责任公司

```
std::string presigned_url = cos.GeneratePresignedUrl(req);
```

对象访问 URL

最近更新时间：2022-05-25 15:47:18

简介

本文档提供获取对象访问 URL 的代码示例。

获取对象访问 URL

方法原型

```
std::string GetObjectUrl(const std::string& bucket, const std::string& object, bool https = true, const std::string& region);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000"; // 替换为用户的存储桶名，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称：https://console.intl.cloud.tencent.com/cos5/bucket
std::string object_name = "exampleobject";
//获取对象https url，region为config.json中配置的地域
cos.GetObjectUrl(bucket_name, object_name);
//获取对象http url，region为config.json中配置的地域
cos.GetObjectUrl(bucket_name, object_name, false);
//获取对象https url，region为ap-shanghai
std::string object_url = cos.GetObjectUrl(bucket_name, object_name, true, "ap-shanghai");
```

参数说明

参数	参数描述	类型	是否必填
bucket	存储桶名	string	是
object	对象名	string	是
https	是否使用 https	bool	否

参数	参数描述	类型	是否必填
region	地域名	string	否

返回结果说明

该方法返回值为对象访问的 URL。

异常处理

最近更新时间：2023-05-24 14:28:12

简介

API 返回 CosResult 结构，如果成功可以获取对应 Response 结构中的数据，失败可以通过 CosResult 获取详细信息。

服务端异常

CosResult 封装了请求出错时返回的错误码和对应错误信息，详情请参见 错误码。

> 注意：
> SDK 内部封装的请求均会返回 CosResult 对象，每次调用完成后，均要使用 IsSucc() 成员函数判断本次调用是否成功。

成员函数

函数	函数描述
bool isSucc()	返回本次调用成功或失败 当返回 false 时：后续的 CosResult 成员函数才有意义 当返回 True 时：可以从 OperatorResp 中获取具体返回内容
string GetErrorCode()	获取 COS 返回的错误码，用来确定错误场景
string GetErrorMsg()	包含具体的错误信息
string GetResourceAddr()	资源地址，Bucket 地址或 Object 地址
string GetXCosRequestId()	当请求发送时，服务端将会自动为请求生成一个唯一的 ID。使用遇到问题时，request-id 能更快地协助 COS 定位问题
string GetXCosTraceId()	当请求出错时，服务端将会自动为这个错误生成一个唯一的 ID，trace-id 与 request-id 一一对应。使用遇到问题时，trace-id 能更快地协助 COS 定位问题
int GetHttpStatus()	获取 HTTP 状态码

请求和响应基类

BaseReq、BaseResp 封装了请求和返回， 调用者只需要根据不同的操作类型生成不同的 OperatorReq，并填充 OperatorReq 的内容即可。
函数返回后，调用对应 BaseResp 的成员函数获取请求结果。

- 对于 Request，如无特殊说明，仅需要关注 Request 的构造函数。
- 对于 Response，所有方法的 Response 均有获取公共返回头部的成员函数。Response 的公共成员函数如下，具体字段含义请参见 公共返回头部，此处不再赘述。

```
uint64_t GetContentLength();
std::string GetContentType();
std::string GetEtag();
std::string GetConnection();
std::string GetDate();
std::string GetServer();
std::string GetXCosRequestId(); //获取 RequestId，即请求 ID
std::string GetXCosTraceId();
```

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。
   COS 自助诊断工具
   点击自助诊断
   输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**，请您耐心等待几分钟，便能看到相应的智能诊断结果。

数据校验
CRC64 校验

最近更新时间：2023-11-06 16:11:52

简介

数据在客户端和服务器间传输时可能会出现错误，对象存储（Cloud Object Storage，COS）除了可以通过 MD5 和自定义属性 验证数据完整性外，还可以通过 CRC64 检验码来进行数据校验。

COS 会对新上传的对象进行 CRC64 计算，并将结果作为对象的属性进行存储，随后在返回的响应头部中携带 x-cos-hash-crc64ecma，该头部表示上传对象的 CRC64 值，根据 ECMA-182标准 计算得到。对于 CRC64 特性上线前就已经存在于 COS 的对象，COS 不会对其计算 CRC64 值，所以获取此类对象时不会返回其 CRC64 值。

操作说明

目前支持 CRC64 的 API 如下：

- 简单上传接口
  PUT Object 和 POST Object：用户可在返回的响应头中获得文件 CRC64 校验值。
- 分块上传接口
  - Upload Part：用户可以根据 COS 返回的 CRC64 值与本地计算的数值进行比较验证。
  - Complete Multipart Upload：如果每个分块都有 CRC64 属性，则会返回整个对象的 CRC64 值，如果某些分块不具备 CRC64 值，则不返回。
- 执行 Upload Part - Copy 时，会返回对应的 CRC64 值。
- 执行 PUT Object - Copy 时，如果源对象存在 CRC64 值，则返回 CRC64，否则不返回。
- 执行 HEAD Object 和 GET Object 时，如果对象存在 CRC64，则返回。用户可以根据 COS 返回的 CRC64 值和本地计算的 CRC64 进行比较验证。

API 接口示例

分块上传响应

下面为用户发出 Upload Part 请求后得到的响应示例。x-cos-hash-crc64ecma 头部表示分块的 CRC64 值，用户可以通过该值与本地计算的 CRC64 值进行比较，从而校验分块完整性。

版权所有：腾讯云计算（北京）有限责任公司

```
HTTP/1.1 200 OK
content-length: 0
connection: close
date: Thu, 05 Dec 2019 01:58:03 GMT
etag: "358e8c8b1bfa35ee3bd44cb3d2cc416b"
server: tencent-cos
x-cos-hash-crc64ecma: 15060521397700495958
x-cos-request-id: NWRlODY0MmJfMjBiNDU4NjRfNjkyZl80ZjZi****
```

完成分块上传响应

下面为用户发出 Complete Multipart Upload 请求后得到的响应示例。x-cos-hash-crc64ecma 头部表示整个对象的 CRC64 值，用户可以通过该值与本地计算的 CRC64 值进行比较，从而校验对象完整性。

```
HTTP/1.1 200 OK
content-type: application/xml
transfer-encoding: chunked
connection: close
date: Thu, 05 Dec 2019 02:01:17 GMT
server: tencent-cos
x-cos-hash-crc64ecma: 15060521397700495958
x-cos-request-id: NWRlODY0ZWRfMjNiMjU4NjRfOGQ4Ml81MDEw****

[Object Content]
```

SDK 校验方式

目前C++ SDK对于不同接口默认校验方式不同：

- 简单上传接口
  PUT Object ：默认使用 MD5 校验，暂不支持 CRC64 校验。
- 分块上传接口
  - Upload Part：默认使用 MD5 校验，暂不支持 CRC64 校验。
  - Complete Multipart Upload：默认使用 CRC64 校验，暂不支持 MD5 校验。

请求示例1：分块上传

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000"; // 修改为用户的存储桶名
std::string object_name = "exampleobject"; // 修改为用户的对象名
```

```
std::string local_file = "./test"; // 修改为用户的本地文件名

qcloud_cos::MultiUploadObjectReq req(bucket_name, object_name, local_file); // 默认开启了CRC64校验
req.SetRecvTimeoutInms(1000 * 60);
qcloud_cos::MultiUploadObjectResp resp;
qcloud_cos::CosResult result = cos.MultiUploadObject(req, &resp); // 内部自动校验CRC64
// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
}
```

设置访问域名（CDN/全球加速）

最近更新时间：2022-05-25 15:45:09

简介

本文档提供关于如何使用非默认域名请求对象存储（Cloud Object Storage，COS）服务。

默认 CDN 加速域名

功能说明

如果需要通过 CDN 加速域名访问 COS，需要在 COS 控制台为存储桶开启 CDN 加速域名。关于如何开启 CDN 加速域名，操作指引请参见 开启默认 CDN 加速域名。

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
// 将 examplebucket-1250000000 修改为真实的存储桶名
CosSysConfig::SetDomainSameToHost(true);
CosSysConfig::SetDestDomain("examplebucket-1250000000.file.myqcloud.com");
```

或者修改 config.json：

```
"IsDomainSameToHost":true,
"DestDomain":"examplebucket-1250000000.file.myqcloud.com",
```

自定义 CDN 加速域名

功能说明

如果需要通过自定义 CDN 加速域名访问 COS，需要在 COS 控制台为存储桶开启自定义 CDN 加速域名。关于如何开启自定义 CDN 加速域名，操作指引请参见 开启自定义 CDN 加速域名。

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
// 将 mycdndomain.com 修改为真实的 CDN 域名
CosSysConfig::SetDomainSameToHost(true);
CosSysConfig::SetDestDomain("mycdndomain.com");
```

或者修改 config.json：

版权所有：腾讯云计算（北京）有限责任公司

```
"IsDomainSameToHost":true,
"DestDomain":"emycdndomain.com",
```

自定义源站域名

功能说明

如果需要通过自定义域名访问 COS，需要在 COS 控制台为存储桶设置自定义源站域名。关于如何设置存储桶自定义域名，操作指引请参见 开启自定义源站域名。

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
// 将 mydomain.com 修改为真实的自定义域名
CosSysConfig::SetDomainSameToHost(true);
CosSysConfig::SetDestDomain("mydomain.com");
```

或者修改config.json：

```
"IsDomainSameToHost":true,
"DestDomain":"mydomain.com",
```

全球加速域名

功能说明

如果需要通过全球加速域名访问 COS，需要在 COS 控制台为存储桶开通全球加速。关于如何开通存储桶全球加速功能，操作指引请参见 开启全球加速。

请求示例

```
qcloud_cos::CosConfig config("./config.json");
config.SetRegion("accelerate")
qcloud_cos::CosAPI cos(config);
```

或者修改 config.json：

```
"Region":"accelerate",
```

内容审核
图片审核

最近更新时间：2022-12-20 14:12:01

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于图片审核的 API 概览和 SDK 示例代码。

API	操作描述
图片单次审核	对对象存储（Cloud Object Storage，COS）存量数据进行涉黄、违法违规以及广告引导类图片的扫描
图片批量审核	对多个图片进行批量审核
查询图片审核任务结果	用于查询图片审核任务详情

图片单次审核

功能说明

图片审核的存量扫描功能通过借助数据万象的持久化处理接口，实现对 COS 存量数据的涉黄、违法违规以及广告引导类图片的扫描。

方法原型

```
CosResult GetImageAuditing(const GetImageAuditingReq& req, GetImageAuditingResp*
resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "test.jpg";
GetImageAuditingReq req(bucket_name);
GetImageAuditingResp resp;
// 添加请求参数 参数详情请见api接口文档
req.SetObjectKey(object_name);
req.SetBizType("b81d45f94b91a683255e9a9506f45a11");
// req.SetDetectType("Porn,Ads");
// 调用接口,获取任务响应对象
CosResult result = cos.GetImageAuditing(req, &resp);
if (result.IsSucc()) {
// 创建审核任务成功，可以调用 GetImageAuditingResp 的成员函数
} else {
// 创建审核任务失败，可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	GetImageAuditing 操作的请求	GetImageAuditingReq	是
resp	GetImageAuditing 操作的响应	GetImageAuditingResp	是

GetImageAuditingReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// COS 存储桶中的图片文件名称，需要审核图片文件对象
void SetObjectKey(const std::string& object_name);
// 表示审核策略的唯一标识，您可以通过控制台上的审核策略页面，配置您希望审核的场景，如涉黄、广
告、违法违规等，配置指引： 设置公共审核策略。
// 您可以在控制台上获取到 BizType。BizType 填写时，此条审核请求将按照该审核策略中配置的场景进
行审核。
void SetBizType(const std::string& biz_type);
// 审核的场景类型，有效值：Porn（涉黄）、Ads（广告）等，可以传入多种类型，不同类型以逗号分隔，
例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。
```

```
void SetDetectType(const std::string& detect_type);
// 审核图片Url，可为任意公网可访问图片链接
// 设置了detect-url时，默认审核detect-url，无需填写ObjectKey
// 不设置detect-url时，默认审核ObjectKey
void SetDetectUrl(const std::string& detect_url);
// 审核GIF动图时，使用该参数截帧审核，例如值设为5，则表示从第1帧开始截取，每隔5帧截取一帧，默认值5
void SetInterval(int interval);
// 审核动图时最大截帧数量，默认为5
void SetMaxFrames(int max_frames);
// 对于超过大小限制的图片是否进行压缩后再审核，取值为： 0（不压缩），1（压缩）。默认为0。
void SetLargeImageDetect(int large_image_detect);
// 图片标识，该字段返回原始内容，长度限制为512字节
void SetDataId(const std::string& data_id);
```

GetImageAuditingResp 提供以下成员函数：

```
// 获取API请求执行返回的任务详情
ImageAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 ImageAuditingJobsDetail 结构中，具体返回参数可查看 图片单次审核 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

图片批量审核

功能说明

图片批量审核接口为同步请求方式，您可以通过本接口对多个图片文件进行内容审核。该接口属于 POST 请求。

方法原型

```
CosResult CosAPI::BatchImageAuditing(const BatchImageAuditingReq& req, BatchImageAuditingResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
```

```
std::string object_name_a = "test1.jpg";
std::string object_name_b = "test2.jpg";
BatchImageAuditingReq req(bucket_name);
BatchImageAuditingResp resp;
// input 1
AuditingInput input_a = AuditingInput();
input_a.SetObject(object_name_a);
req.AddInput(input_a);
// input2
AuditingInput input_b = AuditingInput();
input_a.SetObject(object_name_b);
req.AddInput(input_b);
// 审核配置
req.SetBizType("b81d45f94b91a683255e9a9506f45a11");
// req.SetDetectType("Porn,Ads");
// 调用接口,获取任务响应对象
CosResult result = cos.BatchImageAuditing(req, &resp);
if (result.IsSucc()) {
// 创建审核任务成功，可以调用 BatchImageAuditingResp 的成员函数
} else {
// 创建审核任务失败，可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	BatchImageAuditing 操作的请求	BatchImageAuditingReq	是
resp	BatchImageAuditing 操作的响应	BatchImageAuditingResp	是
input	BatchImageAuditing 操作请求的Input参数	AuditingInput	是

BatchImageAuditingReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// 设置审核配置
void SetConf(const Conf& conf);
// 表示审核策略的唯一标识，您可以通过控制台上的审核策略页面，配置您希望审核的场景，如涉黄、广告、违法违规等，配置指引： 设置公共审核策略。
// 您可以在控制台上获取到 BizType。BizType 填写时，此条审核请求将按照该审核策略中配置的场景进行审核。
void SetBizType(const std::string& biz_type);
// 审核的场景类型，有效值：Porn（涉黄）、Ads（广告）等，可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。
```

```
void SetDetectType(const std::string& detect_type);
// 添加单个图片审核的Input
void AddInput(const AuditingInput& input);
// 设置需要批量审核的Input数组
void SetInputs(const std::vector<AuditingInput>& inputs);
```

AuditingInput 提供以下成员函数：

```
// COS 存储桶中的图片文件名称，需要审核图片文件对象
void SetObject(const std::string& object);
// 图片文件的链接地址，例如 http://a-1250000.cos.ap-shanghai.myqcloud.com/image.jpg。Object 和 Url 只能选择其中一种。
void SetUrl(const std::string& url);
// 截帧频率，GIF 图检测专用，默认值为5，表示从第一帧（包含）开始每隔5帧截取一帧
void SetInterval(const int interval);
// 最大截帧数量，GIF 图检测专用，默认值为5，表示只截取 GIF 的5帧图片进行审核，必须大于0
void SetMaxFrames(const int max_frames);
// 图片标识，该字段在结果中返回原始内容，长度限制为512字节
void SetDataId(const std::string& data_id);
// 对于超过大小限制的图片是否进行压缩后再审核，取值为： 0（不压缩），1（压缩）。默认为0。
void SetLargeImageDetect(const int large_image_detect);
// 用户业务字段。
void SetUserInfo(const UserInfo& user_info);
```

BatchImageAuditingResp 提供以下成员函数：

```
// 获取API请求执行返回的任务详情
std::vector<ImageAuditingJobsDetail> GetJobsDetails();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 ImageAuditingJobsDetail 结构数组中，具体返回参数可查看 图片批量审核 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

查询图片审核任务结果

功能说明

用于提交一个图片审核任务。

方法原型

```
CosResult DescribeImageAuditingJob(const DescribeImageAuditingJobReq &req, Descri
beImageAuditingJobResp *resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
DescribeImageAuditingJobReq req(bucket_name);
DescribeImageAuditingJobResp resp;
// 添加请求参数 参数详情请见api接口文档
req.SetJobId("vab1ca9fc8a3ed11ea834c525400863904");
// 调用接口,获取任务响应对象
CosResult result = cos.DescribeImageAuditingJob(req, &resp);
if (result.IsSucc()) {
// 创建审核任务成功, 可以调用 DescribeImageAuditingJobResp 的成员函数
} else {
// 创建审核任务失败, 可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	DescribeImageAuditingJob 操作的请求	DescribeImageAuditingJobReq	是
resp	DescribeImageAuditingJob 操作的响应	DescribeImageAuditingJobResp	是

DescribeVideoAuditingJobReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// 设置查询的审核任务ID
void SetJobId(const std::string& job_id);
```

DescribeVideoAuditingJobResp 提供的成员函数如下：

```
// 获取API请求执行返回的任务详情
ImageAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 ImageAuditingJobsDetail 结构中，具体返回参数可查看 查询图片审核任务结果 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

视频审核

最近更新时间：2022-12-20 14:12:01

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于视频审核的 API 概览和 SDK 示例代码。

API	操作描述
提交视频审核任务	用于提交一个视频审核任务
查询视频审核任务结果	用于查询指定的视频审核任务

提交视频审核任务

功能说明

用于提交一个视频审核任务。

方法原型

```
CosResult CreateVideoAuditingJob(const CreateVideoAuditingJobReq& req, CreateVideoAuditingJobResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
```

```
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "test.mp4";
CreateVideoAuditingJobReq req(bucket_name);
CreateVideoAuditingJobResp resp;
// 添加请求参数 参数详情请见api接口文档
req.SetObject(object_name);
req.SetBizType("b81d45f94b91a683255e9a9506f45a11");
// req.SetDetectType("Porn,Ads");
SnapShotConf snap_shot = SnapShotConf();
snap_shot.SetCount(100);
snap_shot.SetMode("Interval");
snap_shot.SetTimeInterval(10);
req.SetSnapShot(snap_shot);
// 调用接口,获取任务响应对象
CosResult result = cos.CreateVideoAuditingJob(req, &resp);
if (result.IsSucc()) {
// 创建审核任务成功，可以调用 CreateVideoAuditingJobResp 的成员函数
} else {
// 创建审核任务失败，可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	CreateVideoAuditing 操作的请求	CreateVideoAuditingReq	是
resp	CreateVideoAuditing 操作的响应	CreateVideoAuditingResp	是

CreateVideoAuditingReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// 设置input需要审核的视频
void SetInput(const AuditingInput& input);
// 设置审核配置规则
void SetConf(const Conf& conf);
// conf
// 表示审核策略的唯一标识，您可以通过控制台上的审核策略页面，配置您希望审核的场景，如涉黄、广告、违法违规等，配置指引： 设置公共审核策略。
// 您可以在控制台上获取到 BizType。BizType 填写时，此条审核请求将按照该审核策略中配置的场景进行审核。
void SetBizType(const std::string& biz_type);
// 审核的场景类型，有效值：Porn（涉黄）、Ads（广告）等，可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。
```

```
void SetDetectType(const std::string& detect_type);
// 视频画面的审核通过视频截帧能力截取出一定量的截图，通过对截图逐一审核而实现的，该参数用于指定视频截帧的配置。
void SetSnapShot(const SnapShotConf& snap_shot);
// 回调地址，以http://或者https://开头的地址。
void SetCallBack(const std::string& callback);
// 回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。
void SetCallBackVersion(const std::string& callback_version);
// 用于指定是否审核视频声音，当值为0时：表示只审核视频画面截图；值为1时：表示同时审核视频画面截图和视频声音。默认值为0。
void SetDetectContent(const int detect_content);
// input
// 当前 COS 存储桶中的视频文件名称，例如在目录 test 中的文件 video.mp4，则文件名称为 test/video.mp4。
void SetObject(const std::string& object);
// 视频文件的链接地址，例如 http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/test.mp4。Object 和 Url 只能选择其中一种。
void SetUrl(const std::string& url);
// 该字段在审核结果中会返回原始内容，长度限制为512字节。您可以使用该字段对待审核的数据进行唯一业务标识。
void SetDataId(const std::string& data_id);
// 用户业务字段。
void SetUserInfo(const UserInfo& user_info);
```

CreateVideoAuditingResp 提供的成员函数如下：

```
// 获取API请求执行返回的任务详情
VideoAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 VideoAuditingJobsDetail 结构中，具体返回参数可查看 提交视频审核任务 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

查询视频审核任务结果

功能说明

用于查询视频审核任务执行情况和结果。

方法原型

```
CosResult DescribeVideoAuditingJob(const DescribeVideoAuditingJobReq& req, Descri
beVideoAuditingJobResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
DescribeVideoAuditingJobReq req(bucket_name);
DescribeVideoAuditingJobResp resp;
// 添加请求参数 参数详情请见api接口文档
req.SetJobId("vab1ca9fc8a3ed11ea834c525400863904");
// 调用接口,获取任务响应对象
CosResult result = cos.DescribeVideoAuditingJob(req, &resp);
if (result.IsSucc()) {
// 查询审核任务成功, 可以调用 DescribeVideoAuditingJobResp 的成员函数
} else {
// 查询审核任务失败, 可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	DescribeVideoAuditingJob 操作的请求	DescribeVideoAuditingJobReq	是
resp	DescribeVideoAuditingJob 操作的响应	DescribeVideoAuditingJobResp	是

DescribeVideoAuditingJobReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// 设置查询的审核任务ID
void SetJobId(const std::string& job_id);
```

DescribeVideoAuditingJobResp 提供的成员函数如下：

```
// 获取API请求执行返回的任务详情
VideoAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 VideoAuditingJobsDetail 结构中，具体返回参数可查看 提交视频审核任务 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

音频审核

最近更新时间：2022-12-20 14:12:01

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于音频审核的 API 概览和 SDK 示例代码。

API	操作描述
提交音频审核任务	用于提交一个音频审核任务
查询音频审核任务结果	用于查询指定的音频审核任务

提交音频审核任务

功能说明

用于提交一个音频审核任务。

方法原型

```
CosResult CosAPI::CreateAudioAuditingJob(const CreateAudioAuditingJobReq& req, Cr
eateAudioAuditingJobResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
```

```
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "test.mp3";
CreateAudioAuditingJobReq req(bucket_name);
CreateAudioAuditingJobResp resp;
// 添加请求参数 参数详情请见api接口文档
req.SetObject(object_name);
req.SetBizType("b81d45f94b91a683255e9a9506f45a11");
// req.SetDetectType("Porn,Ads");
// 调用接口,获取任务响应对象
CosResult result = cos.CreateAudioAuditingJob(req, &resp);
if (result.IsSucc()) {
// 创建审核任务成功，可以调用 CreateAudioAuditingJobResp 的成员函数
} else {
// 创建审核任务失败，可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	CreateAudioAuditing 操作的请求	CreateAudioAuditingJobReq	是
resp	CreateAudioAuditing 操作的响应	CreateAudioAuditingJobResp	是

CreateAudioAuditingJobReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// 设置input需要审核的视频
void SetInput(const AuditingInput& input);
// 设置审核配置规则
void SetConf(const Conf& conf);
// conf
// 表示审核策略的唯一标识，您可以通过控制台上的审核策略页面，配置您希望审核的场景，如涉黄、广告、违法违规等，配置指引： 设置公共审核策略。
// 您可以在控制台上获取到 BizType。BizType 填写时，此条审核请求将按照该审核策略中配置的场景进行审核。
void SetBizType(const std::string& biz_type);
// 审核的场景类型，有效值：Porn（涉黄）、Ads（广告）等，可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。
void SetDetectType(const std::string& detect_type);
// 回调地址，以http://或者https://开头的地址。
void SetCallBack(const std::string& callback);
// 回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。
```

```
void SetCallBackVersion(const std::string& callback_version);
// input
// 存储在 COS 存储桶中的音频文件名称，例如在目录 test 中的文件audio.mp3，则文件名称为 test/audio.mp3。Object 和 Url 只能选择其中一种。
void SetObject(const std::string& object);
// 音频文件的链接地址，例如 http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/audio.mp3。Object 和 Url 只能选择其中一种。
void SetUrl(const std::string& url);
// 该字段在审核结果中会返回原始内容，长度限制为512字节。您可以使用该字段对待审核的数据进行唯一业务标识。
void SetDataId(const std::string& data_id);
// 用户业务字段。
void SetUserInfo(const UserInfo& user_info);
```

CreateAudioAuditingJobResp 提供的成员函数如下：

```
// 获取API请求执行返回的任务详情
AudioAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 AudioAuditingJobsDetail 结构中，具体返回参数可查看 提交音频审核任务 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

查询音频审核任务结果

功能说明

用于查询音频审核任务执行情况和结果。

方法原型

```
CosResult DescribeAudioAuditingJob(const DescribeAudioAuditingJobReq& req, DescribeAudioAuditingJobResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
DescribeAudioAuditingJobReq req(bucket_name);
```

```
DescribeAudioAuditingJobResp resp;
// 添加请求参数 参数详情请见api接口文档
req.SetJobId("aab1ca9fc8a3ed11ea834c525400863904");
// 调用接口,获取任务响应对象
CosResult result = cos.DescribeAudioAuditingJob(req, &resp);
if (result.IsSucc()) {
// 查询审核任务成功, 可以调用 DescribeAudioAuditingJobResp 的成员函数
} else {
// 查询审核任务失败, 可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	DescribeAudioAuditingJob 操作的请求	DescribeAudioAuditingJobReq	是
resp	DescribeAudioAuditingJob 操作的响应	DescribeAudioAuditingJobResp	是

DescribeAudioAuditingJobReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// 设置查询的审核任务ID
void SetJobId(const std::string& job_id);
```

DescribeAudioAuditingJobResp 提供的成员函数如下：

```
// 获取API请求执行返回的任务详情
AudioAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 AudioAuditingJobsDetail 结构中，具体返回参数可查看 查询音频审核任务结果 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

文本审核

最近更新时间：2022-12-20 14:12:01

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于文本审核的 API 概览和 SDK 示例代码。

API	操作描述
提交文本审核任务	用于提交一个文本审核任务
查询文本审核任务结果	用于查询指定的文本审核任务

提交文本审核任务

功能说明

用于提交一个文本审核任务。

方法原型

```
CosResult CosAPI::CreateTextAuditingJob(const CreateTextAuditingJobReq& req, Crea
teTextAuditingJobResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
```

```
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "test.txt";
CreateTextAuditingJobReq req(bucket_name);
CreateTextAuditingJobResp resp;
// 添加请求参数 参数详情请见api接口文档
req.SetObject(object_name);
req.SetBizType("b81d45f94b91a683255e9a9506f45a11");
// req.SetDetectType("Porn,Ads");
// 调用接口,获取任务响应对象
CosResult result = cos.CreateTextAuditingJob(req, &resp);
if (result.IsSucc()) {
// 创建审核任务成功，可以调用 CreateTextAuditingJobResp 的成员函数
} else {
// 创建审核任务失败，可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	CreateTextAuditing 操作的请求	CreateTextAuditingJobReq	是
resp	CreateTextAuditing 操作的响应	CreateTextAuditingJobResp	是

CreateTextAuditingJobReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// 设置input需要审核的视频
void SetInput(const AuditingInput& input);
// 设置审核配置规则
void SetConf(const Conf& conf);
// conf
// 表示审核策略的唯一标识，您可以通过控制台上的审核策略页面，配置您希望审核的场景，如涉黄、广告、违法违规等，配置指引： 设置公共审核策略。
// 您可以在控制台上获取到 BizType。BizType 填写时，此条审核请求将按照该审核策略中配置的场景进行审核。
void SetBizType(const std::string& biz_type);
// 审核的场景类型，有效值：Porn（涉黄）、Ads（广告）等，可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。
void SetDetectType(const std::string& detect_type);
// 回调地址，以http://或者https://开头的地址。
void SetCallBack(const std::string& callback);
// 回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。
```

```
void SetCallBackVersion(const std::string& callback_version);
// input
// 当前 COS 存储桶中的文本文件名称，例如在目录 test 中的文件 test.txt，则文件名称为 test/test.txt，文本文件仅支持 UTF8 编码和 GBK 编码的内容，且文件大小不得超过1MB。
void SetObject(const std::string& object);
// 文本文件的完整链接，例如：https://www.test.com/test.txt。
void SetUrl(const std::string& url);
// 该字段在审核结果中会返回原始内容，长度限制为512字节。您可以使用该字段对待审核的数据进行唯一业务标识。
void SetDataId(const std::string& data_id);
// 用户业务字段。
void SetUserInfo(const UserInfo& user_info);
// 当传入的内容为纯文本信息，需要先经过 base64 编码，文本编码前的原文长度不能超过10000个 UTF8 编码字符。若超出长度限制，接口将会报错。
void SetContent(const std::string& content) { m_input.SetContent(content); }
```

CreateTextAuditingJobResp 提供的成员函数如下：

```
// 获取API请求执行返回的任务详情
TextAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 TextAuditingJobsDetail 结构中，具体返回参数可查看 提交文本审核任务 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

查询文本审核任务结果

功能说明

用于查询文本审核任务执行情况和结果。

方法原型

```
CosResult DescribeTextAuditingJob(const DescribeTextAuditingJobReq& req, DescribeTextAuditingJobResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
```

```
std::string bucket_name = "examplebucket-1250000000";
DescribeTextAuditingJobReq req(bucket_name);
DescribeTextAuditingJobResp resp;
// 添加请求参数 参数详情请见api接口文档
req.SetJobId("aab1ca9fc8a3ed11ea834c525400863904");
// 调用接口,获取任务响应对象
CosResult result = cos.DescribeTextAuditingJob(req, &resp);
if (result.IsSucc()) {
// 查询审核任务成功，可以调用 DescribeTextAuditingJobResp 的成员函数
} else {
// 查询审核任务失败，可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	DescribeTextAuditingJob 操作的请求	DescribeTextAuditingJobReq	是
resp	DescribeTextAuditingJob 操作的响应	DescribeTextAuditingJobResp	是

DescribeTextAuditingJobReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// 设置查询的审核任务ID
void SetJobId(const std::string& job_id);
```

DescribeTextAuditingJobResp 提供的成员函数如下：

```
// 获取API请求执行返回的任务详情
TextAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 TextAuditingJobsDetail 结构中，具体返回参数可查看 查询文本审核任务结果 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

文档审核

最近更新时间：2022-12-20 14:12:01

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于文档审核的 API 概览和 SDK 示例代码。

API	操作描述
提交文档审核任务	用于提交一个文档审核任务
查询文档审核任务结果	用于查询指定的文档审核任务

提交文档审核任务

功能说明

用于提交一个文档审核任务。

方法原型

```
CosResult CosAPI::CreateDocumentAuditingJob(const CreateDocumentAuditingJobReq& r
eq, CreateDocumentAuditingJobResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
```

```
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "test.docx";
CreateDocumentAuditingJobReq req(bucket_name);
CreateDocumentAuditingJobResp resp;
// 添加请求参数 参数详情请见api接口文档
req.SetObject(object_name);
req.SetType("docx");
req.SetBizType("b81d45f94b91a683255e9a9506f45a11");
// req.SetDetectType("Porn,Ads");
// 调用接口,获取任务响应对象
CosResult result = cos.CreateDocumentAuditingJob(req, &resp);
if (result.IsSucc()) {
// 创建审核任务成功，可以调用 CreateDocumentAuditingJobResp 的成员函数
} else {
// 创建审核任务失败，可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	CreateDocumentAuditing 操作的请求	CreateDocumentAuditingJobReq	是
resp	CreateDocumentAuditing 操作的响应	CreateDocumentAuditingJobResp	是

CreateDocumentAuditingJobReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// 设置input需要审核的视频
void SetInput(const AuditingInput& input);
// 设置审核配置规则
void SetConf(const Conf& conf);
// conf
// 表示审核策略的唯一标识，您可以通过控制台上的审核策略页面，配置您希望审核的场景，如涉黄、广告、违法违规等，配置指引： 设置公共审核策略。
// 您可以在控制台上获取到 BizType。BizType 填写时，此条审核请求将按照该审核策略中配置的场景进行审核。
void SetBizType(const std::string& biz_type);
// 审核的场景类型，有效值：Porn（涉黄）、Ads（广告）等，可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。
void SetDetectType(const std::string& detect_type);
// 回调地址，以http://或者https://开头的地址。
void SetCallBack(const std::string& callback);
```

```
// input
// 存储在 COS 存储桶中的文件名称，例如在目录 test 中的文件test.doc，则文件名称为 test/test.doc。Object 和 Url 只能选择其中一种。
void SetObject(const std::string& object);
// 文档文件的链接地址，例如 http://www.example.com/doctest.doc，Object 和 Url 只能选择其中一种。
void SetUrl(const std::string& url);
// 该字段在审核结果中会返回原始内容，长度限制为512字节。您可以使用该字段对待审核的数据进行唯一业务标识。
void SetDataId(const std::string& data_id);
// 用户业务字段。
void SetUserInfo(const UserInfo& user_info);
// 指定文档文件的类型，如未指定则默认以文件的后缀为类型。
void SetType(const std::string& type);
```

CreateDocumentAuditingJobResp 提供的成员函数如下：

```
// 获取API请求执行返回的任务详情
DocumentAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 DocumentAuditingJobsDetail 结构中，具体返回参数可查看 提交文档审核任务 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

查询文档审核任务结果

功能说明

用于查询文档审核任务执行情况和结果。

方法原型

```
CosResult DescribeDocumentAuditingJob(const DescribeDocumentAuditingJobReq& req, DescribeDocumentAuditingJobResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
```

```
DescribeDocumentAuditingJobReq req(bucket_name);
DescribeDocumentAuditingJobResp resp;
// 添加请求参数 参数详情请见api接口文档
req.SetJobId("stb1ca9fc8a34c525400863904****");
// 调用接口,获取任务响应对象
CosResult result = cos.DescribeDocumentAuditingJob(req, &resp);
if (result.IsSucc()) {
// 查询审核任务成功，可以调用 DescribeDocumentAuditingJobResp 的成员函数
} else {
// 查询审核任务失败，可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	DescribeDocumentAuditingJob 操作的请求	DescribeDocumentAuditingJobReq	是
resp	DescribeDocumentAuditingJob 操作的响应	DescribeDocumentAuditingJobResp	是

DescribeDocumentAuditingJobReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// 设置查询的审核任务ID
void SetJobId(const std::string& job_id);
```

DescribeDocumentAuditingJobResp 提供的成员函数如下：

```
// 获取API请求执行返回的任务详情
DocumentAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 DocumentAuditingJobsDetail 结构中，具体返回参数可查看 查询文档审核任务结果 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

版权所有：腾讯云计算（北京）有限责任公司

网页审核

最近更新时间：2022-12-20 14:12:01

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于网页审核的 API 概览和 SDK 示例代码。

API	操作描述
提交网页审核任务	用于提交一个网页审核任务
查询网页审核任务结果	用于查询指定的网页审核任务

提交网页审核任务

功能说明

用于提交一个网页审核任务。

方法原型

```
CosResult CosAPI::CreateWebPageAuditingJob(const CreateWebPageAuditingJobReq& req, CreateWebPageAuditingJobResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
```

```
std::string bucket_name = "examplebucket-1250000000";
std::string url = "http://test.com";
CreateWebPageAuditingJobReq req(bucket_name);
CreateWebPageAuditingJobResp resp;
// 添加请求参数 参数详情请见api接口文档
req.SetUrl(url);
req.SetDetectType("Porn,Ads");
// 调用接口,获取任务响应对象
CosResult result = cos.CreateWebPageAuditingJob(req, &resp);
if (result.IsSucc()) {
// 创建审核任务成功，可以调用 CreateWebPageAuditingJobResp 的成员函数
} else {
// 创建审核任务失败，可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	CreateWebPageAuditing 操作的请求	CreateWebPageAuditingJobReq	是
resp	CreateWebPageAuditing 操作的响应	CreateWebPageAuditingJobResp	是

CreateWebPageAuditingJobReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// 设置input需要审核的视频
void SetInput(const AuditingInput& input);
// 设置审核配置规则
void SetConf(const Conf& conf);
// conf
// 审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。
void SetDetectType(const std::string& detect_type);
// 回调地址，以http://或者https://开头的地址。
void SetCallBack(const std::string& callback);
// 指定是否需要高亮展示网页内的违规文本，查询及回调结果时会根据此参数决定是否返回高亮展示的 html 内容。取值为 true 或者 false，默认为 false。
void SetReturnHighligthHtml(const bool return_highlight_html);
// input
// 网页文件的链接地址，例如 http://www.test.com。
void SetUrl(const std::string& url);
// 该字段在审核结果中会返回原始内容，长度限制为512字节。您可以使用该字段对待审核的数据进行唯一业务标识。
```

```
void SetDataId(const std::string& data_id);
// 用户业务字段。
void SetUserInfo(const UserInfo& user_info);
```

CreateWebPageAuditingJobResp 提供的成员函数如下：

```
// 获取API请求执行返回的任务详情
WebPageAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 WebPageAuditingJobsDetail 结构中，具体返回参数可查看 提交网页审核任务 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

查询网页审核任务结果

功能说明

用于查询网页审核任务执行情况和结果。

方法原型

```
CosResult DescribeWebPageAuditingJob(const DescribeWebPageAuditingJobReq& req, De
scribeWebPageAuditingJobResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";
DescribeWebPageAuditingJobReq req(bucket_name);
DescribeWebPageAuditingJobResp resp;
// 添加请求参数 参数详情请见api接口文档
req.SetJobId("aab1ca9fc8a3ed11ea834c525400863904");
// 调用接口,获取任务响应对象
CosResult result = cos.DescribeWebPageAuditingJob(req, &resp);
if (result.IsSucc()) {
// 查询审核任务成功，可以调用 DescribeWebPageAuditingJobResp 的成员函数
} else {
// 查询审核任务失败，可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	DescribeWebPageAuditingJob 操作的请求	DescribeWebPageAuditingJobReq	是
resp	DescribeWebPageAuditingJob 操作的响应	DescribeWebPageAuditingJobResp	是

DescribeWebPageAuditingJobReq 提供以下成员函数：

```
// 设置执行操作的bucket
void SetBucketName(const std::string& bucket_name);
// 设置查询的审核任务ID
void SetJobId(const std::string& job_id);
```

DescribeWebPageAuditingJobResp 提供的成员函数如下：

```
// 获取API请求执行返回的任务详情
WebPageAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 WebPageAuditingJobsDetail 结构中，具体返回参数可查看查询网页审核任务结果 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 异常处理。

.NET(C#) SDK 快速入门

最近更新时间：2023-12-12 16:33:18

相关资源

SDK 源码下载请参见：XML .NET SDK。
SDK 快速下载地址：XML .NET SDK。
SDK 文档中的所有示例代码请参见 SDK 代码示例。
SDK 更新日志请参见 ChangeLog。
SDK 常见问题请参见：.NET（C#）SDK 常见问题。

说明

如果您在使用 SDK 时遇到函数或方法不存在等错误，请先将 SDK 升级到最新版再重试。

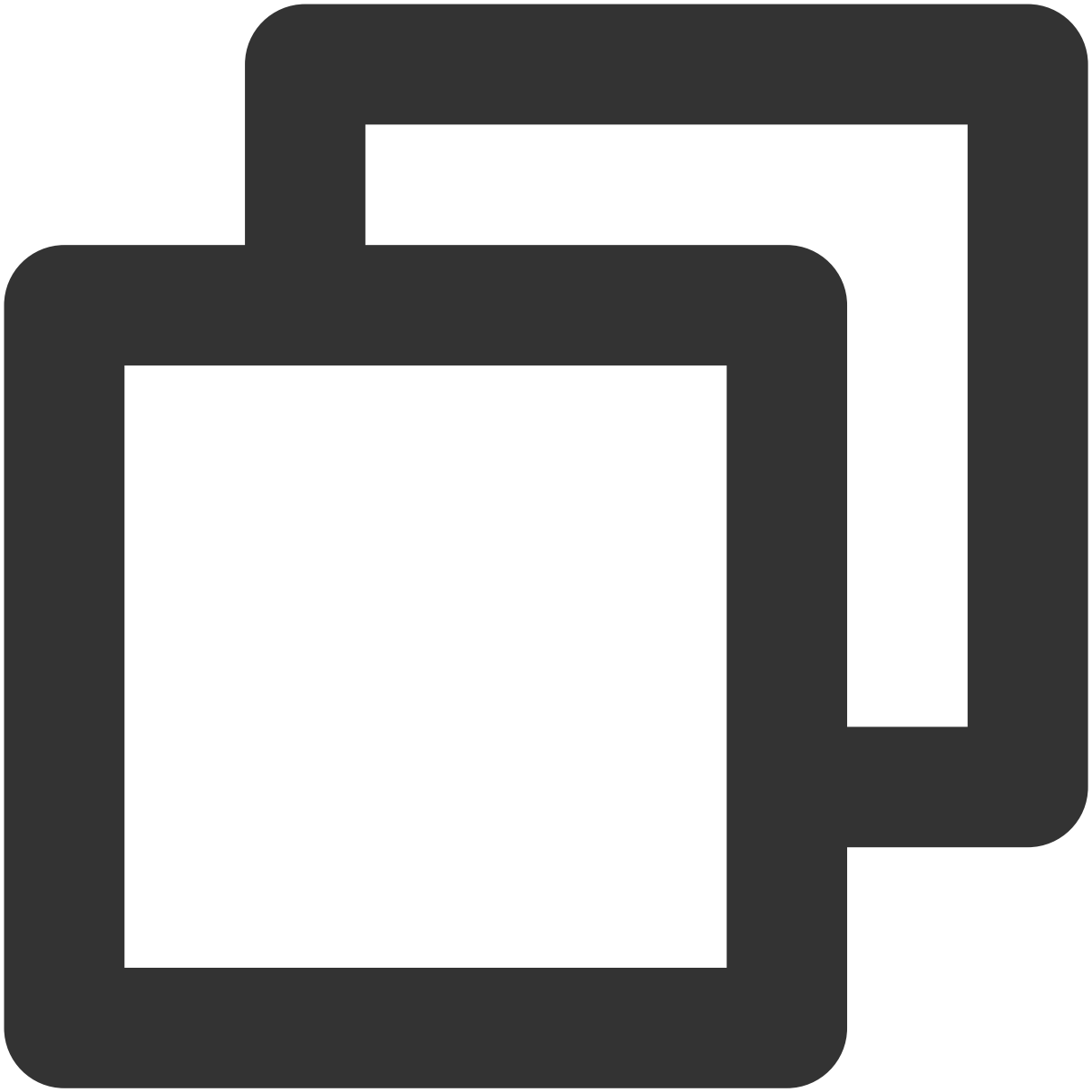
第一步：集成 SDK

环境依赖

.NET SDK 基于 .NET Standard 2.0 开发。
Windows：安装 .NET Core 2.0 及以上版本，或者 .NET Framework 2.0 及以上版本。
Linux/Mac：安装 .NET Core 2.0 及以上版本。

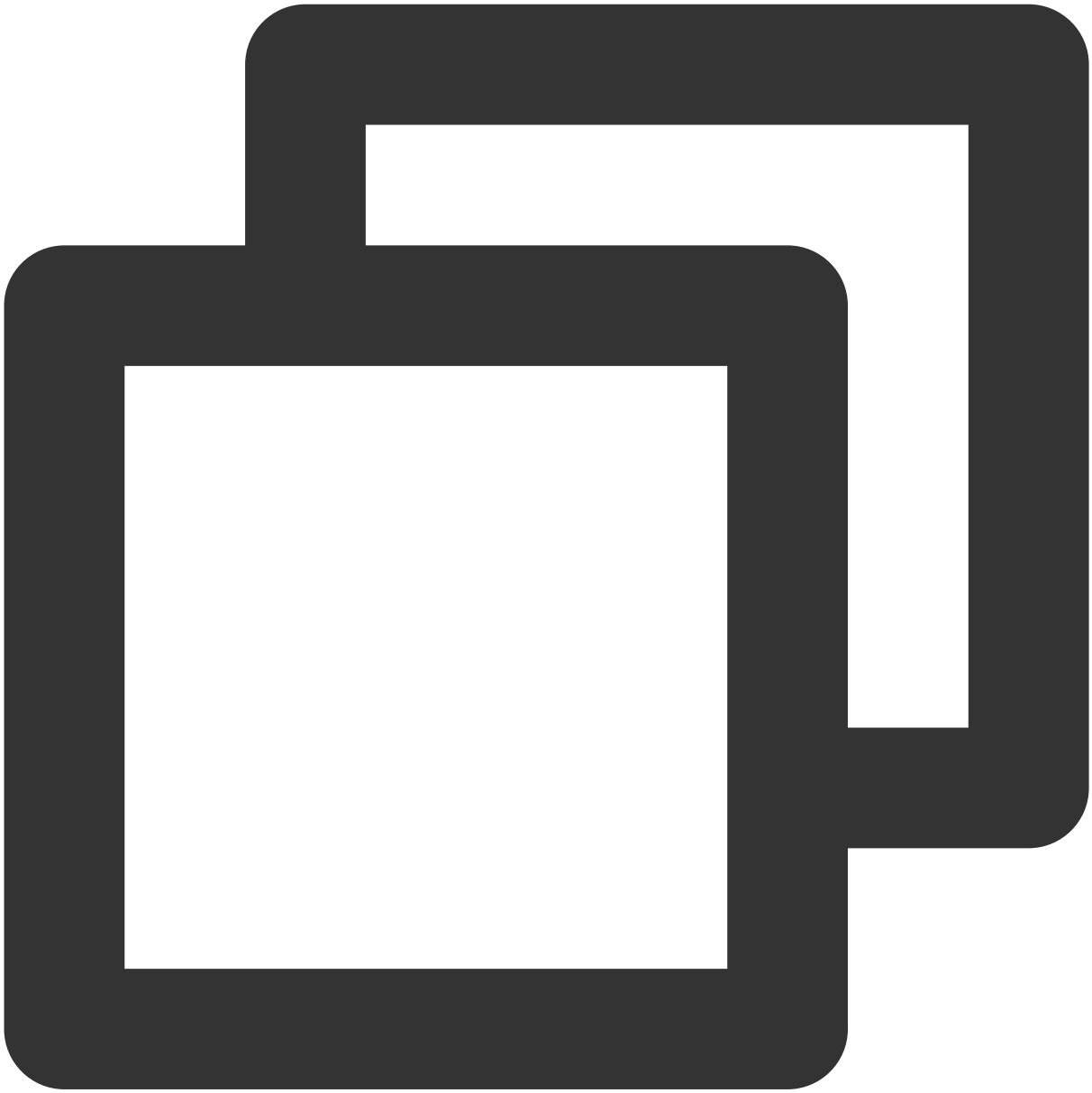
添加 SDK

我们提供 `Nuget` 的集成方式，您可以在工程的 `csproj` 文件里添加：



```
<PackageReference Include="Tencent.QCloud.Cos.Sdk" Version="5.4.*" />
```

如果您是使用 .NET CLI，请使用如下命令安装：



```
dotnet add package Tencent.QCloud.Cos.Sdk
```

如果您是用于目标框架 .Net Framework 4.0 及以下版本的开发，请下载 Releases 并使用 COSXML-Compatible.dll 文件。

在 Visual Studio 项目中选择**项目** > **添加引用** > **浏览** > **COSXML-Compatible.dll** 的方式添加 .NET(C#) SDK。

说明

兼容包中不包含高级上传、下载等功能的支持，详情参见 向下兼容指南。

第二步：初始化 COS 服务

注意

建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。

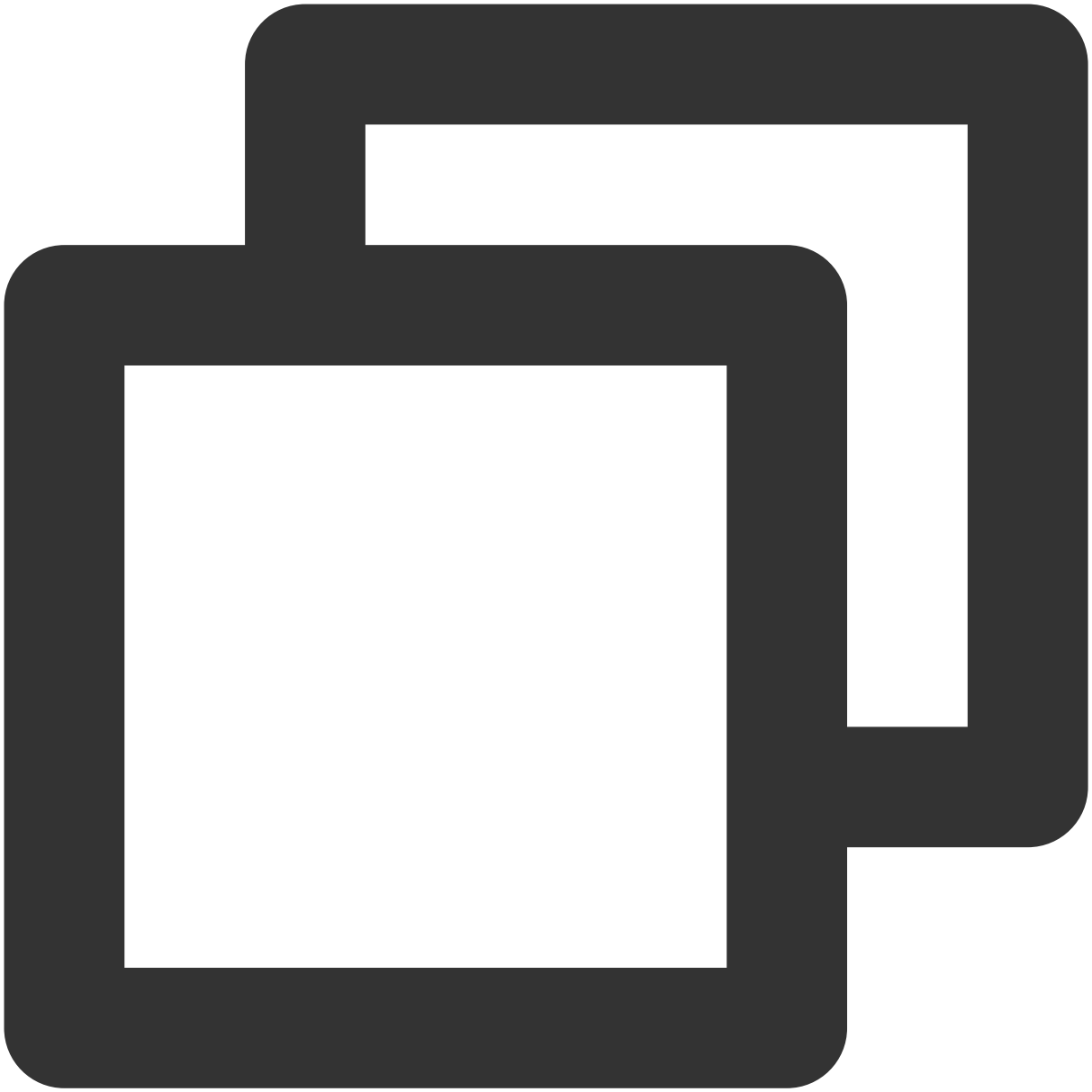
如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

下面为您介绍如何使用 COS .NET SDK 完成一个基础操作，例如初始化客户端、创建存储桶、查询存储桶列表、上传对象、查询对象列表、下载对象和删除对象。

说明

关于文章中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参见 COS 术语信息。

SDK 中常用的命名空间有：



```
using COSXML;
using COSXML.Auth;
using COSXML.Model.Object;
using COSXML.Model.Bucket;
using COSXML.CosException;
```

在执行任何和 COS 服务相关请求之前，都需要先实例化 `CosXmlConfig` ，`QCloudCredentialProvider` ，`CosXmlServer` 3个对象。其中：

`CosXmlConfig` 提供配置 SDK 接口。

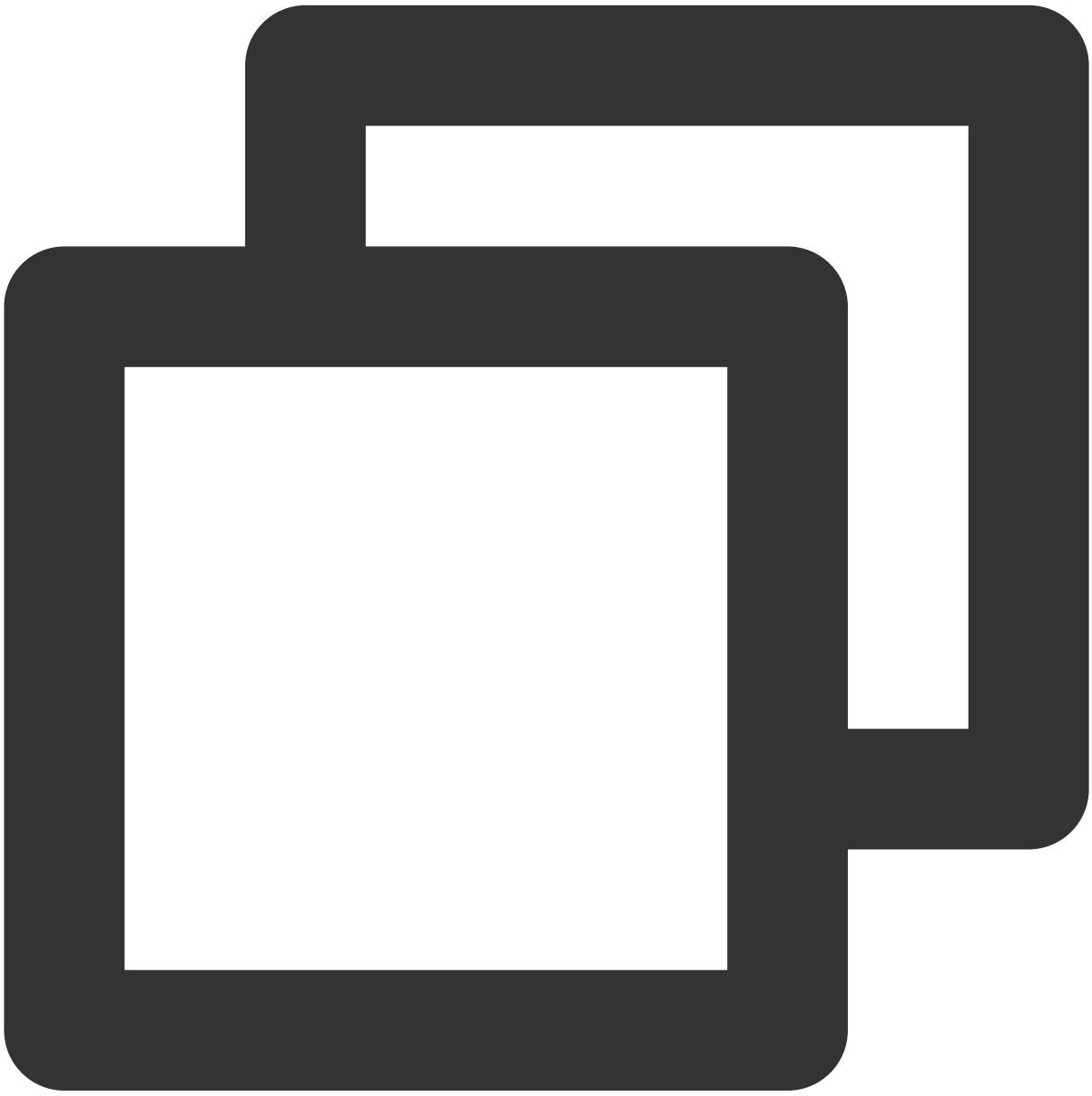
`QCloudCredentialProvider` 提供设置密钥信息接口。

`CosXmlServer` 提供各种 COS API 服务接口。

说明

下述初始化示例中使用的临时密钥，其生成和使用可参见 临时密钥生成及使用指引。

1. 初始化服务设置



```
//初始化 CosXmlConfig
string appid = "1250000000";//设置腾讯云账户的账户标识 APPID
string region = "COS_REGION"; //设置一个默认的存储桶地域
CosXmlConfig config = new CosXmlConfig.Builder()
```

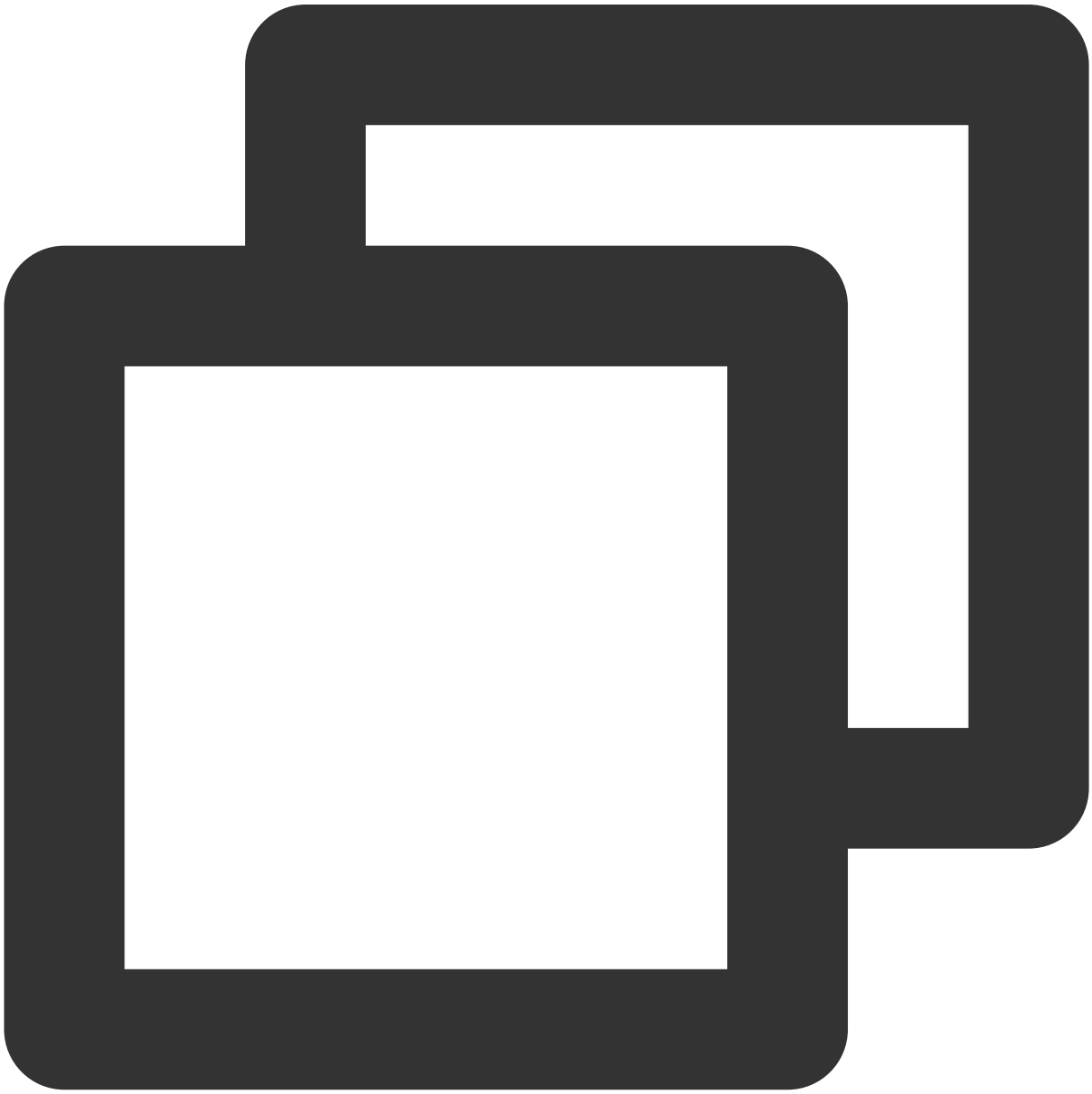
```
    .IsHttps(true)  //设置默认 HTTPS 请求
    .SetRegion(region)  //设置一个默认的存储桶地域
    .SetDebugLog(true)  //显示日志
    .Build();  //创建 CosXmlConfig 对象
```

2. 提供访问凭证

SDK 中提供了3种方式：持续更新的临时密钥、不变的临时密钥、永久密钥。

方式1：持续更新的临时密钥

因为临时密钥在一定时效后过期，以下方式保证可以在过期后自动获取到新的临时密钥。



```
public class CustomQCloudCredentialProvider : DefaultSessionQCloudCredentialProvide
{

  // 这里假设开始没有密钥，也可以用初始的临时密钥来初始化
  public CustomQCloudCredentialProvider(): base(null, null, 0L, null) {
    ;
  }

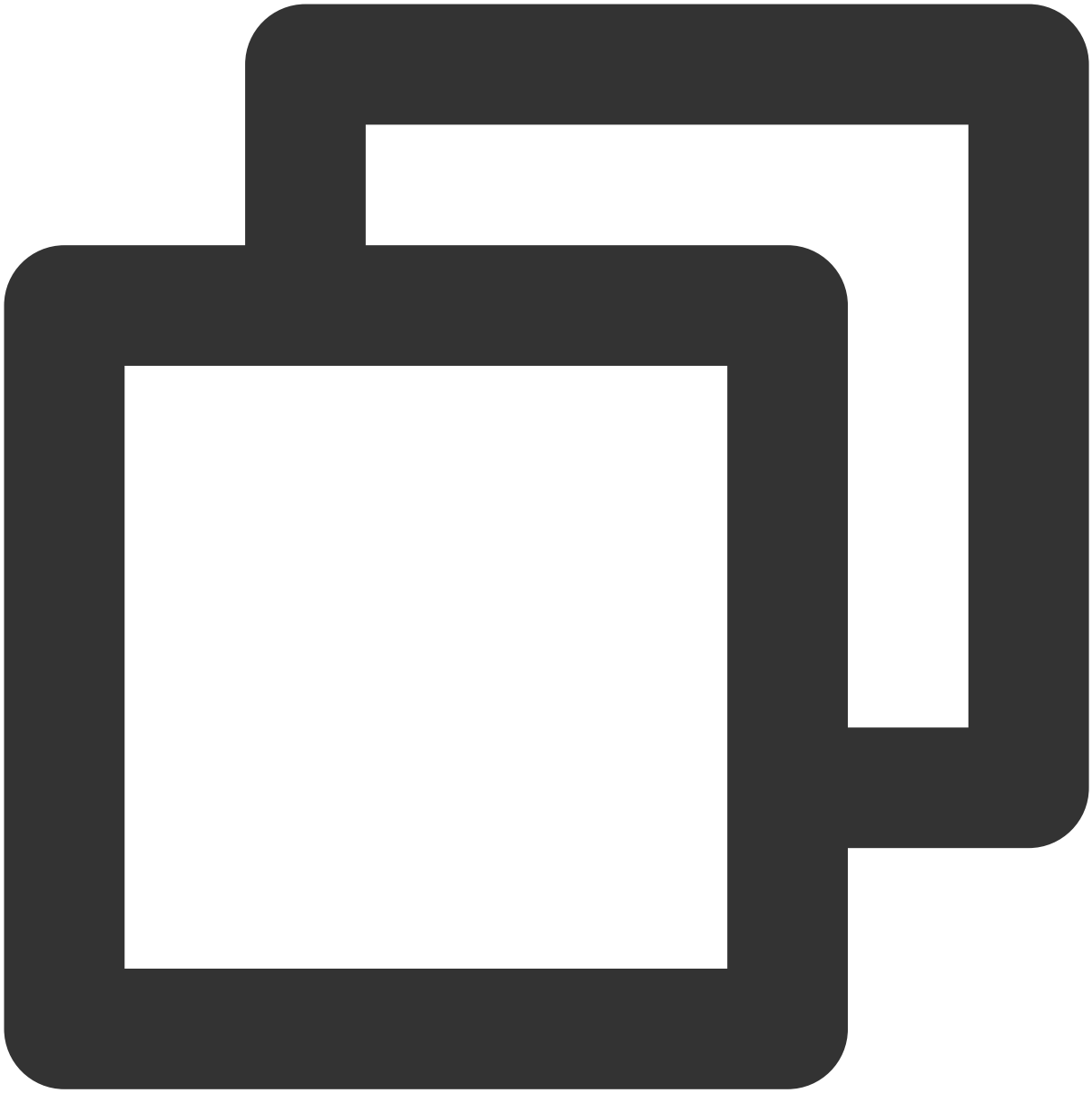
  public override void Refresh()
  {
    //... 首先通过腾讯云请求临时密钥
    string tmpSecretId = "SECRET_ID"; //"临时密钥 SecretId", 临时密钥生成和使用指引参见 h
    string tmpSecretKey = "SECRET_KEY"; //"临时密钥 SecretKey", 临时密钥生成和使用指引参
    string tmpToken = "COS_TOKEN"; //"临时密钥 token", 临时密钥生成和使用指引参见 https:/
    long tmpStartTime = 1546860702;//临时密钥有效开始时间，精确到秒
    long tmpExpiredTime = 1546862502;//临时密钥有效截止时间，精确到秒
    // 调用接口更新密钥
    SetQCloudCredential(tmpSecretId, tmpSecretKey,
      String.Format("{0};{1}", tmpStartTime, tmpExpiredTime), tmpToken);
  }
}

QCloudCredentialProvider cosCredentialProvider = new CustomQCloudCredentialProvider
```

方式2：不变的临时密钥（不建议）

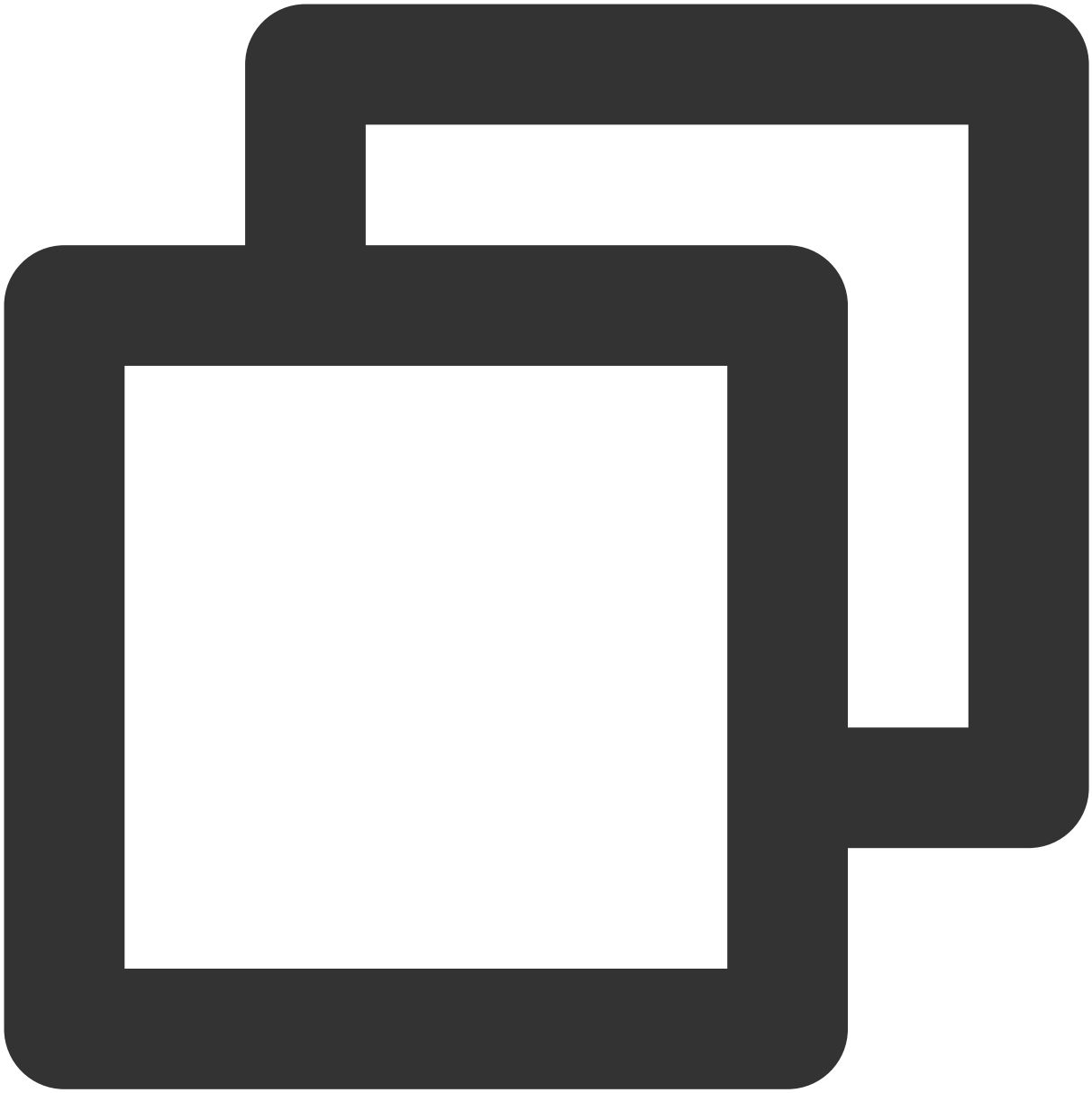
注意

由于临时密钥在一定时效后过期，这种方式在过期后会请求失败，不建议使用。



```
string tmpSecretId = "SECRET_ID"; //"临时密钥 SecretId";
string tmpSecretKey = "SECRET_KEY"; //"临时密钥 SecretKey";
string tmpToken = "COS_TOKEN"; //"临时密钥 token";
long tmpExpireTime = 1546862502;//临时密钥有效截止时间，精确到秒
QCloudCredentialProvider cosCredentialProvider = new DefaultSessionQCloudCredential
  tmpSecretId, tmpSecretKey, tmpExpireTime, tmpToken);
```

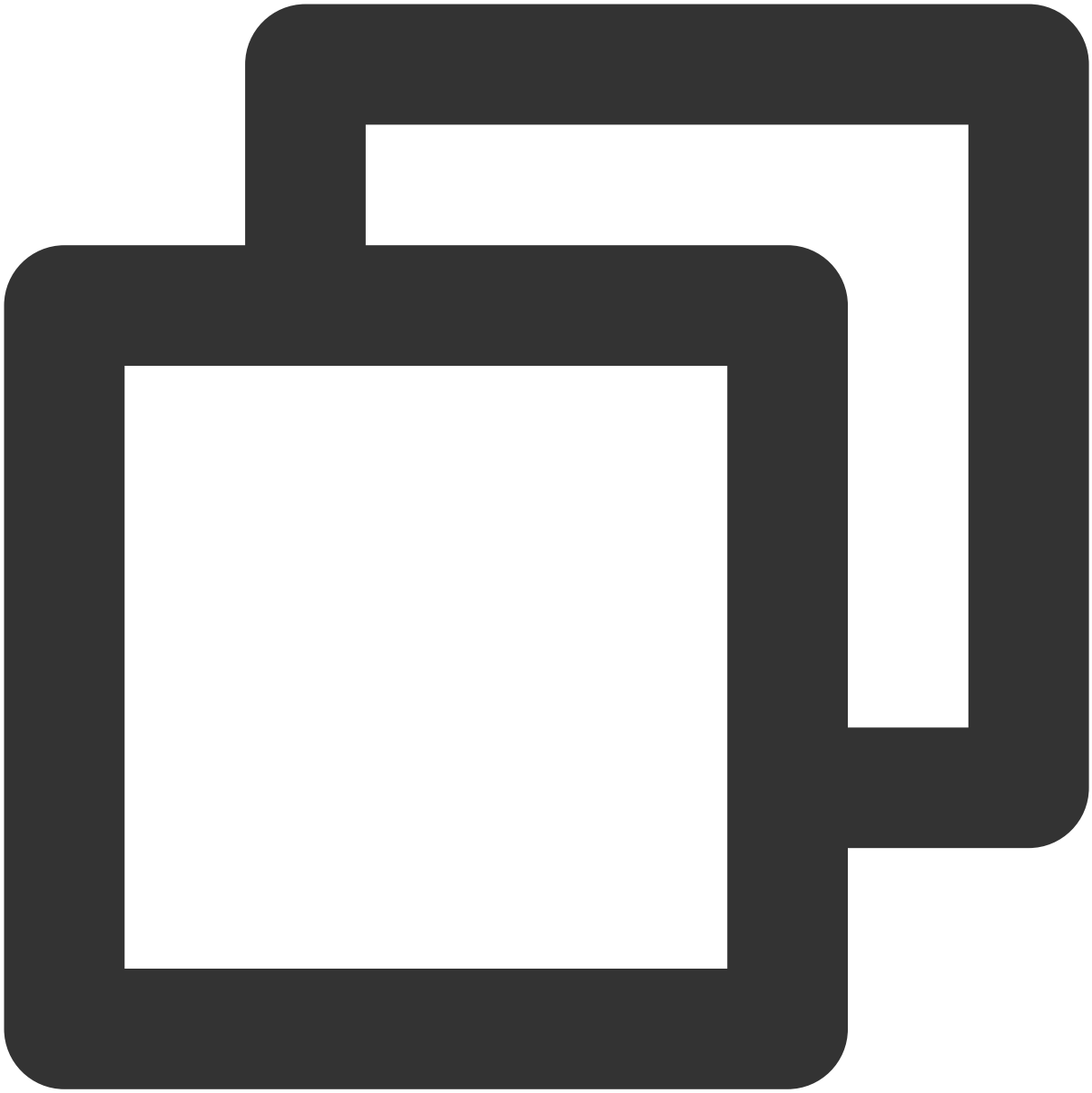
方式3：永久密钥（不建议）



```
string secretId = Environment.GetEnvironmentVariable("SECRET_ID"); //用户的 SecretId
string secretKey = Environment.GetEnvironmentVariable("SECRET_KEY"); //用户的 Secret
long durationSecond = 600;  //每次请求签名有效时长，单位为秒
QCloudCredentialProvider cosCredentialProvider = new DefaultQCloudCredentialProvide
  secretId, secretKey, durationSecond);
```

3. 初始化 CosXmlServer

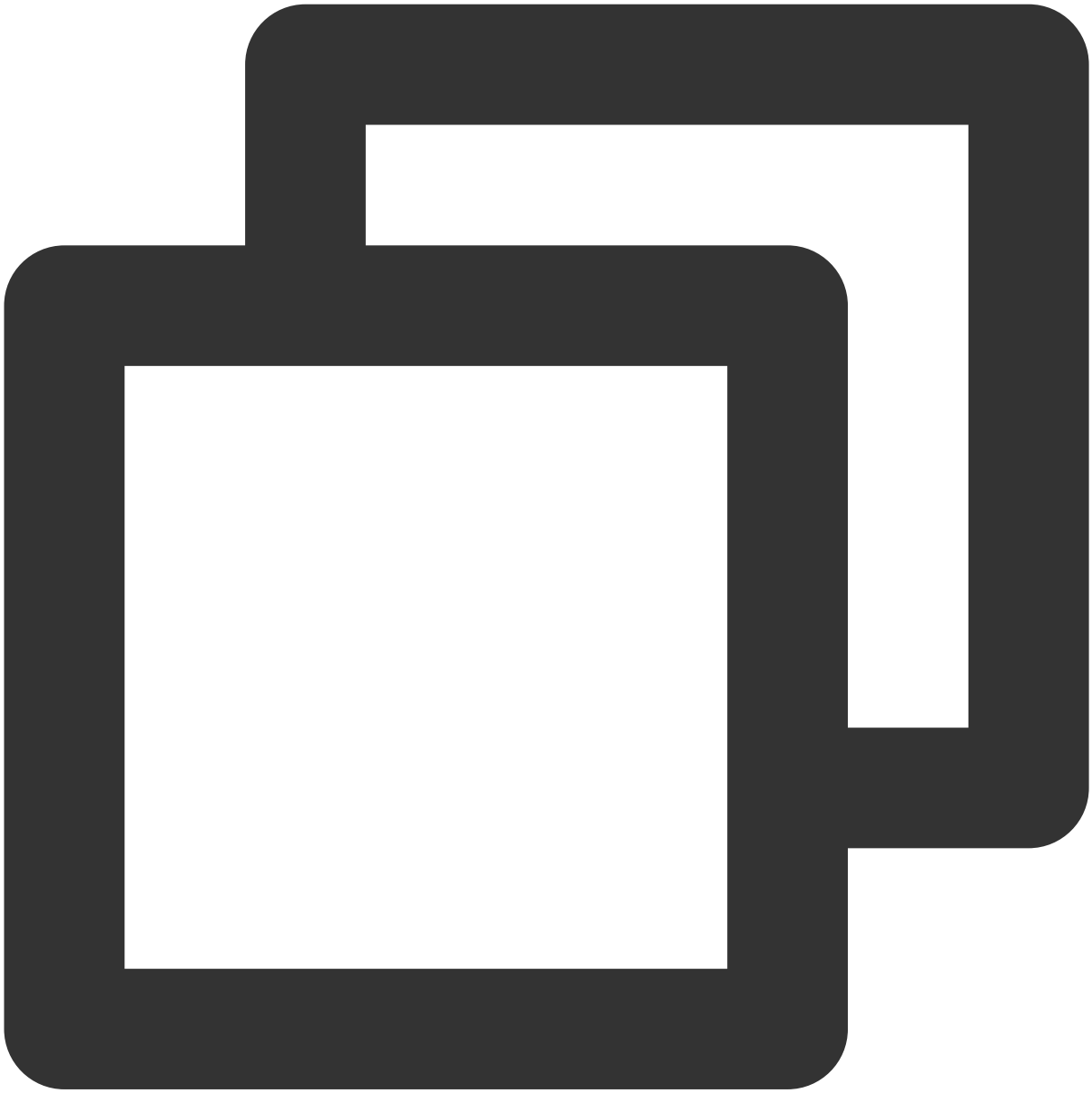
使用 `CosXmlConfig` 与 `QCloudCredentialProvider` 初始化 `CosXmlServer` 服务类。服务类建议在程序中作为**单例**使用。



```
CosXml cosXml = new CosXmlServer(config, cosCredentialProvider);
```

第三步：访问 COS 服务

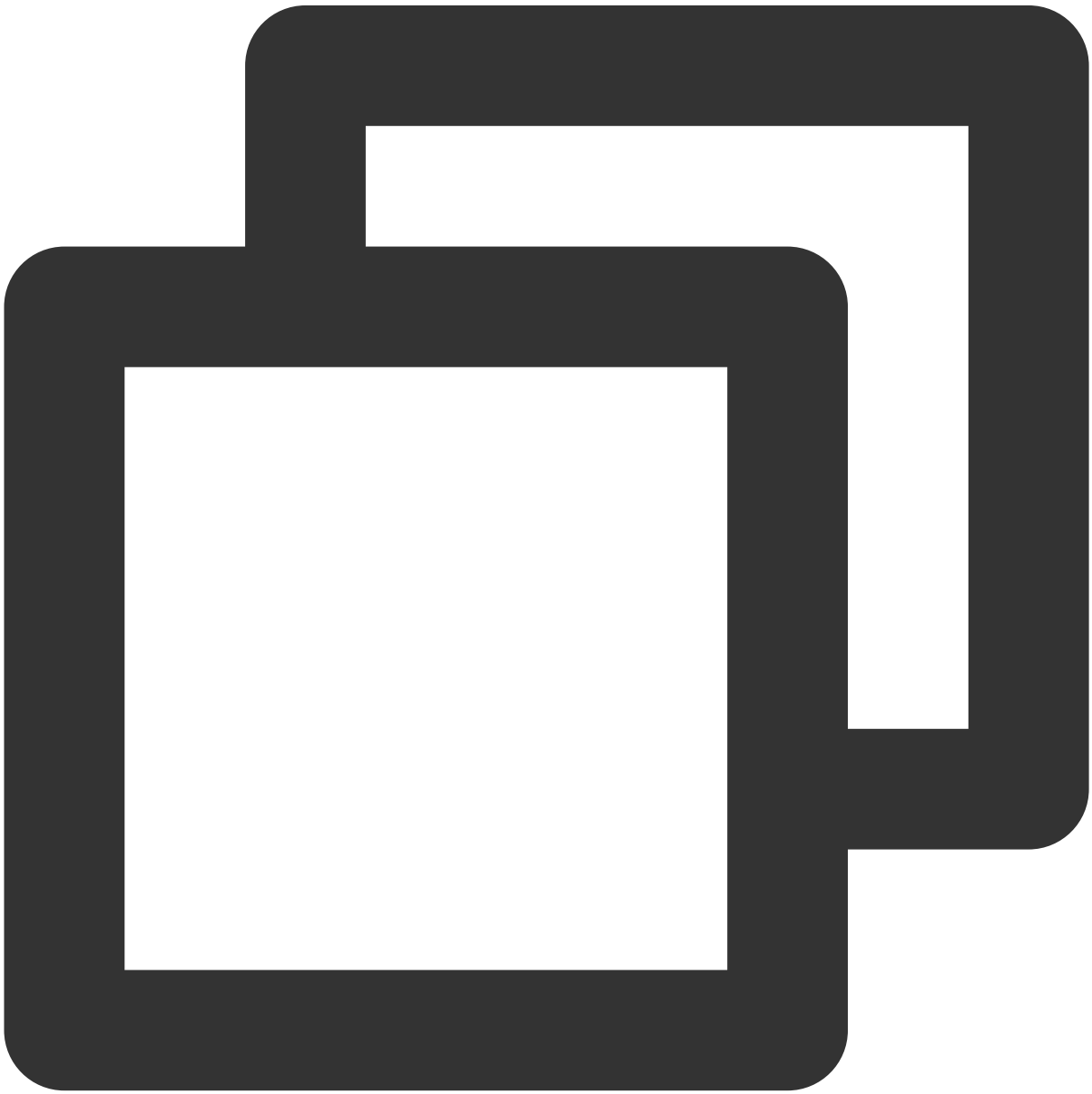
创建存储桶



```
try
{
  string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
  PutBucketRequest request = new PutBucketRequest(bucket);
  //执行请求
  PutBucketResult result = cosXml.PutBucket(request);
  //请求成功
  Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
```

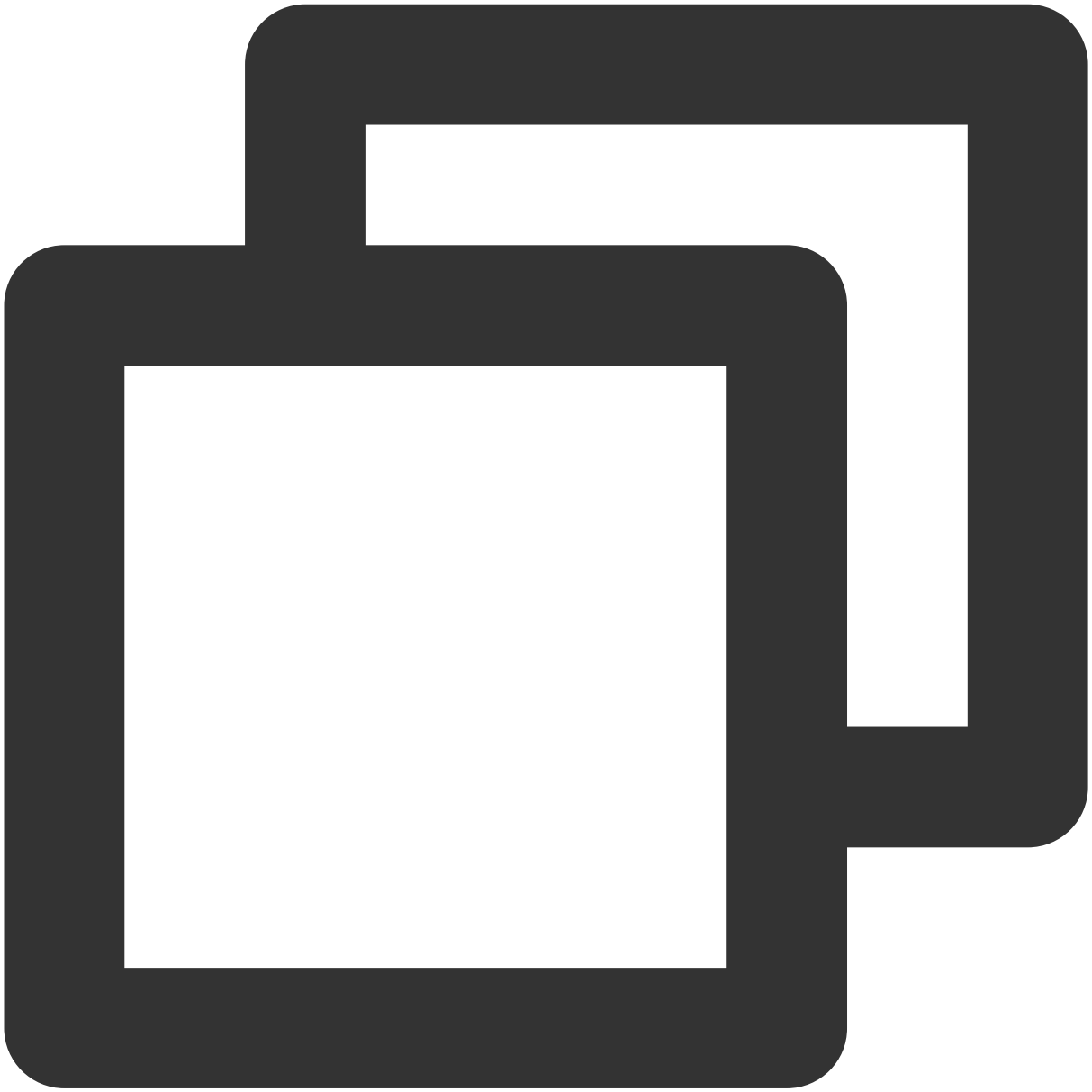
```
    //请求失败
    Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
    //请求失败
    Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

查询存储桶列表



```
try
{
  GetServiceRequest request = new GetServiceRequest();
  //执行请求
  GetServiceResult result = cosXml.GetService(request);
  //得到所有的 buckets
  List<ListAllMyBuckets.Bucket> allBuckets = result.listAllMyBuckets.buckets;
}
catch (COSXML.CosException.CosClientException clientEx)
{
  //请求失败
  Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
  //请求失败
  Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

上传对象



```
// 初始化 TransferConfig
TransferConfig transferConfig = new TransferConfig();

// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXml, transferConfig);

String bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String srcPath = @"temp-source-file";//本地文件绝对路径

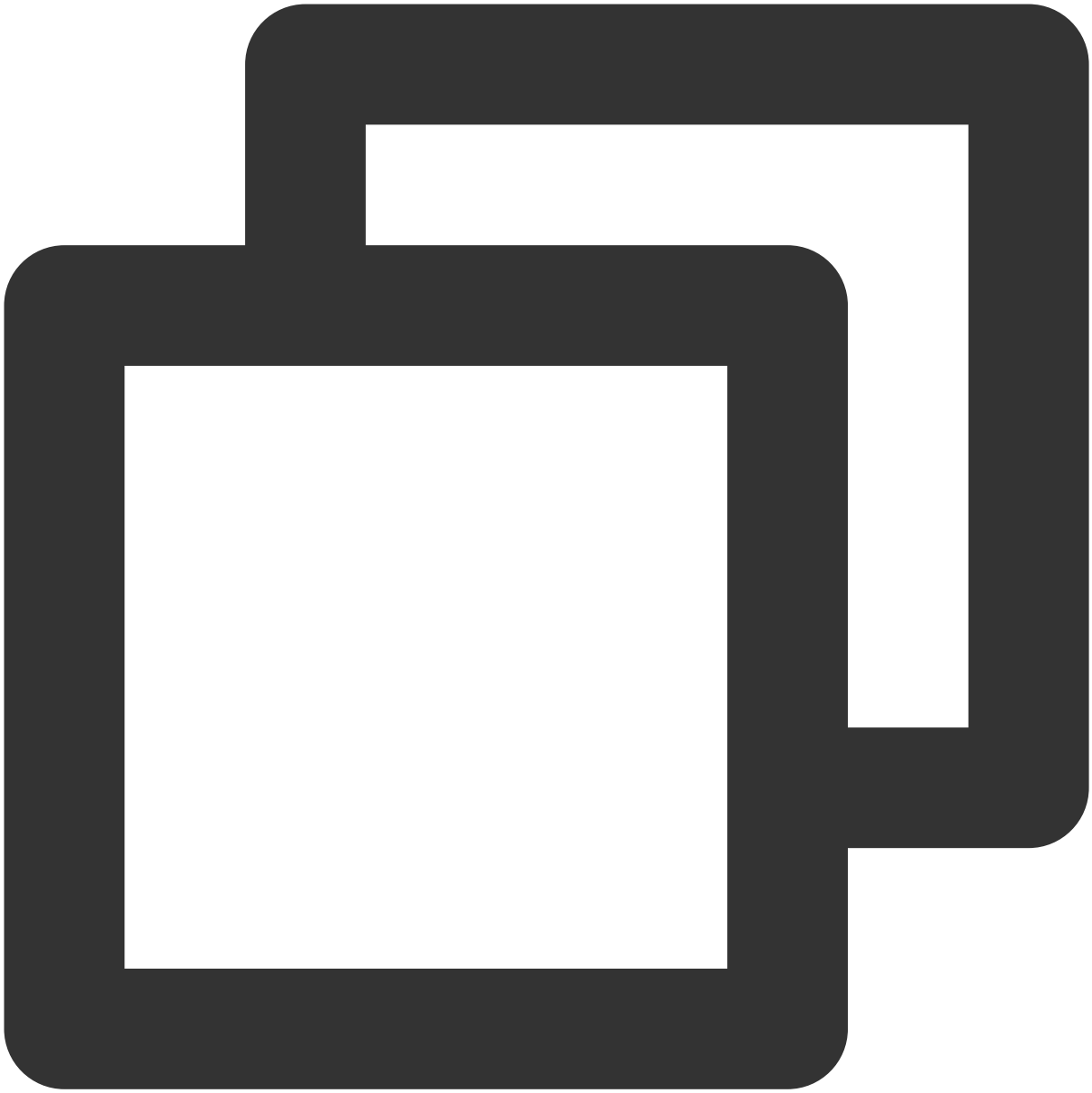
// 上传对象
```

```
COSXMLUploadTask uploadTask = new COSXMLUploadTask(bucket, cosPath);
uploadTask.SetSrcPath(srcPath);

uploadTask.progressCallback = delegate (long completed, long total)
{
    Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / to
};

try {
  COSXML.Transfer.COSXMLUploadTask.UploadTaskResult result = await
    transferManager.UploadAsync(uploadTask);
  Console.WriteLine(result.GetResultInfo());
  string eTag = result.eTag;
} catch (Exception e) {
    Console.WriteLine("CosException: " + e);
}
```

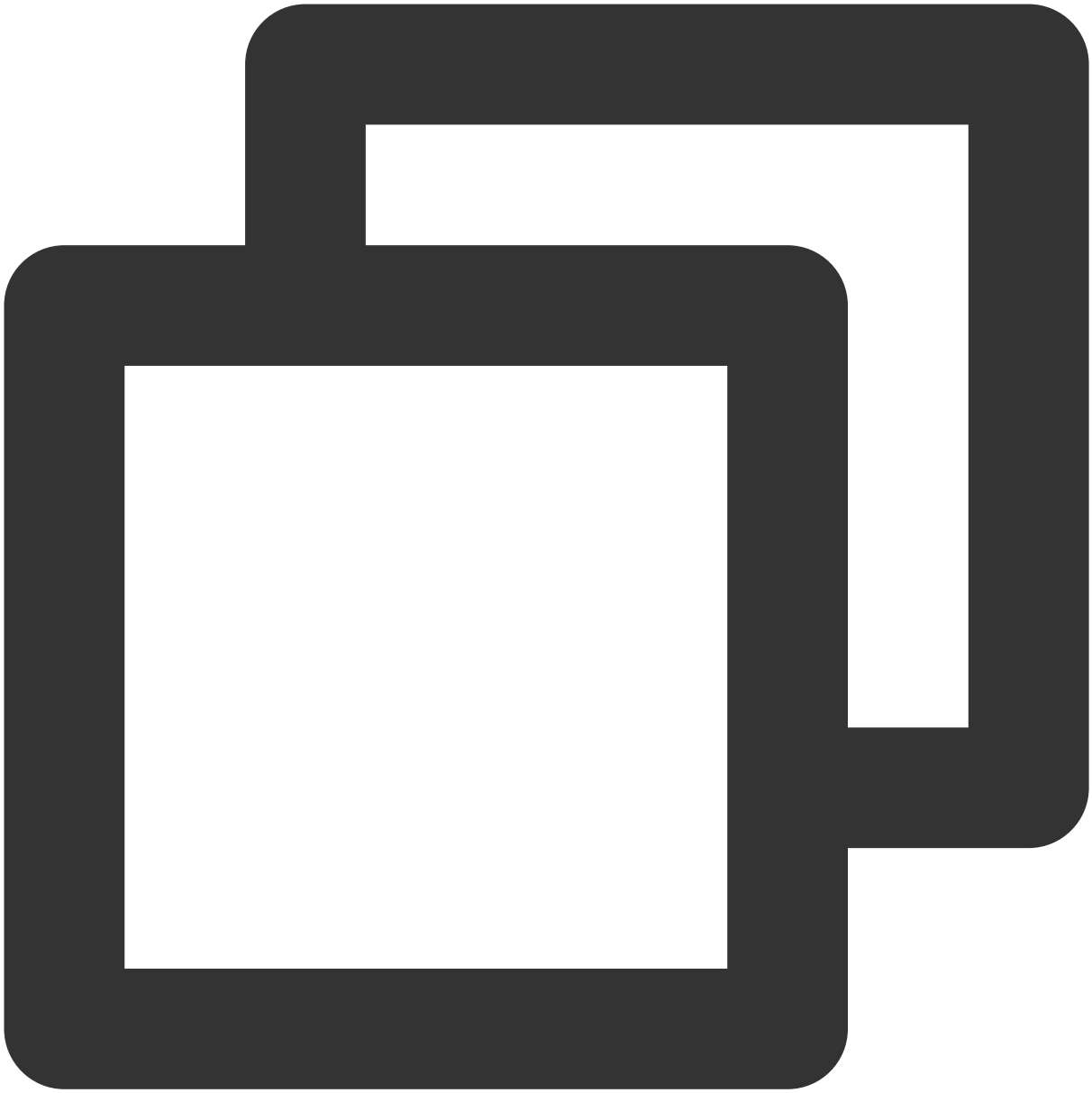
查询对象列表



```
try
{
  string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
  GetBucketRequest request = new GetBucketRequest(bucket);
  //执行请求
  GetBucketResult result = cosXml.GetBucket(request);
  //bucket的相关信息
  ListBucket info = result.listBucket;
  if (info.isTruncated) {
    // 数据被截断，记录下数据下标
    this.nextMarker = info.nextMarker;
```

```
    }
}
catch (COSXML.CosException.CosClientException clientEx)
{
    //请求失败
    Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
    //请求失败
    Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

下载对象



```
// 初始化 TransferConfig
TransferConfig transferConfig = new TransferConfig();

// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXml, transferConfig);

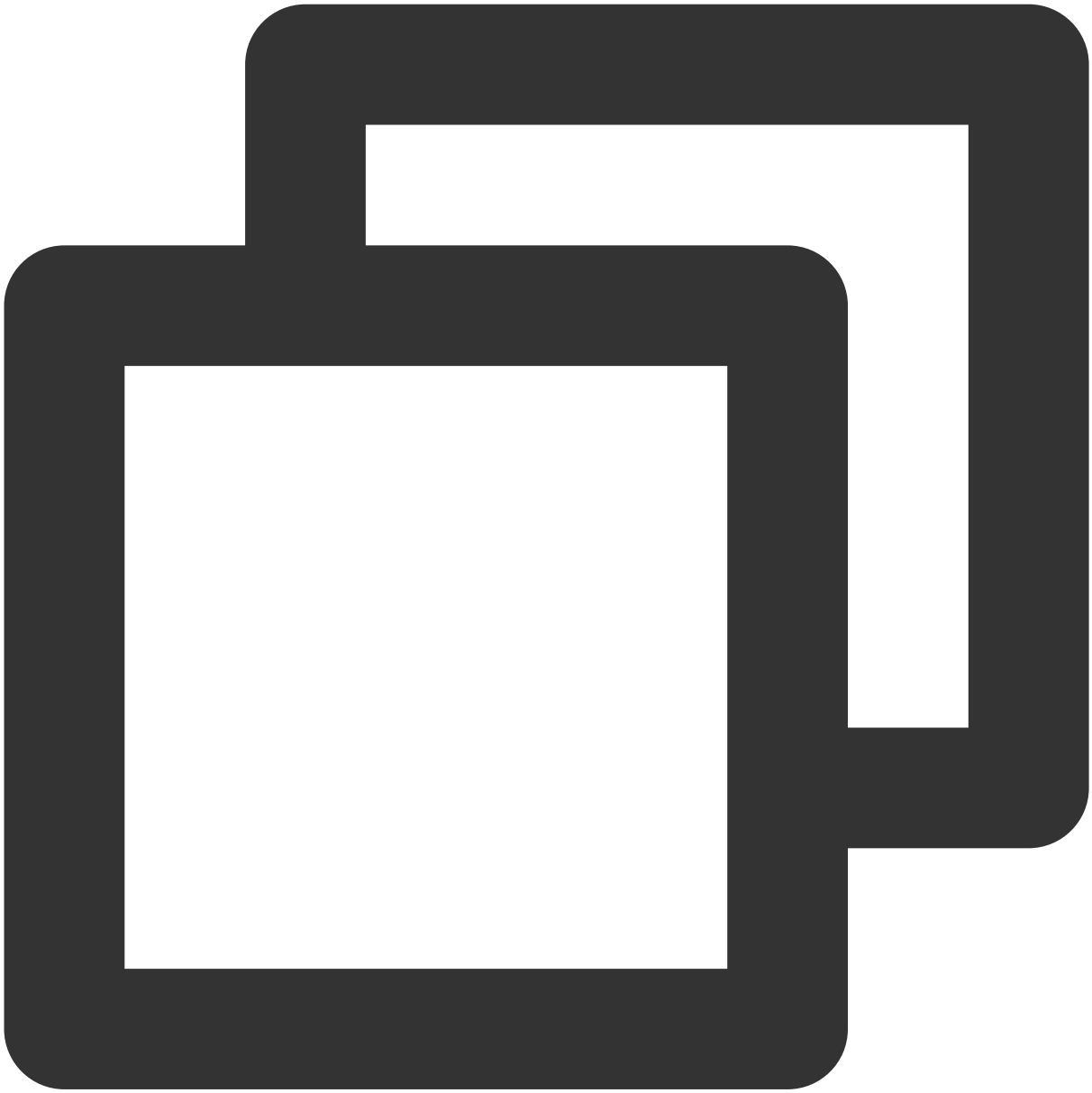
String bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
string localDir = System.IO.Path.GetTempPath();//本地文件夹
string localFileName = "my-local-temp-file"; //指定本地保存的文件名
```

```
// 下载对象
COSXMLDownloadTask downloadTask = new COSXMLDownloadTask(bucket, cosPath,
  localDir, localFileName);

downloadTask.progressCallback = delegate (long completed, long total)
{
    Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / to
};

try {
  COSXML.Transfer.COSXMLDownloadTask.DownloadTaskResult result = await
    transferManager.DownloadAsync(downloadTask);
  Console.WriteLine(result.GetResultInfo());
  string eTag = result.eTag;
} catch (Exception e) {
    Console.WriteLine("CosException: " + e);
}
```

删除对象



```
try
{
  string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
  string key = "exampleobject"; //对象键
  DeleteObjectRequest request = new DeleteObjectRequest(bucket, key);
  //执行请求
  DeleteObjectResult result = cosXml.DeleteObject(request);
  //请求成功
  Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
```

```
{
  //请求失败
  Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
  //请求失败
  Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

.NET（C#）SDK 常见问题

最近更新时间：2023-03-08 17:26:25

.NET SDK 是否支持下载对象到内存？

支持。

.NET SDK 能否直接获取对象文件流？

目前支持下载到本地和下载到内存两种方法。如果您需要获取文件流，可以将文件下载到本地之后打开一个本地的文件流。

版权所有：腾讯云计算（北京）有限责任公司

存储桶操作
创建存储桶

最近更新时间：2023-12-12 16:30:21

简介

本文档提供关于创建存储桶的 API 概览以及 SDK 示例代码。

注意

建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄露目标存储桶或对象之外的资源。

如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

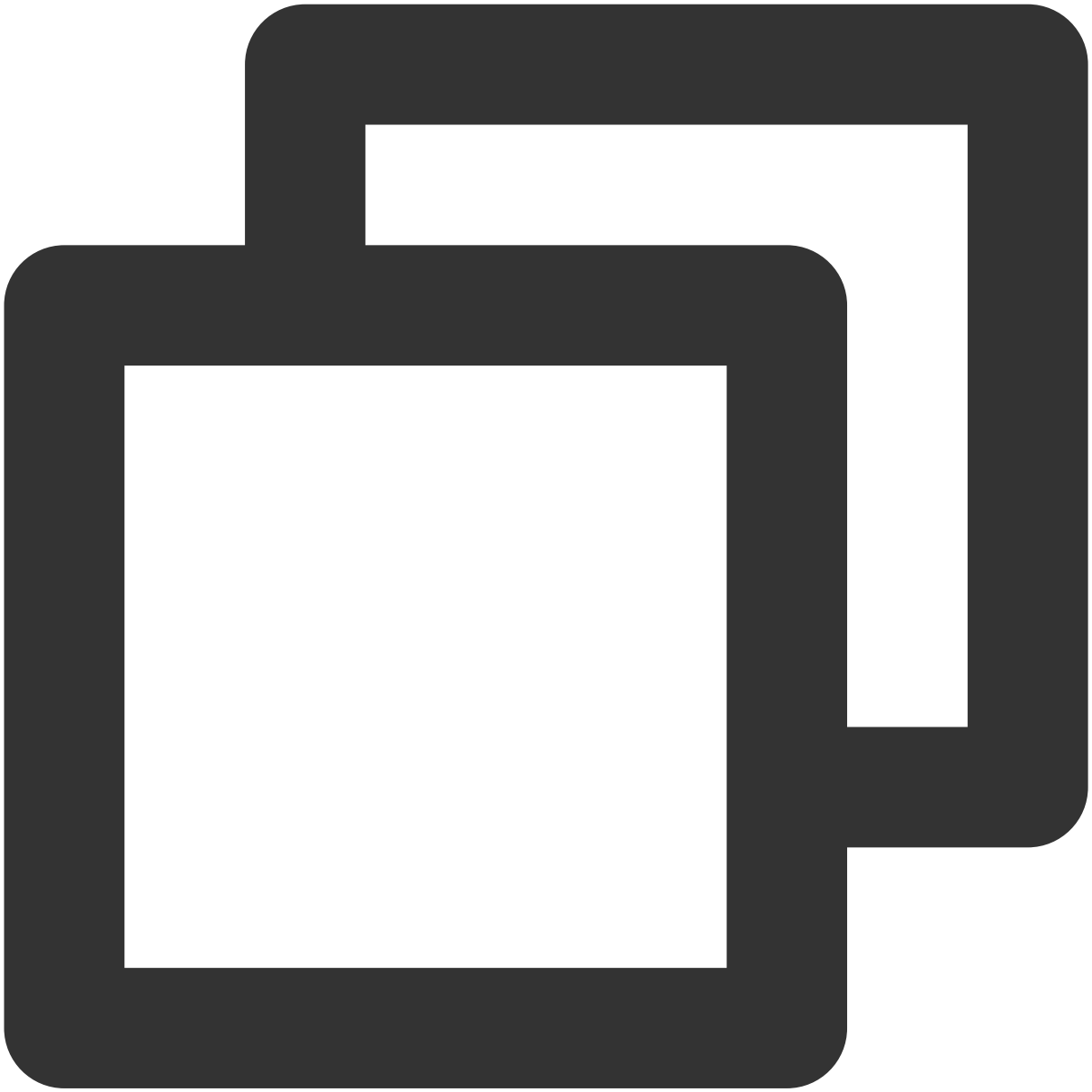
API	操作名	操作描述
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶

创建存储桶

功能说明

在指定账号下创建一个存储桶。同一用户账号下，可以创建多个存储桶，数量上限是200个（不区分地域），存储桶中的对象数量没有限制。创建存储桶是低频操作，一般建议在控制台创建 Bucket，在 SDK 进行 Object 的操作。

示例代码



```
using COSXML.Common;
using COSXML.Model.Bucket;
using COSXML.Auth;
using System;
using COSXML;

namespace COSSnippet
{
    public class PutBucketModel {

      private CosXml cosXml;
```

```
PutBucketModel() {
  CosXmlConfig config = new CosXmlConfig.Builder()
    .SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud
    .Build();

  string secretId = "SECRET_ID";    // 云 API 密钥 SecretId, 获取 API 密钥请参照 h
  string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照
  long durationSecond = 600;          //每次请求签名有效时长, 单位为秒
  QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCreden
    secretKey, durationSecond);

  this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}

/// 创建存储桶
public void PutBucket()
{
  //.cssg-snippet-body-start:[put-bucket]
  try
  {
    // 存储桶名称, 此处填入格式必须为 BucketName-APPID, 其中 APPID 获取参考 https://
    string bucket = "examplebucket-1250000000";
    PutBucketRequest request = new PutBucketRequest(bucket);
    //执行请求
    PutBucketResult result = cosXml.PutBucket(request);
    //请求成功
    Console.WriteLine(result.GetResultInfo());
  }
  catch (COSXML.CosException.CosClientException clientEx)
  {
    //请求失败
    Console.WriteLine("CosClientException: " + clientEx);
  }
  catch (COSXML.CosException.CosServerException serverEx)
  {
    //请求失败
    Console.WriteLine("CosServerException: " + serverEx.GetInfo());
  }

  //.cssg-snippet-body-end
}

// .cssg-methods-pragma

static void Main(string[] args)
{
```

```
            PutBucketModel m = new PutBucketModel();

            /// 创建存储桶
            m.PutBucket();
            // .cssg-methods-pragma
        }
    }
}
```

说明

更多完整示例，请前往 GitHub 查看。

删除存储桶

最近更新时间：2023-12-18 14:47:15

简介

本文档提供关于删除存储桶的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

删除存储桶

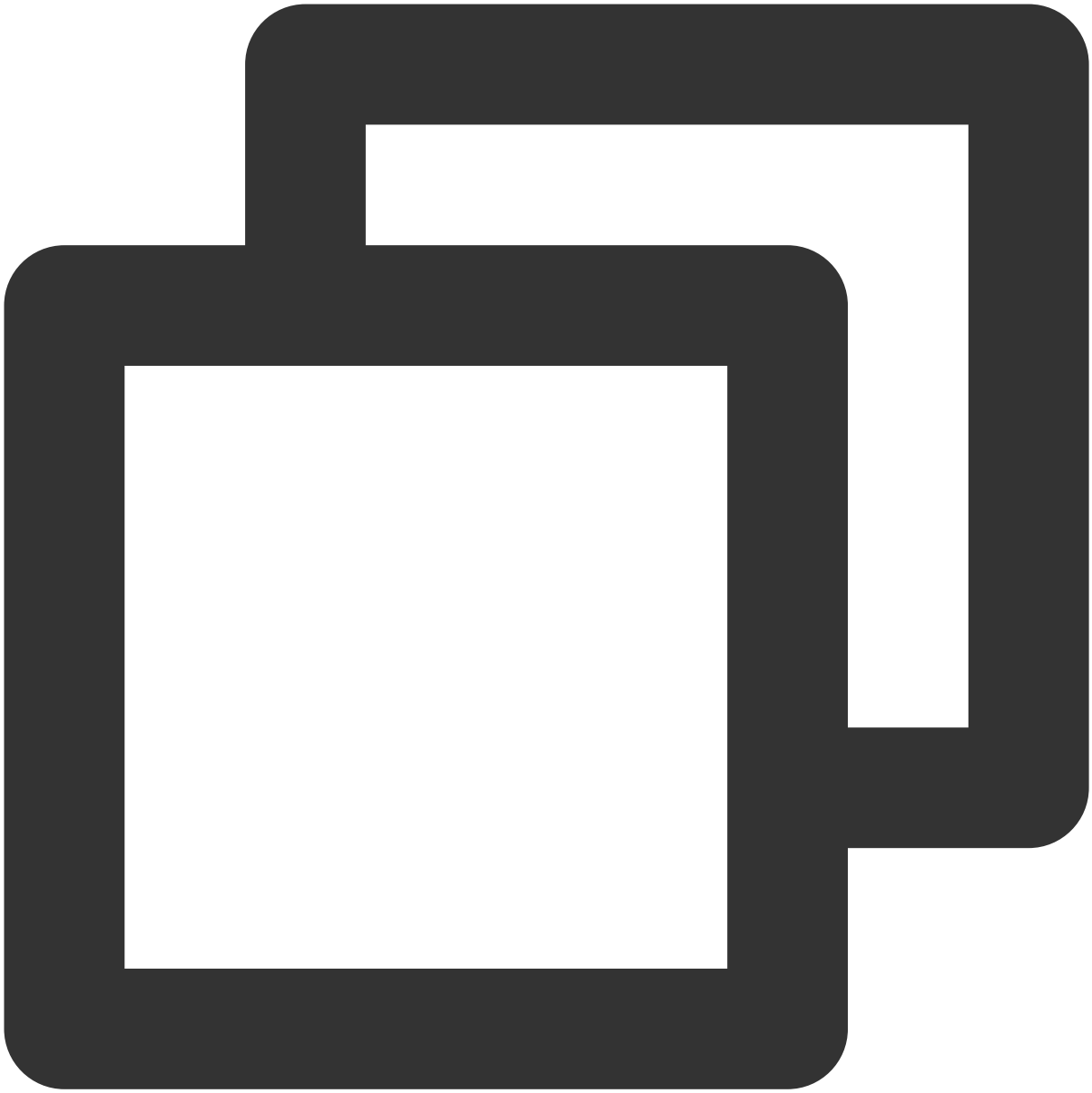
功能说明

删除指定的存储桶（DELETE Bucket）。

注意

删除存储桶前，请确保存储桶内的数据和未完成上传的分块数据已全部清空，否则会无法删除存储桶。

示例代码



```
using COSXML.Model.Bucket;
using COSXML.Auth;
using System;
using COSXML;

namespace COSSnippet
{
    public class DeleteBucketModel {

      private CosXml cosXml;
```

版权所有：腾讯云计算（北京）有限责任公司

```
    DeleteBucketModel() {
      CosXmlConfig config = new CosXmlConfig.Builder()
        .SetRegion("COS_REGION") // 设置默认的区域, COS 地域的简称请参照 https://cloud
        .Build();

      string secretId = "SECRET_ID";    // 云 API 密钥 SecretId, 获取 API 密钥请参照 h
      string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照
      long durationSecond = 600;          //每次请求签名有效时长, 单位为秒
      QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCreden
        secretKey, durationSecond);

      this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
    }

    /// 删除存储桶
    public void DeleteBucket()
    {
      //.cssg-snippet-body-start:[delete-bucket]
      try
      {
        // 存储桶名称, 此处填入格式必须为 BucketName-APPID, 其中 APPID 获取参考 https://
        string bucket = "examplebucket-1250000000";
        DeleteBucketRequest request = new DeleteBucketRequest(bucket);
        //执行请求
        DeleteBucketResult result = cosXml.DeleteBucket(request);
        //请求成功
        Console.WriteLine(result.GetResultInfo());
      }
      catch (COSXML.CosException.CosClientException clientEx)
      {
        //请求失败
        Console.WriteLine("CosClientException: " + clientEx);
      }
      catch (COSXML.CosException.CosServerException serverEx)
      {
        //请求失败
        Console.WriteLine("CosServerException: " + serverEx.GetInfo());
      }

      //.cssg-snippet-body-end
    }
    // .cssg-methods-pragma

    static void Main(string[] args)
    {
      DeleteBucketModel m = new DeleteBucketModel();
```

```
            /// 删除存储桶
            m.DeleteBucket();
            // .cssg-methods-pragma
        }
    }
}
```

说明

更多完整示例，请前往 GitHub 查看。

查询存储桶列表

最近更新时间：2021-11-25 12:52:18

简介

本文档提供关于查询存储桶列表的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Service（List Buckets）	查询存储桶列表	查询指定账号下所有的存储桶列表

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

查询存储桶列表

功能说明

用于查询指定账号下所有存储桶列表。

示例代码

```
using COSXML.Model.Tag;
using COSXML.Model.Service;
using COSXML.Auth;
using System;
using COSXML;
using System.Collections.Generic;
namespace COSSnippet
{
public class GetServiceModel {
private CosXml cosXml;
GetServiceModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencen
t.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://
```

```
console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 http
s://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 获取存储桶列表
public void GetService()
{
//.cssg-snippet-body-start:[get-service]
try
{
GetServiceRequest request = new GetServiceRequest();
//执行请求
GetServiceResult result = cosXml.GetService(request);
//得到所有的 buckets
List<ListAllMyBuckets.Bucket> allBuckets = result.listAllMyBuckets.buckets;
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

//.cssg-snippet-body-end
}
/// 获取地域的存储桶列表
public void GetRegionalService()
{
//.cssg-snippet-body-start:[get-regional-service]
try
{
GetServiceRequest request = new GetServiceRequest();
string region = "ap-guangzhou";
request.host = $"cos.{region}.myqcloud.com";
//执行请求
GetServiceResult result = cosXml.GetService(request);
//得到所有的 buckets
```

```
List<ListAllMyBuckets.Bucket> allBuckets = result.listAllMyBuckets.buckets;
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
//.cssg-snippet-body-end
}
// .cssg-methods-pragma
static void Main(string[] args)
{
GetServiceModel m = new GetServiceModel();
/// 获取存储桶列表
m.GetService();
/// 获取地域的存储桶列表
m.GetRegionalService();
// .cssg-methods-pragma
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

检索存储桶

最近更新时间：2021-11-25 12:52:19

简介

本文档提供关于检索存储桶的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

检索存储桶及其权限

功能说明

HEAD Bucket 请求可以确认该存储桶是否存在，是否有权限访问。有以下几种情况：

- 存储桶存在且有读取权限，返回 HTTP 状态码为200。
- 无存储桶读取权限，返回 HTTP 状态码为403。
- 存储桶不存在，返回 HTTP 状态码为404。

示例代码

```
using COSXML.Model.Bucket;
using COSXML.Auth;
using System;
using COSXML;
namespace COSSnippet
{
public class HeadBucketModel {
private CosXml cosXml;
HeadBucketModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencen
t.com/document/product/436/6224
```

```
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://
console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 http
s://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 获取存储桶信息
public void HeadBucket()
{
//.cssg-snippet-body-start:[head-bucket]
try
{
// 存储桶名称, 此处填入格式必须为 BucketName-APPID, 其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
HeadBucketRequest request = new HeadBucketRequest(bucket);
//执行请求
HeadBucketResult result = cosXml.HeadBucket(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

//.cssg-snippet-body-end
}
// .cssg-methods-pragma
static void Main(string[] args)
{
HeadBucketModel m = new HeadBucketModel();
/// 获取存储桶信息
m.HeadBucket();
```

```
// .cssg-methods-pragma
}
}
}
```

说明：
更多完整示例，请前往 GitHub 查看。

判断存储桶是否存在

最近更新时间：2022-01-17 12:50:00

简介

本文档提供关于检查存储桶是否存在的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

检查存储桶是否存在

功能说明

DoesBucketExist 请求可以确认该存储桶是否存在。

> 说明：
>
> - DoesBucketExist 是 .NET SDK 提供的一种简化的 HEAD Bucket 接口，仅用于检查存储桶是否存在，并返回布尔类型。
> - 如需进一步获取存储桶详情，请使用 检索存储桶 接口。

示例代码

```
using COSXML.Model.Bucket;
using COSXML.Auth;
using System;
using COSXML;
namespace COSSnippet
{
public class DoesBucketExistModel {
private CosXml cosXml;
```

```
DoesBucketExistModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencen
t.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://
console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 http
s://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 检查存储桶是否存在
public void DoesBucketExist()
{
try
{
// 存储桶名称, 此处填入格式必须为 BucketName-APPID, 其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
DoesBucketExistRequest request = new DoesBucketExistRequest(bucket);
//执行请求
bool existState = cosXml.DoesBucketExist(request);
//请求成功
Console.WriteLine("bucket exist state is: " + existState);
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

}
static void Main(string[] args)
{
DoesBucketExistModel m = new DoesBucketExistModel();
/// 检查存储桶是否存在
```

```
m.DoesBucketExist();
}
}
}
```

说明：
更多完整示例，请前往 GitHub 查看。

对象操作
上传对象

最近更新时间：2022-01-23 14:45:54

简介

本文档提供关于上传对象的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object	简单上传对象	上传一个对象至存储桶
POST Object	表单上传对象	使用表单请求上传对象
APPEND Object	追加上传对象	使用分块追加的方式上传对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传操作
Upload Part	上传分块	分块上传对象
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

高级接口（推荐）

版权所有：腾讯云计算（北京）有限责任公司

上传对象

功能说明

高级接口封装了简单上传、分块上传接口，根据文件大小智能的选择上传方式，同时支持续传功能。

说明：

- 文件大小小于分块阈值时选择简单上传，超过阈值时使用分块上传，阈值支持用户自行配置，默认为5MB。
- 分块大小支持用户自行配置，默认为1MB。
- 针对.NET Framework 4.0 及以下的用户，高级接口暂不可用，详见 向下兼容指南。

示例代码一: 高级接口上传本地文件

```
using COSXML.Model.Object;
using COSXML.Auth;
using COSXML.Transfer;
using System;
using COSXML;
using System.Threading.Tasks;
namespace COSSnippet
{
public class TransferUploadObjectModel {
private CosXml cosXml;
TransferUploadObjectModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://intl.cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 高级接口上传文件
public async Task TransferUploadFile()
```

```
{
// 初始化 TransferConfig
TransferConfig transferConfig = new TransferConfig();
// 手动设置开始分块上传的大小阈值为10MB，默认值为5MB
transferConfig.DivisionForUpload = 10 * 1024 * 1024;
// 手动设置分块上传中每个分块的大小为2MB，默认值为1MB
transferConfig.SliceSizeForUpload = 2 * 1024 * 1024;

// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXml, transferConfig);
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String srcPath = @"temp-source-file";//本地文件绝对路径

// 上传对象
COSXMLUploadTask uploadTask = new COSXMLUploadTask(bucket, cosPath);
uploadTask.SetSrcPath(srcPath);

uploadTask.progressCallback = delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
};
try {
COSXML.Transfer.COSXMLUploadTask.UploadTaskResult result = await
transferManager.UploadAsync(uploadTask);
Console.WriteLine(result.GetResultInfo());
string eTag = result.eTag;
} catch (Exception e) {
Console.WriteLine("CosException: " + e);
}
}
static void Main(string[] args)
{
TransferUploadObjectModel m = new TransferUploadObjectModel();
/// 高级接口上传对象
m.TransferUploadFile().Wait();
// .cssg-methods-pragma
}
}
}
```

说明：

- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法请参见 生成预签名链接。但注意，如果您的文件是私有读权限，那么下载链接只有一定的有效期。

示例代码二: 上传二进制数据

```
using COSXML.Model.Object;
using COSXML.Auth;
using COSXML.Transfer;
using System;
using COSXML;
namespace COSSnippet
{
public class TransferUploadObjectModel {
private CosXml cosXml;
TransferUploadObjectModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://intl.cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 上传二进制数据
public void TransferUploadBytes()
{
try
{
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string cosPath = "exampleObject"; // 对象键
byte[] data = new byte[1024]; // 二进制数据
PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, data);
// 发起上传
PutObjectResult result = cosXml.PutObject(putObjectRequest);
```

```
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
}
static void Main(string[] args)
{
TransferUploadObjectModel m = new TransferUploadObjectModel();
/// 高级接口上传二进制数据
m.TransferUploadBytes();
}
}
}
```

说明：

- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法请参见 生成预签名链接。但注意，如果您的文件是私有读权限，那么下载链接只有一定的有效期。

示例代码三: 从文件流上传

```
using COSXML.Model.Object;
using COSXML.Utils;
using COSXML.Auth;
using System;
using COSXML;
using System.IO;
namespace COSSnippet
{
public class PutObjectModel {
private CosXml cosXml;
PutObjectModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://intl.cloud.te
```

```
ncent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://
console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 http
s://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}

/// 文件流上传, 从 5.4.24 版本开始支持
public void PutObjectStream()
{
try
{
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string srcPath = @"temp-source-file";//本地文件绝对路径
// 打开只读的文件流对象
FileStream fileStream = new FileStream(srcPath, FileMode.Open, FileAccess.Read);
// 组装上传请求, 其中 offset sendLength 为可选参数
long offset = 0L;
long sendLength = fileStream.Length;
PutObjectRequest request = new PutObjectRequest(bucket, key, fileStream, offset,
sendLength);
//设置进度回调
request.SetCosProgressCallback(delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / tota
l));
});
//执行请求
PutObjectResult result = cosXml.PutObject(request);
//关闭文件流
fileStream.Close();
//对象的 eTag
string eTag = result.eTag;
//对象的 crc64ecma 校验值
string crc64ecma = result.crc64ecma;
//打印请求结果
```

```
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
}
static void Main(string[] args)
{
PutObjectModel m = new PutObjectModel();
/// 从文件流上传对象
m.PutObjectStream();
}
}
}
```

说明：

- 文件流上传特性从 5.4.24 版本开始支持，下载新版 SDK 前往 Releases 或参见 快速入门。
- 查看版本 Changelog 请前往 GitHub 查看。
- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法请参见 生成预签名链接。但注意，如果您的文件是私有读权限，那么下载链接只有一定的有效期。

示例代码四: 上传暂停、继续与取消

对于上传任务，可以通过以下方式暂停：

```
uploadTask.Pause();
```

暂停之后，可以通过以下方式续传：

```
uploadTask.Resume();
```

也通过以下方式取消上传：

```
uploadTask.Cancel();
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码五: 批量上传

```
TransferConfig transferConfig = new TransferConfig();
// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXml, transferConfig);
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
for (int i = 0; i < 5; i++) {
// 上传对象
string cosPath = "exampleobject" + i; //对象在存储桶中的位置标识符，即称对象键
string srcPath = @"temp-source-file";//本地文件绝对路径
COSXMLUploadTask uploadTask = new COSXMLUploadTask(bucket, cosPath);
uploadTask.SetSrcPath(srcPath);
await transferManager.UploadAsync(uploadTask);
}
```

示例代码六：创建目录

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string cosPath = "dir/"; // 对象键
PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, new byte[0]);

cosXml.PutObject(putObjectRequest);
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
```

```
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法请参见 生成预签名链接。但注意，如果您的文件是私有读权限，那么下载链接只有一定的有效期。

简单操作

简单上传对象

功能说明

PUT Object 接口可以上传一个对象至指定存储桶中，该操作需要请求者对存储桶有 WRITE 权限。最大支持上传不超过5GB的对象，5GB以上对象请使用 分块上传 或 高级接口 上传。

注意：

- Key（文件名）不能以 `/` 结尾，否则会被识别为文件夹。
- 每个主账号（即同一个 APPID），存储桶的 ACL 规则数量最多为1000条，对象 ACL 规则数量不限制。如果您不需要进行对象 ACL 控制，请在上传时不要设置，默认继承存储桶权限。
- 由于 SDK 底层依赖语言组件的限制，普通上传接口在某些环境下，单次上传超过2GB的文件可能遇到预期外的异常，建议使用 高级接口 或 分块上传 以规避潜在问题。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string srcPath = @"temp-source-file";//本地文件绝对路径
PutObjectRequest request = new PutObjectRequest(bucket, key, srcPath);
//设置进度回调
request.SetCosProgressCallback(delegate (long completed, long total)
{
```

```
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / tota
l));
});
//执行请求
PutObjectResult result = cosXml.PutObject(request);
//对象的 eTag
string eTag = result.eTag;
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法请参见 生成预签名链接。但注意，如果您的文件是私有读权限，那么下载链接只有一定的有效期。

表单上传对象

功能说明

使用表单请求上传对象。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string srcPath = @"temp-source-file";//本地文件绝对路径
PostObjectRequest request = new PostObjectRequest(bucket, key, srcPath);
//设置进度回调
request.SetCosProgressCallback(delegate (long completed, long total)
{
```

```
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / tota
l));
});
//执行请求
PostObjectResult result = cosXml.PostObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

追加上传对象

功能说明

使用分块追加的方式上传对象。

说明：
此功能从 5.4.22 版本开始支持，查看版本 Changelog 请前往 GitHub。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string srcPath = @"temp-source-file";//本地文件绝对路径
//首次append上传，追加位置传0，创建一个appendable对象
```

```
long next_append_position = 0;
AppendObjectRequest request = new AppendObjectRequest(bucket, key, srcPath, next_
append_position);
//设置进度回调
request.SetCosProgressCallback(delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / tota
l));
});
AppendObjectResult result = cosXml.AppendObject(request);
//获取下次追加位置
next_append_position = result.nextAppendPosition;
Console.WriteLine(result.GetResultInfo());
//执行追加,传入上次获取的对象末尾
request = new AppendObjectRequest(bucket, key, srcPath, next_append_position);
request.SetCosProgressCallback(delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / tota
l));
});
result = cosXml.AppendObject(request);
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

分块操作

这里说明下分块上传的流程。

分块上传的流程

1. 初始化分块上传（Initiate Multipart Upload），得到 UploadId。
2. 使用 UploadId 上传分块（Upload Part）。
3. 完成分块上传（Complete Multipart Upload）。

分块继续上传的流程

1. 如果没有记录 UploadId，查询分块上传任务（List Multipart Uploads），得到对应文件的 UploadId。
2. 使用 UploadId 列出已上传的分块（List Parts）。
3. 使用 UploadId 上传剩余的分块（Upload Part）。
4. 完成分块上传（Complete Multipart Upload）。

终止分块上传的流程

1. 如果没有记录 UploadId，查询分块上传任务（List Multipart Uploads），得到对应文件的 UploadId。
2. 终止分块上传并删除已上传分块（Abort Multipart Upload）。

查询分块上传

功能说明

查询指定存储桶中正在进行的分块上传（List Multipart Uploads）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
ListMultiUploadsRequest request = new ListMultiUploadsRequest(bucket);
//执行请求
ListMultiUploadsResult result = cosXml.ListMultiUploads(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

初始化分块上传

功能说明

初始化 Multipart Upload 上传操作，获取对应的 uploadId（Initiate Multipart Upload）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
InitMultipartUploadRequest request = new InitMultipartUploadRequest(bucket, key);
//执行请求
InitMultipartUploadResult result = cosXml.InitMultipartUpload(request);
//请求成功
this.uploadId = result.initMultipartUpload.uploadId; //用于后续分块上传的 uploadId
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

上传分块

功能说明

分块上传对象（Upload Part）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string uploadId = "exampleUploadId"; //初始化分块上传返回的uploadId
int partNumber = 1; //分块编号，必须从1开始递增
string srcPath = @"temp-source-file";//本地文件绝对路径
UploadPartRequest request = new UploadPartRequest(bucket, key, partNumber, uploadId, srcPath, 0, -1);
//设置进度回调
request.SetCosProgressCallback(delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
UploadPartResult result = cosXml.UploadPart(request);
//请求成功
//获取返回分块的eTag,用于后续CompleteMultiUploads
this.eTag = result.eTag;
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

查询已上传的分块

功能说明

查询特定分块上传操作中的已上传的块（List Parts）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string uploadId = "exampleUploadId"; //初始化分块上传返回的uploadId
ListPartsRequest request = new ListPartsRequest(bucket, key, uploadId);
//执行请求
ListPartsResult result = cosXml.ListParts(request);
//请求成功
//列举已上传的分块
List<COSXML.Model.Tag.ListParts.Part> alreadyUploadParts = result.listParts.parts;
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

完成分块上传

功能说明

完成整个文件的分块上传（Complete Multipart Upload）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string uploadId = "exampleUploadId"; //初始化分块上传返回的uploadId
CompleteMultipartUploadRequest request = new CompleteMultipartUploadRequest(bucket,
key, uploadId);
//设置已上传的parts,必须有序，按照partNumber递增
request.SetPartNumberAndETag(1, this.eTag);
//执行请求
CompleteMultipartUploadResult result = cosXml.CompleteMultiUpload(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块（Abort Multipart Upload）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
```

```
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string uploadId = "exampleUploadId"; //初始化分块上传返回的uploadId
AbortMultipartUploadRequest request = new AbortMultipartUploadRequest(bucket, ke
y, uploadId);
//执行请求
AbortMultipartUploadResult result = cosXml.AbortMultiUpload(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

下载对象

最近更新时间：2023-01-12 14:55:02

简介

本文档提供关于下载对象的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

高级接口（推荐）

下载对象（断点续传）

高级接口支持暂停、恢复以及取消下载请求，同时支持断点下载功能。

> 说明：
>
> - .NET Framework 4.0 及以下的用户，高级接口暂不可用，详见 向下兼容指南。
> - 高级下载接口从 5.4.26 版本开始支持并发分块下载，下载新版 SDK 前往 Releases 或参见 快速入门。

示例代码一: 下载对象

```
using COSXML.Model.Object;
using COSXML.Auth;
using COSXML.Transfer;
using System;
using COSXML;
using System.Threading.Tasks;
namespace COSSnippet
{
public class TransferDownloadObjectModel {
```

```
private CosXml cosXml;
TransferDownloadObjectModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 高级接口下载对象
public async Task TransferDownloadObject()
{
// 初始化 TransferConfig
TransferConfig transferConfig = new TransferConfig();
// 手动设置高级下载接口的分块阈值为 20MB(默认为20MB), 从5.4.26版本开始支持！
//transferConfig.DivisionForDownload = 20 * 1024 * 1024;
// 手动设置高级下载接口的分块大小为 10MB(默认为5MB),不建议此处设置过小的分块值,可能导致频繁重试或下载速度不合预期
//transferConfig.SliceSizeForDownload = 10 * 1024 * 1024;

// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXml, transferConfig);

String bucket = "examplebucket-1250000000"; //存储桶, 格式：BucketName-APPID
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符, 即称对象键
string localDir = System.IO.Path.GetTempPath();//本地文件夹
string localFileName = "my-local-temp-file"; //指定本地保存的文件名

// 下载对象
COSXMLDownloadTask downloadTask = new COSXMLDownloadTask(bucket, cosPath,
localDir, localFileName);
// 手动设置高级下载接口的并发数 (默认为5), 从5.4.26版本开始支持！
//downloadTask.SetMaxTasks(10);
downloadTask.progressCallback = delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
};
```

```
try {
COSXML.Transfer.COSXMLDownloadTask.DownloadTaskResult result = await
transferManager.DownloadAsync(downloadTask);
Console.WriteLine(result.GetResultInfo());
string eTag = result.eTag;
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
}
static void Main(string[] args)
{
TransferDownloadObjectModel m = new TransferDownloadObjectModel();
/// 高级接口下载对象
m.TransferDownloadObject().Wait();
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

示例代码二: 设置下载支持断点续传

```
COSXMLDownloadTask downloadTask = new COSXMLDownloadTask(request);
//开启断点续传，当本地存在未下载完成文件时，追加下载到文件末尾
//本地文件已存在部分不符合本次下载的内容，可能导致下载失败，请删除文件重试
downloadTask.SetResumableDownload(true);
try {
COSXML.Transfer.COSXMLDownloadTask.DownloadTaskResult result = await
transferManager.DownloadAsync(downloadTask);
Console.WriteLine(result.GetResultInfo());
string eTag = result.eTag;
} catch (Exception e) {
Console.WriteLine("CosException: " + e);
}
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码三: 批量下载

```
TransferConfig transferConfig = new TransferConfig();
// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXml, transferConfig);
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string localDir = System.IO.Path.GetTempPath();//本地文件夹
for (int i = 0; i < 5; i++) {
// 下载对象
string cosPath = "exampleobject" + i; //对象在存储桶中的位置标识符，即称对象键
string localFileName = "my-local-temp-file"; //指定本地保存的文件名
COSXMLDownloadTask downloadTask = new COSXMLDownloadTask(bucket, cosPath,
localDir, localFileName);
await transferManager.DownloadAsync(downloadTask);
}
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码四：单链接限速下载

```
TransferConfig transferConfig = new TransferConfig();
// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXml, transferConfig);
String bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
string localDir = System.IO.Path.GetTempPath();//本地文件夹
string localFileName = "my-local-temp-file"; //指定本地保存的文件名

GetObjectRequest request = new GetObjectRequest(bucket,
cosPath, localDir, localFileName);
request.LimitTraffic(8 * 1000 * 1024); // 限制为1MB/s
COSXMLDownloadTask downloadTask = new COSXMLDownloadTask(request);
await transferManager.DownloadAsync(downloadTask);
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码五：文件夹下载

```
using COSXML.Common;
using COSXML.CosException;
using COSXML.Model;
using COSXML.Model.Object;
using COSXML.Model.Tag;
using COSXML.Model.Bucket;
using COSXML.Model.Service;
using COSXML.Utils;
using COSXML.Auth;
using COSXML.Transfer;
using System;
using COSXML;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
namespace COSSnippet
{
public class TransferDownloadObjectModel {
private CosXml cosXml;
private string nextMarker;
TransferDownloadObjectModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);
```

```
this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
public void GetObjectsFromFolder()
{
// 注意：COS中实际不存在文件夹下载的接口
// 需要通过组合“指定前缀列出”和“遍历列出的对象key做下载”两种操作，实现类似文件夹下载的操作
// 下面的操作，把对象列出到列表里，然后异步下载列表中的对象
String nextMarker = null;

List<string> downloadList = new List<string>();
string bucket = "examplebucket-1250000000";
string prefix = "folder1/"; //指定前缀
// 循环请求直到没有下一页数据
do
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
GetBucketRequest listRequest = new GetBucketRequest(bucket);
//获取 folder1/ 下的所有对象以及子目录
listRequest.SetPrefix(prefix);
listRequest.SetMarker(nextMarker);
//执行列出对象请求
GetBucketResult listResult = cosXml.GetBucket(listRequest);
ListBucket info = listResult.listBucket;
// 对象列表
List<ListBucket.Contents> objects = info.contentsList;
// 下一页的下标
nextMarker = info.nextMarker;

//对象列表
foreach (var content in objects)
{
downloadList.Add(content.key);
Console.WriteLine("adding key:" + content.key);
}
} while (nextMarker != null);
Console.WriteLine("download list construct done, " + downloadList.Count + " objec
ts added");
TransferConfig transferConfig = new TransferConfig();
TransferManager transferManager = new TransferManager(cosXml, transferConfig);
string localDir = System.IO.Path.GetTempPath();//本地文件夹

List<Task> taskList = new List<Task>();
for (int i = 0; i < downloadList.Count; i++)
{
// 遍历待下载列表，下载内容写到 filename_i 文件中
COSXMLDownloadTask downloadTask = new COSXMLDownloadTask(bucket, downloadList[i],
```

```
localDir, "filename_" + i.ToString());
// 异步下载，加入task队列
Task task = transferManager.DownloadAsync(downloadTask);
taskList.Add(task);
}
Console.WriteLine("download tasks submitted, total " + taskList.Count + " tasks a
dded");

//等待TaskList中所有Task结束
foreach(Task task in taskList)
{
task.Wait();
Console.WriteLine("download completed");
}
}
static void Main(string[] args)
{
TransferDownloadObjectModel m = new TransferDownloadObjectModel();
/// 批量下载文件夹中文件
m.GetObjectsFromFolder();
}
}
}
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码六: 下载暂停、继续与取消

对于下载任务，可以通过以下方式暂停：

```
downloadTask.Pause();
```

暂停之后，可以通过以下方式续传：

```
downloadTask.Resume();
```

也通过以下方式取消上传：

```
downloadTask.Cancel();
```

说明：
更多完整示例，请前往 GitHub 查看。

简单操作

下载对象

功能说明

下载一个 Object（文件/对象）至本地（GET Object）。

示例代码一：单对象简单下载

```
using COSXML.Model.Object;
using COSXML.Auth;
using System;
using COSXML;
namespace COSSnippet
{
public class GetObjectModel {
private CosXml cosXml;
GetObjectModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencen
t.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://
console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 http
s://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 下载对象
public void GetObject()
{
//.cssg-snippet-body-start:[get-object]
try
{
```

```
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string localDir = System.IO.Path.GetTempPath();//本地文件夹
string localFileName = "my-local-temp-file"; //指定本地保存的文件名
GetObjectRequest request = new GetObjectRequest(bucket, key, localDir, localFileName);
//设置进度回调
request.SetCosProgressCallback(delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
GetObjectResult result = cosXml.GetObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
}
static void Main(string[] args)
{
GetObjectModel m = new GetObjectModel();
/// 下载对象
m.GetObject();
}
}
}
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码二：下载对象到内存中

```
using COSXML.Model.Object;
using COSXML.Auth;
using System;
using COSXML;
namespace COSSnippet
{
public class GetObjectModel {
private CosXml cosXml;
GetObjectModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencen
t.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://
console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 http
s://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 下载返回 bytes 数据
public void downloadToMem() {
try
{
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键

GetObjectBytesRequest request = new GetObjectBytesRequest(bucket, key);
//设置进度回调
request.SetCosProgressCallback(delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / tota
l));
});
//执行请求
GetObjectBytesResult result = cosXml.GetObject(request);
//获取内容到 byte 数组中
byte[] content = result.content;
//请求成功
```

```
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
}
// .cssg-methods-pragma
static void Main(string[] args)
{
GetObjectModel m = new GetObjectModel();
/// 下载对象到内存中
m.downloadToMem();
// .cssg-methods-pragma
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

复制与移动对象

最近更新时间：2022-01-23 14:45:54

简介

本文档提供关于对象复制与移动操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object - Copy	设置对象复制（修改对象属性）	复制文件到目标路径
DELETE Object	删除单个对象	在存储桶中删除指定对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块	查询正在进行中的分块信息
Initiate Multipart Upload	初始化分块	初始化分块操作
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传/复制块	查询特定分块操作中的已复制/上传的块
Complete Multipart Upload	完成分块上传/复制	完成整个文件的分块上传/复制
Abort Multipart Upload	终止分块上传/复制	终止一个分块操作并删除已上传/复制的块

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

高级接口（推荐）

复制对象

功能说明

高级接口封装了简单复制、分块复制接口的异步请求，并支持暂停、恢复以及取消复制请求。

说明：

- 对象大小小于分块阈值时选择简单复制，超过阈值时使用分块复制，阈值支持用户自行配置，默认为5MB。
- 分块大小支持用户自行配置，默认为2MB。

示例代码

```
using COSXML.Model.Tag;
using COSXML.Auth;
using COSXML.Transfer;
using System;
using COSXML;
using System.Threading.Tasks;
namespace COSSnippet
{
public class TransferCopyObjectModel {
private CosXml cosXml;
TransferCopyObjectModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://intl.cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 高级接口拷贝对象
public async Task TransferCopyObject()
{
TransferConfig transferConfig = new TransferConfig();
//手动设置分块复制阈值, 小于阈值的对象使用简单复制, 大于阈值的对象使用分块复制, 不设定则默认为5MB
transferConfig.DdivisionForCopy = 5242880;
```

```
//手动设置高级接口的自动分块大小，不设定则默认为2MB
transferConfig.SliceSizeForCopy = 2097152;

// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXml, transferConfig);
string sourceAppid = "1250000000"; //账号 appid
string sourceBucket = "sourcebucket-1250000000"; //"源对象所在的存储桶
string sourceRegion = "COS_REGION"; //源对象的存储桶所在的地域
string sourceKey = "sourceObject"; //源对象键
//构造源对象属性
CopySourceStruct copySource = new CopySourceStruct(sourceAppid, sourceBucket,
sourceRegion, sourceKey);
string bucket = "examplebucket-1250000000"; //目标存储桶，格式：BucketName-APPID
string key = "exampleobject"; //目标对象的对象键
COSXMLCopyTask copytask = new COSXMLCopyTask(bucket, key, copySource);

try {
COSXML.Transfer.COSXMLCopyTask.CopyTaskResult result = await
transferManager.CopyAsync(copytask);
Console.WriteLine(result.GetResultInfo());
string eTag = result.eTag;
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

}
static void Main(string[] args)
{
TransferCopyObjectModel m = new TransferCopyObjectModel();
/// 高级接口拷贝对象
m.TransferCopyObject().Wait();
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

简单操作

复制对象（修改属性）

功能说明

复制文件到目标路径（PUT Object - Copy）。复制对象接口，能够在复制请求中设置修改对象属性及元数据，并且支持原地复制，即能够支持原地修改对象元数据的特性。

示例代码一: 复制对象时保留对象属性

```
try
{
string sourceAppid = "1250000000"; //账号 appid
string sourceBucket = "sourcebucket-1250000000"; //"源对象所在的存储桶
string sourceRegion = "COS_REGION"; //源对象的存储桶所在的地域
string sourceKey = "sourceObject"; //源对象键
//构造源对象属性
CopySourceStruct copySource = new CopySourceStruct(sourceAppid, sourceBucket,
sourceRegion, sourceKey);
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
CopyObjectRequest request = new CopyObjectRequest(bucket, key);
//设置拷贝源
request.SetCopySource(copySource);
//设置是否拷贝还是更新,此处是拷贝
request.SetCopyMetaDataDirective(COSXML.Common.CosMetaDataDirective.Copy);
//执行请求
CopyObjectResult result = cosXml.CopyObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
```

```
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码二: 修改对象存储类型

说明：

- 标准存储可以修改为低频存储、智能分层存储、归档存储和深度归档存储等。
- 如果希望将归档存储或深度归档存储的对象修改为其他存储类型，首先需要使用 PostRestore 将归档存储或深度归档存储的对象回热，才能使用该接口请求修改存储类型。
- 关于存储类型的详细说明请参见 存储类型概述。

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string appId = "1250000000"; //账号 appid
string region = "COS_REGION"; //源对象的存储桶所在的地域
//构造对象属性
CopySourceStruct copySource = new CopySourceStruct(appId, bucket,
region, key);
CopyObjectRequest request = new CopyObjectRequest(bucket, key);
//设置拷贝源
request.SetCopySource(copySource);
//设置是否拷贝还是更新,此处是拷贝
request.SetCopyMetaDataDirective(COSXML.Common.CosMetaDataDirective.Replaced);
// 修改为归档存储
request.SetCosStorageClass("ARCHIVE");
//执行请求
CopyObjectResult result = cosXml.CopyObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
```

```
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

示例代码三: 复制对象时替换对象属性

```
try
{
string sourceAppid = "1250000000"; //账号 appid
string sourceBucket = "sourcebucket-1250000000"; //"源对象所在的存储桶
string sourceRegion = "COS_REGION"; //源对象的存储桶所在的地域
string sourceKey = "sourceObject"; //源对象键
//构造源对象属性
CopySourceStruct copySource = new CopySourceStruct(sourceAppid, sourceBucket,
sourceRegion, sourceKey);
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
CopyObjectRequest request = new CopyObjectRequest(bucket, key);
//设置拷贝源
request.SetCopySource(copySource);
//设置是否拷贝还是更新,此处是拷贝
request.SetCopyMetaDataDirective(COSXML.Common.CosMetaDataDirective.Replaced);
// 替换元数据
request.SetRequestHeader("Content-Disposition", "attachment; filename=example.jp
g");
//执行请求
CopyObjectResult result = cosXml.CopyObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
```

```
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码四: 修改对象元数据

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string appId = "1250000000"; //账号 appid
string region = "COS_REGION"; //源对象的存储桶所在的地域
//构造对象属性
CopySourceStruct copySource = new CopySourceStruct(appId, bucket,
region, key);
CopyObjectRequest request = new CopyObjectRequest(bucket, key);
//设置拷贝源
request.SetCopySource(copySource);
//设置是否拷贝还是更新,此处是拷贝
request.SetCopyMetaDataDirective(COSXML.Common.CosMetaDataDirective.Replaced);
// 替换元数据
request.SetRequestHeader("Content-Disposition", "attachment; filename=example.jpg");
request.SetRequestHeader("Content-Type", "image/png");
//执行请求
CopyObjectResult result = cosXml.CopyObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
```

```
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

移动对象

移动对象主要包括两个操作：复制源对象到目标位置，删除源对象。

由于 COS 通过存储桶名称（Bucket）和对象键（ObjectKey）来标识对象。移动对象也就意味着修改这个对象的标识，COS .NET SDK 目前没有提供修改对象唯一标识名的单独接口，但是可以通过组合**复制对象**加上**删除对象**的基本操作，来达到修改对象标识的目的，从而实现移动对象。

例如将 mybucket-1250000000 这个存储桶中的 picture.jpg 这个对象移动到同个存储桶的 doc 路径下。首先可以复制 picture.jpg 对象到存储桶的 doc 路径下，即对象键设定为 doc/picture.jpg，复制完成后删除 picture.jpg 这个对象，来实现“移动”的效果。

同样的，如果想将 mybucket-1250000000 这个存储桶里的 picture.jpg 这个对象移动到 myanothorbucket-1250000000 这个存储桶里，可以先将对象复制到 myanothorbucket-1250000000 存储桶，然后删除原来的对象。

示例代码

```
string sourceAppid = "1250000000"; // 账号 appid
// 源对象所在存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 http
s://console.intl.cloud.tencent.com/developer
string sourceBucket = "sourcebucket-1250000000";
string sourceRegion = "COS_REGION"; //源对象的存储桶所在的地域
string sourceKey = "sourceObject"; //源对象键
//构造源对象属性
CopySourceStruct copySource = new CopySourceStruct(sourceAppid, sourceBucket,
sourceRegion, sourceKey);
// 目标存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://cons
ole.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //目标对象的对象键
COSXMLCopyTask copyTask = new COSXMLCopyTask(bucket, key, copySource);
try {
// 拷贝对象
COSXML.Transfer.COSXMLCopyTask.CopyTaskResult result = await
```

```
transferManager.CopyAsync(copyTask);
Console.WriteLine(result.GetResultInfo());
// 删除对象
DeleteObjectRequest request = new DeleteObjectRequest(sourceBucket, sourceKey);
DeleteObjectResult deleteResult = cosXml.DeleteObject(request);
// 打印结果
Console.WriteLine(deleteResult.GetResultInfo());
} catch (Exception e) {
Console.WriteLine("CosException: " + e);
}
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

分块操作

这里主要介绍分块复制的流程。需要注意，COS API 接口提供了 InitiateMultipartUpload、UploadPart/UploadPartCopy、CompleteMultipartUpload 一套接口可用于实现分块复制。因此分块上传和分块复制在涉及接口及使用上具有一定重合度。

分块复制的流程

1. 初始化分块（Initiate Multipart Upload），得到 UploadId。
2. 使用 UploadId 复制分块（Upload Part - Copy）。
3. 完成分块复制（Complete Multipart Upload）。

剩余分块复制的流程

1. 如果没有记录 UploadId，查询分块任务（List Multipart Uploads），得到对应文件的 UploadId。
2. 使用 UploadId 列出已复制的分块（List Parts）。
3. 使用 UploadId 复制剩余的分块（Upload Part Copy）。
4. 完成分块复制（Complete Multipart Upload）。

终止分块复制的流程

1. 如果没有记录 UploadId，查询分块复制任务（List Multipart Uploads），得到对应文件的 UploadId。
2. 终止分块复制并删除已复制分块（Abort Multipart Upload）。

查询分块复制

版权所有：腾讯云计算（北京）有限责任公司

功能说明

查询指定存储桶中正在进行的分块复制（List Multipart Uploads）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
ListMultiUploadsRequest request = new ListMultiUploadsRequest(bucket);
//执行请求
ListMultiUploadsResult result = cosXml.ListMultiUploads(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

初始化分块复制

功能说明

初始化 Multipart Upload 复制操作，获取对应的 uploadId（Initiate Multipart Upload）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
```

```
string key = "exampleobject"; //对象键
InitMultipartUploadRequest request = new InitMultipartUploadRequest(bucket, key);
//执行请求
InitMultipartUploadResult result = cosXml.InitMultipartUpload(request);
//请求成功
this.uploadId = result.initMultipartUpload.uploadId; //用于后续分块的 uploadId
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

复制分块

功能说明

将其他对象复制为一个分块（Upload Part - Copy）。

示例代码

```
try
{
string sourceAppid = "1250000000"; //账号 appid
string sourceBucket = "sourcebucket-1250000000"; //"源对象所在的存储桶
string sourceRegion = "COS_REGION"; //源对象的存储桶所在的地域
string sourceKey = "sourceObject"; //源对象键
//构造源对象属性
COSXML.Model.Tag.CopySourceStruct copySource = new CopySourceStruct(sourceAppid,
sourceBucket, sourceRegion, sourceKey);
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string uploadId = this.uploadId; //初始化分块上传/复制返回的uploadId
```

```
int partNumber = 1; //分块编号，必须从1开始递增
UploadPartCopyRequest request = new UploadPartCopyRequest(bucket, key,
partNumber, uploadId);
//设置拷贝源
request.SetCopySource(copySource);
//设置复制分块（指定块的范围，如 0 ~ 1M）
request.SetCopyRange(0, 1024 * 1024);
//执行请求
UploadPartCopyResult result = cosXml.PartCopy(request);
//请求成功
//获取返回分块的eTag,用于后续CompleteMultiUploads
this.eTag = result.copyPart.eTag;
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

查询已复制的分块

功能说明

查询特定分块复制操作中的已复制的块（List Parts）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string uploadId = "exampleUploadId"; //初始化分块上传/复制返回的uploadId
ListPartsRequest request = new ListPartsRequest(bucket, key, uploadId);
```

```
//执行请求
ListPartsResult result = cosXml.ListParts(request);
//请求成功
//列举已上传/复制的分块
List<COSXML.Model.Tag.ListParts.Part> alreadyUploadParts = result.listParts.part
s;
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

完成分块复制

功能说明

完成整个文件的分块复制（Complete Multipart Upload）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string uploadId = "exampleUploadId"; //初始化分块/复制返回的uploadId
CompleteMultipartUploadRequest request = new CompleteMultipartUploadRequest(bucke
t,
key, uploadId);
//设置已/复制的parts,必须有序，按照partNumber递增
request.SetPartNumberAndETag(1, this.eTag);
//执行请求
CompleteMultipartUploadResult result = cosXml.CompleteMultiUpload(request);
```

```
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

终止分块复制

功能说明

终止一个分块复制操作并删除已复制的块（Abort Multipart Upload）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string uploadId = "exampleUploadId"; //初始化分返回的uploadId
AbortMultipartUploadRequest request = new AbortMultipartUploadRequest(bucket, key, uploadId);
//执行请求
AbortMultipartUploadResult result = cosXml.AbortMultiUpload(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
```

```
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

列出对象

最近更新时间：2021-11-30 12:24:31

简介

本文档提供关于列出对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象
GET Bucket Object Versions	查询对象及其历史版本列表	查询存储桶下的部分或者全部对象及其历史版本信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

示例代码一: 获取第一页数据

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
GetBucketRequest request = new GetBucketRequest(bucket);
//执行请求
GetBucketResult result = cosXml.GetBucket(request);
//bucket的相关信息
ListBucket info = result.listBucket;
if (info.isTruncated) {
// 数据被截断，记录下数据下标
this.nextMarker = info.nextMarker;
```

```
}
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码二：请求下一页数据

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
GetBucketRequest request = new GetBucketRequest(bucket);
//上一次拉取数据的下标
request.SetMarker(this.nextMarker);
//执行请求
GetBucketResult result = cosXml.GetBucket(request);
//bucket的相关信息
ListBucket info = result.listBucket;
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码三：获取对象列表与子目录

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
GetBucketRequest request = new GetBucketRequest(bucket);
//获取 a/ 下的对象以及子目录
request.SetPrefix("a/");
request.SetDelimiter("/");
//执行请求
GetBucketResult result = cosXml.GetBucket(request);
//bucket的相关信息
ListBucket info = result.listBucket;
// 对象列表
List<ListBucket.Contents> objects = info.contentsList;
// 子目录列表
List<ListBucket.CommonPrefixes> subDirs = info.commonPrefixesList;
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

查询对象历史版本列表

功能说明

查询开启版本控制的存储桶下的部分或者全部对象。

示例代码一：获取对象历史版本列表第一页数据

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
ListBucketVersionsRequest request = new ListBucketVersionsRequest(bucket);
//执行请求
ListBucketVersionsResult result = cosXml.ListBucketVersions(request);
//bucket的相关信息
ListBucketVersions info = result.listBucketVersions;
List<ListBucketVersions.Version> objects = info.objectVersionList;
List<ListBucketVersions.CommonPrefixes> prefixes = info.commonPrefixesList;
if (info.isTruncated) {
// 数据被截断，记录下数据下标
this.keyMarker = info.nextKeyMarker;
this.versionIdMarker = info.nextVersionIdMarker;
}
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

示例代码二：获取对象历史版本列表下一页数据

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
```

```
string bucket = "examplebucket-1250000000";
ListBucketVersionsRequest request = new ListBucketVersionsRequest(bucket);
// 上一页的数据结束下标
request.SetKeyMarker(this.keyMarker);
request.SetVersionIdMarker(this.versionIdMarker);
//执行请求
ListBucketVersionsResult result = cosXml.ListBucketVersions(request);
ListBucketVersions info = result.listBucketVersions;
if (info.isTruncated) {
// 数据被截断，记录下数据下标
this.keyMarker = info.nextKeyMarker;
this.versionIdMarker = info.nextVersionIdMarker;
}
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

删除对象

最近更新时间：2022-01-13 12:38:27

简介

本文档提供关于对象的删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

删除单个对象

功能说明

删除指定的对象（DELETE Object）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
DeleteObjectRequest request = new DeleteObjectRequest(bucket, key);
//执行请求
DeleteObjectResult result = cosXml.DeleteObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
```

```
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

删除多个对象

功能说明

批量删除多个对象（DELETE Multiple Objects）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
DeleteMultiObjectRequest request = new DeleteMultiObjectRequest(bucket);
//设置返回结果形式
request.SetDeleteQuiet(false);
//对象key
string key = "exampleobject"; //对象键
List<string> objects = new List<string>();
objects.Add(key);
request.SetObjectKeys(objects);
//执行请求
DeleteMultiObjectResult result = cosXml.DeleteMultiObjects(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
```

```
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

指定前缀删除(删除文件夹)

功能说明

指定前缀删除可以实现类似于删除目录的功能。
对象存储中本身是没有目录的概念的，为了满足用户使用习惯，用户可通过分隔符/来模拟“目录”。
删除目录及其文件这一场景，实际在 COS 上相当于删除一批有着同样前缀的对象。目前 COS .NET SDK 没有提供一个接口去实现这样的操作，但是可以通过组合查询对象列表加上批量删除对象的基本操作，达到删除文件夹下对象的目的。

示例代码

```
try
{
String nextMarker = null;
// 循环请求直到没有下一页数据
do
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string prefix = "folder1/"; //指定前缀
GetBucketRequest listRequest = new GetBucketRequest(bucket);
//获取 folder1/ 下的所有对象以及子目录
listRequest.SetPrefix(prefix);
listRequest.SetMarker(nextMarker);
//执行列出对象请求
GetBucketResult listResult = cosXml.GetBucket(listRequest);
ListBucket info = listResult.listBucket;
// 对象列表
List<ListBucket.Contents> objects = info.contentsList;
// 下一页的下标
```

```
nextMarker = info.nextMarker;

DeleteMultiObjectRequest deleteRequest = new DeleteMultiObjectRequest(bucket);
//设置返回结果形式
deleteRequest.SetDeleteQuiet(false);
//对象列表
List<string> deleteObjects = new List<string>();
foreach (var content in objects)
{
deleteObjects.Add(content.key);
}
deleteRequest.SetObjectKeys(deleteObjects);
//执行批量删除请求
DeleteMultiObjectResult deleteResult = cosXml.DeleteMultiObjects(deleteRequest);
//打印请求结果
Console.WriteLine(deleteResult.GetResultInfo());
} while (nextMarker != null);
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

判断对象是否存在

最近更新时间：2022-01-17 12:50:00

简介

本文档提供关于检查对象是否存在的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

检查对象是否存在

功能说明

此接口接受存储桶名称、对象名两个参数作为入参，返回对象是否存在的布尔类型。接口实际上为 HeadObject（查询对象元数据）接口的封装，需要获取对象详细元数据，请使用 查询对象元数据。

示例代码

```
using COSXML.Model.Object;
using COSXML.Auth;
using System;
using COSXML;
namespace COSSnippet
{
public class DoesObjectExistModel {
private CosXml cosXml;
DoesObjectExistModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencen
t.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://
console.intl.cloud.tencent.com/cam/capi
```

```
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 http
s://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 检查对象是否存在
public void DoesObjectExist()
{
try
{
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
DoesObjectExistRequest request = new DoesObjectExistRequest(bucket, key);
//执行请求
bool exist = cosXml.DoesObjectExist(request);
//请求成功
Console.WriteLine("object exist state is: " + exist);
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

//.cssg-snippet-body-end
}
// .cssg-methods-pragma
static void Main(string[] args)
{
DoesObjectExistModel m = new DoesObjectExistModel();
/// 检查对象是否存在
m.DoesObjectExist();
// .cssg-methods-pragma
}
}
}
```

说明：
更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

恢复归档对象

最近更新时间：2021-11-30 12:24:32

简介

本文档提供关于恢复归档对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
POST Object restore	恢复归档对象	将归档类型的对象取回访问

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

恢复归档对象

功能说明

将归档类型的对象取回访问（POST Object restore）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
RestoreObjectRequest request = new RestoreObjectRequest(bucket, key);
//恢复时间
request.SetExpireDays(3);
request.SetTier(COSXML.Model.Tag.RestoreConfigure.Tier.Bulk);
//执行请求
RestoreObjectResult result = cosXml.RestoreObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
```

版权所有：腾讯云计算（北京）有限责任公司

```
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

查询对象元数据

最近更新时间：2021-12-14 15:44:01

简介

本文档提供关于查询对象元数据操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

查询对象元数据

功能说明

查询 Object 的 Meta 信息，包括用户自定义头部，Etag等对象元信息

示例代码

```
using COSXML.Model.Object;
using COSXML.Auth;
using System;
using COSXML;
namespace COSSnippet
{
public class HeadObjectModel {
private CosXml cosXml;
HeadObjectModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的区域, COS 地域的简称请参照 https://cloud.tencen
t.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://
console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 http
```

```
s://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长，单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 获取对象信息
public void HeadObject()
{
//.cssg-snippet-body-start:[head-object]
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
HeadObjectRequest request = new HeadObjectRequest(bucket, key);
//执行请求
HeadObjectResult result = cosXml.HeadObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
// 打印对象crc64
Console.WriteLine(result.crc64ecma);
// 打印对象eTag
Console.WriteLine(result.eTag);
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

//.cssg-snippet-body-end
}
// .cssg-methods-pragma
static void Main(string[] args)
{
HeadObjectModel m = new HeadObjectModel();
/// 获取对象信息
m.HeadObject();
```

```
// .cssg-methods-pragma
}
}
}
```

说明：
更多完整示例，请前往 GitHub 查看。

对象访问 URL

最近更新时间：2022-05-25 15:36:34

简介

本文档提供关于生成对象访问 URL 的示例代码。

说明：

- 本文档中生成的访问对象 URL 方法，仅适用于通过默认域名访问公有读对象。
- 当对象为私有读对象时，请参见 生成预签名 URL 文档生成。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

生成对象访问链接

当对象 ACL 属性设置为“公有读”时，可以通过以下 SDK 接口生成的 URL 直接访问对象（仅支持生成 COS 默认源站域名的 URL）。

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
// 生成公有读URL
string requestURL = cosXml.GetObjectUrl(bucket, key);
//请求成功
Console.WriteLine(requestURL);
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
```

```
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

生成预签名 URL

最近更新时间：2022-01-17 12:50:00

简介

本文档提供关于生成对象预签名链接的示例代码。

说明：

- 建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。
- 关于临时密钥的获取，请参见 临时密钥生成及使用指引。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

生成对象预签名链接

示例代码一：生成预签名 URL

```
try
{
PreSignatureStruct preSignatureStruct = new PreSignatureStruct();
// APPID 获取参考 https://console.intl.cloud.tencent.com/developer
preSignatureStruct.appid = "1250000000";
// 存储桶所在地域，COS 地域的简称请参照 https://cloud.tencent.com/document/product/436/6224
preSignatureStruct.region = "COS_REGION";
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
preSignatureStruct.bucket = "examplebucket-1250000000";
preSignatureStruct.key = "exampleobject"; //对象键
preSignatureStruct.httpMethod = "GET"; //HTTP 请求方法
preSignatureStruct.isHttps = true; //生成 HTTPS 请求 URL
```

```
preSignatureStruct.signDurationSecond = 600; //请求签名时间为600s
preSignatureStruct.headers = null;//签名中需要校验的 header
preSignatureStruct.queryParameters = null; //签名中需要校验的 URL 中请求参数
string requestSignURL = cosXml.GenerateSignURL(preSignatureStruct);
//下载请求预签名 URL (使用永久密钥方式计算的签名 URL)
string localDir = System.IO.Path.GetTempPath();//本地文件夹
string localFileName = "my-local-temp-file"; //指定本地保存的文件名
GetObjectRequest request = new GetObjectRequest(null, null, localDir, localFileNa
me);
//设置下载请求预签名 URL
request.RequestURLWithSign = requestSignURL;
//设置进度回调
request.SetCosProgressCallback(delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / tota
l));
});
//执行请求
GetObjectResult result = cosXml.GetObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

示例代码二：生成预签名上传链接

```
try
{
PreSignatureStruct preSignatureStruct = new PreSignatureStruct();
// APPID 获取参考 https://console.intl.cloud.tencent.com/developer
preSignatureStruct.appid = "1250000000";
```

```
// 存储桶所在地域，COS 地域的简称请参照 https://cloud.tencent.com/document/product/436/6224
preSignatureStruct.region = "COS_REGION";
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
preSignatureStruct.bucket = "examplebucket-1250000000";
preSignatureStruct.key = "exampleobject"; //对象键
preSignatureStruct.httpMethod = "PUT"; //HTTP 请求方法
preSignatureStruct.isHttps = true; //生成 HTTPS 请求 URL
preSignatureStruct.signDurationSecond = 600; //请求签名时间为 600s
preSignatureStruct.headers = null;//签名中需要校验的 header
preSignatureStruct.queryParameters = null; //签名中需要校验的 URL 中请求参数
//上传预签名 URL（使用永久密钥方式计算的签名 URL）
string requestSignURL = cosXml.GenerateSignURL(preSignatureStruct);
string srcPath = @"temp-source-file";//本地文件绝对路径
PutObjectRequest request = new PutObjectRequest(null, null, srcPath);
//设置上传请求预签名 URL
request.RequestURLWithSign = requestSignURL;
//设置进度回调
request.SetCosProgressCallback(delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
PutObjectResult result = cosXml.PutObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码三：生成预签名 URL，并在签名中携带 Host

```
try
{
PreSignatureStruct preSignatureStruct = new PreSignatureStruct();
// APPID 获取参考 https://console.intl.cloud.tencent.com/developer
preSignatureStruct.appid = "1250000000";
// 存储桶所在地域，COS 地域的简称请参照 https://cloud.tencent.com/document/product/436/6224
preSignatureStruct.region = "COS_REGION";
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
preSignatureStruct.bucket = "examplebucket-1250000000";
preSignatureStruct.key = "exampleobject"; //对象键
preSignatureStruct.httpMethod = "GET"; //HTTP 请求方法
preSignatureStruct.isHttps = true; //生成 HTTPS 请求 URL
preSignatureStruct.signDurationSecond = 600; //请求签名时间为600s
preSignatureStruct.signHost = true; // 请求中签入Host，建议开启，能够有效防止越权请求，需要注意，开启后实际请求也需要携带Host请求头
preSignatureStruct.headers = null;//签名中需要校验的 header
preSignatureStruct.queryParameters = null; //签名中需要校验的 URL 中请求参数
string requestSignURL = cosXml.GenerateSignURL(preSignatureStruct);
Console.WriteLine("requestUrl is:" + requestSignURL);
//下载请求预签名 URL（使用永久密钥方式计算的签名 URL）
string localDir = System.IO.Path.GetTempPath();//本地文件夹
string localFileName = "my-local-temp-file"; //指定本地保存的文件名
GetObjectRequest request = new GetObjectRequest(null, null, localDir, localFileName);
//设置下载请求预签名 URL
request.RequestURLWithSign = requestSignURL;
//设置进度回调
request.SetCosProgressCallback(delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
GetObjectResult result = cosXml.GetObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
```

```
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

示例代码四：生成预签名URL，并在签名中携带请求参数

```
try
{
PreSignatureStruct preSignatureStruct = new PreSignatureStruct();
// APPID 获取参考 https://console.intl.cloud.tencent.com/developer
preSignatureStruct.appid = "1250000000";
// 存储桶所在地域，COS 地域的简称请参照 https://cloud.tencent.com/document/product/436/6224
preSignatureStruct.region = "COS_REGION";
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
preSignatureStruct.bucket = "examplebucket-1250000000";
preSignatureStruct.key = "exampleobject"; //对象键
preSignatureStruct.httpMethod = "GET"; //HTTP 请求方法
preSignatureStruct.isHttps = true; //生成 HTTPS 请求 URL
preSignatureStruct.signDurationSecond = 600; //请求签名时间为600s
preSignatureStruct.signHost = true; // 请求中签入Host，建议开启，能够有效防止越权请求，需要注意，开启后实际请求也需要携带Host请求头
preSignatureStruct.headers = null; // 签名中需要校验的 header
string ci_params = "imageMogr2/thumbnail/!50p";
preSignatureStruct.queryParameters = new Dictionary<string, string>(); // 签名中需要校验的 URL 中请求参数，以请求万象图片处理为例
preSignatureStruct.queryParameters.Add(ci_params, null);
string requestSignURL = cosXml.GenerateSignURL(preSignatureStruct);
Console.WriteLine("requestUrl is:" + requestSignURL);
//下载请求预签名 URL（使用永久密钥方式计算的签名 URL）
string localDir = System.IO.Path.GetTempPath();//本地文件夹
string localFileName = "my-local-temp-file"; //指定本地保存的文件名
GetObjectRequest request = new GetObjectRequest(null, null, localDir, localFileName);
//设置下载请求预签名 URL
request.RequestURLWithSign = requestSignURL;
//设置进度回调
request.SetCosProgressCallback(delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
GetObjectResult result = cosXml.GetObject(request);
//请求成功
```

```
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

预请求跨域配置

最近更新时间：2020-08-21 11:54:44

简介

本文档提供关于预请求跨域配置操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
Options Object	预请求跨域配置	用预请求来确认是否可以发送真正的跨域请求

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

预请求跨域配置

功能说明

获取预请求跨域配置（Options Object）。

示例代码

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象键
string origin = "http://cloud.tencent.com";
string accessMthod = "PUT";
OptionObjectRequest request = new OptionObjectRequest(bucket, key, origin, access
Mthod);

//执行请求
OptionObjectResult result = cosXml.OptionObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
```

```
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

服务端加密

最近更新时间：2021-12-27 12:43:13

简介

本文档提供关于如何使用在上传对象时开启服务端加密。服务端加密的密钥分为三种：

- COS 托管加密密钥
- KMS 托管加密密钥
- 客户提供的加密密钥

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

使用 COS 托管加密密钥的服务端加密（SSE-COS）保护数据

功能说明

由腾讯云 COS 托管主密钥和管理数据。COS 会帮助您在数据写入数据中心时自动加密，并在您取用该数据时自动解密。目前支持使用 COS 主密钥对数据进行 AES-256 加密。

示例代码

```
PutObjectRequest request = new PutObjectRequest(bucket, key, srcPath);
request.SetCosServerSideEncryption();
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

使用 KMS 托管加密密钥的服务端加密（SSE-KMS）保护数据

功能说明

SSE-KMS 加密即使用 KMS 托管密钥的服务端加密。KMS 是腾讯云推出的一款安全管理类服务，使用经过第三方认证的硬件安全模块 HSM（Hardware Security Module）来生成和保护密钥。它能够帮助用户轻松创建和管理密钥，满足用户多应用多业务的密钥管理需求以及满足监管和合规要求。关于如何开通 KMS 服务请参考：服务端加密概述。

示例代码

```
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
string srcPath = @"temp-source-file";//本地文件绝对路径
//.cssg-snippet-body-start:[put-object-sse-c]
PutObjectRequest request = new PutObjectRequest(bucket, key, srcPath);
request.SetCosServerSideEncryptionWithKMS("KMS Custem Key ID", "Context Json");
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

使用客户提供的加密密钥的服务端加密 （SSE-C）保护数据

功能说明

加密密钥由用户自己提供，用户在上传对象时，COS 将使用用户提供的加密密钥对用户的数据进行 AES-256 加密。

> 注意：
> - 该加密所运行的服务需要使用 HTTPS 请求。
> - 用户需要提供一个32字节的字符串作为密钥，支持数字、字母、字符的组合，不支持中文。
> - 如果上传文件时设置了密钥加密，那么在使用 GET（下载）、HEAD（查询）源对象时也需要在请求中带上相同的密钥，才能正常响应。

示例代码

```
PutObjectRequest request = new PutObjectRequest(bucket, key, srcPath);
request.SetCosServerSideEncryptionWithCustomerKey("Your Secret Key");
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

单链接限速

最近更新时间：2021-12-14 15:44:01

简介

本文档提供关于调用上传下载接口时对链接进行限速。

使用说明

限速值设置范围为**819200 - 838860800**，单位默认为 bit/s，即100KB/s - 100MB/s，如果超出该范围将返回400错误。

示例代码一：上传时对单链接限速

```
TransferConfig transferConfig = new TransferConfig();
// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXml, transferConfig);
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string cosPath = "dir/exampleObject"; // 对象键
string srcPath = @"temp-source-file";//本地文件绝对路径
PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, srcPath);
putObjectRequest.LimitTraffic(8 * 1024 * 1024); // 限制为1MB/s
COSXMLUploadTask uploadTask = new COSXMLUploadTask(putObjectRequest);
uploadTask.SetSrcPath(srcPath);
await transferManager.UploadAsync(uploadTask);
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

示例代码二：下载时对单链接限速

```
TransferConfig transferConfig = new TransferConfig();
// 初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXml, transferConfig);
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
```

```
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
string localDir = System.IO.Path.GetTempPath();//本地文件夹
string localFileName = "my-local-temp-file"; //指定本地保存的文件名
GetObjectRequest request = new GetObjectRequest(bucket,
cosPath, localDir, localFileName);
request.LimitTraffic(8 * 1024 * 1024); // 限制为1MB/s
COSXMLDownloadTask downloadTask = new COSXMLDownloadTask(request);
await transferManager.DownloadAsync(downloadTask);
```

说明：
更多完整示例，请前往 GitHub 查看。

检索对象内容

最近更新时间：2021-11-30 12:28:05

简介

本文档提供关于检索对象内容操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
SELECT Object Content	检索对象内容	从指定对象（CSV 格式或者 JSON 格式）中检索内容

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

检索对象内容

功能说明

COS Select 支持检索以下格式的对象数据：

- CSV 格式：对象以 CSV 格式存储，并以固定的分隔符划分。
- JSON 格式：对象以 JSON 格式存储，可以是 JSON 文件或者 JSON 列表。

> 注意：
>
> - 使用 COS Select，您必须具有 `cos:GetObject` 的授权。
> - CSV、JSON 对象需要以 UTF-8 格式编码。
> - COS Select 支持检索 GZIP 或者 BZIP2 压缩的 CSV、JSON 对象。
> - COS Select 支持检索 SSE-COS 加密的 CSV、JSON 对象。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
```

```
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
SelectObjectRequest request = new SelectObjectRequest(bucket, key);
ObjectSelectionFormat.JSONFormat jSONFormat = new ObjectSelectionFormat.JSONForma
t();
jSONFormat.Type = "DOCUMENT";
jSONFormat.RecordDelimiter = "\n";

string outputFile = "select_local_file.json";
request.SetExpression("Select * from COSObject")
.SetInputFormat(new ObjectSelectionFormat(null, jSONFormat))
.SetOutputFormat(new ObjectSelectionFormat(null, jSONFormat))
.SetCosProgressCallback(delegate (long progress, long total) {
Console.WriteLine("OnProgress : " + progress + "," + total);
})
.OutputToFile(outputFile)
;
SelectObjectResult selectObjectResult = cosXml.SelectObject(request);
Console.WriteLine(selectObjectResult.stat);
}
catch (COSXML.CosException.CosClientException clientEx)
{
Console.WriteLine("CosClientException: " + clientEx.StackTrace);
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

异地容灾
版本控制

最近更新时间：2021-11-30 12:29:43

简介

本文档提供关于版本控制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置版本控制

功能说明

设置指定存储桶的版本控制功能。开启版本控制功能后，只能暂停，不能关闭。

示例代码

```
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
PutBucketVersioningRequest request = new PutBucketVersioningRequest(bucket);
request.IsEnableVersionConfig(true); //true: 开启版本控制; false:暂停版本控制
try
{
PutBucketVersioningResult result = cosXml.PutBucketVersioning(request);
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
Console.WriteLine("CosClientException: " + clientEx);
```

```
}
catch (COSXML.CosException.CosServerException serverEx)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

查询版本控制

功能说明

查询指定存储桶的版本控制信息。

- 获取存储桶版本控制的状态，需要有该存储桶的读权限。
- 有三种版本控制状态：未启用版本控制、启用版本控制和暂停版本控制。

示例代码

```
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
GetBucketVersioningRequest request = new GetBucketVersioningRequest(bucket);
try
{
GetBucketVersioningResult result = cosXml.GetBucketVersioning(request);
// 存储桶的生命周期配置
VersioningConfiguration conf = result.versioningConfiguration;
}
catch (COSXML.CosException.CosClientException clientEx)
{
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

存储桶复制

最近更新时间：2021-11-30 12:29:43

简介

本文档提供关于存储桶复制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket replication	设置存储桶复制	设置存储桶的存储桶复制规则
GET Bucket replication	查询存储桶复制	查询存储桶的存储桶复制规则
DELETE Bucket replication	删除存储桶复制	删除存储桶的存储桶复制规则

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置存储桶复制

功能说明

设置指定存储桶的存储桶复制规则。

示例代码

```
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string ownerUin = "100000000001"; //发起者身份标示：OwnerUin
string subUin = "100000000001"; //发起者身份标示：SubUin
PutBucketReplicationRequest request = new PutBucketReplicationRequest(bucket);
//设置 replication
PutBucketReplicationRequest.RuleStruct ruleStruct =
new PutBucketReplicationRequest.RuleStruct();
ruleStruct.id = "replication_01"; //用来标注具体 Rule 的名称
ruleStruct.isEnable = true; //标识 Rule 是否生效 :true，生效； false，不生效
ruleStruct.appid = "1250000000"; //APPID
ruleStruct.region = "ap-beijing"; //目标存储桶地域信息
```

版权所有：腾讯云计算（北京）有限责任公司

```
ruleStruct.bucket = "destinationbucket-1250000000"; //格式：BucketName-APPID
ruleStruct.prefix = "34"; //前缀匹配策略
List<PutBucketReplicationRequest.RuleStruct> ruleStructs =
new List<PutBucketReplicationRequest.RuleStruct>();
ruleStructs.Add(ruleStruct);
request.SetReplicationConfiguration(ownerUin, subUin, ruleStructs);
try
{
PutBucketReplicationResult result = cosXml.PutBucketReplication(request);
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

查询存储桶复制

功能说明

查询指定存储桶的存储桶复制规则。

示例代码

```
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
GetBucketReplicationRequest request = new GetBucketReplicationRequest(bucket);
try
{
GetBucketReplicationResult result = cosXml.GetBucketReplication(request);
// 存储桶的跨区域复制配置
ReplicationConfiguration conf = result.replicationConfiguration;
}
catch (COSXML.CosException.CosClientException clientEx)
```

```
{
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

删除存储桶复制

功能说明

删除指定存储桶的存储桶复制规则。

示例代码

```
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
DeleteBucketReplicationRequest request = new DeleteBucketReplicationRequest(bucket);
try
{
DeleteBucketReplicationResult result = cosXml.DeleteBucketReplication(request);
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

跨地域复制

最近更新时间：2020-08-25 15:37:21

简介

本文档提供关于跨地域复制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置跨地域复制

功能说明

设置指定存储桶的跨地域复制规则。

示例代码

```
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
string ownerUin = "100000000001"; //发起者身份标示: OwnerUin
string subUin = "100000000001"; //发起者身份标示: SubUin
PutBucketReplicationRequest request = new PutBucketReplicationRequest(bucket);
//设置 replication
PutBucketReplicationRequest.RuleStruct ruleStruct =
new PutBucketReplicationRequest.RuleStruct();
ruleStruct.id = "replication_01"; //用来标注具体 Rule 的名称
ruleStruct.isEnable = true; //标识 Rule 是否生效 :true, 生效； false, 不生效
ruleStruct.appid = "1250000000"; //APPID
ruleStruct.region = "ap-beijing"; //目标存储桶地域信息
ruleStruct.bucket = "destinationbucket-1250000000"; //格式：BucketName-APPID
ruleStruct.prefix = "34"; //前缀匹配策略
```

```
List<PutBucketReplicationRequest.RuleStruct> ruleStructs =
new List<PutBucketReplicationRequest.RuleStruct>();
ruleStructs.Add(ruleStruct);
request.SetReplicationConfiguration(ownerUin, subUin, ruleStructs);

try
{
PutBucketReplicationResult result = cosXml.PutBucketReplication(request);
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

查询跨地域复制

功能说明

查询指定存储桶的跨地域复制规则。

示例代码

```
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
GetBucketReplicationRequest request = new GetBucketReplicationRequest(bucket);
try
{
GetBucketReplicationResult result = cosXml.GetBucketReplication(request);
// 存储桶的跨区域复制配置
ReplicationConfiguration conf = result.replicationConfiguration;
}
catch (COSXML.CosException.CosClientException clientEx)
{
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
```

```
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

删除跨地域复制

功能说明

删除指定存储桶的跨地域复制规则。

示例代码

```
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
DeleteBucketReplicationRequest request = new DeleteBucketReplicationRequest(bucke
t);
try
{
DeleteBucketReplicationResult result = cosXml.DeleteBucketReplication(request);
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

数据管理
生命周期

最近更新时间：2021-11-30 12:29:43

简介

本文档提供关于生命周期的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket lifecycle	设置生命周期	设置存储桶的生命周期管理的配置
GET Bucket lifecycle	查询生命周期	查询存储桶生命周期管理的配置
DELETE Bucket lifecycle	删除生命周期	删除存储桶生命周期管理的配置

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置生命周期

功能说明

设置指定存储桶的生命周期配置信息。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
PutBucketLifecycleRequest request = new PutBucketLifecycleRequest(bucket);
//设置 lifecycle
LifecycleConfiguration.Rule rule = new LifecycleConfiguration.Rule();
rule.id = "lfiecycleConfigureId";
rule.status = "Enabled"; //Enabled, Disabled
rule.filter = new COSXML.Model.Tag.LifecycleConfiguration.Filter();
```

```
rule.filter.prefix = "2/";
//指定分片过期删除操作
rule.abortIncompleteMultiUpload = new LifecycleConfiguration.AbortIncompleteMulti
Upload();
rule.abortIncompleteMultiUpload.daysAfterInitiation = 2;
request.SetRule(rule);
//执行请求
PutBucketLifecycleResult result = cosXml.PutBucketLifecycle(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

查询生命周期

功能说明

查询存储桶的生命周期管理配置。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
GetBucketLifecycleRequest request = new GetBucketLifecycleRequest(bucket);
//执行请求
GetBucketLifecycleResult result = cosXml.GetBucketLifecycle(request);
//存储桶的生命周期配置
```

```
LifecycleConfiguration conf = result.lifecycleConfiguration;
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

删除生命周期

功能说明

删除存储桶生命周期管理的配置。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
DeleteBucketLifecycleRequest request = new DeleteBucketLifecycleRequest(bucket);
//执行请求
DeleteBucketLifecycleResult result = cosXml.DeleteBucketLifecycle(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
```

```
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

日志管理

最近更新时间：2021-12-14 15:37:57

简介

本文档提供关于日志管理的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket logging	设置日志管理	为源存储桶开启日志记录
GET Bucket logging	查询日志管理	查询源存储桶的日志配置信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置日志管理

功能说明

PUT Bucket logging 用于为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
PutBucketLoggingRequest request = new PutBucketLoggingRequest(bucket);
// 设置保存日志的目标路径
request.SetTarget("targetbucket-1250000000", "logs/");
//执行请求
PutBucketLoggingResult result = cosXml.PutBucketLogging(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
```

版权所有：腾讯云计算（北京）有限责任公司

```
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

查询日志管理

功能说明

GET Bucket logging 用于查询指定存储桶的日志配置信息。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
GetBucketLoggingRequest request = new GetBucketLoggingRequest(bucket);
//执行请求
GetBucketLoggingResult getResult = cosXml.GetBucketLogging(request);
//请求成功
BucketLoggingStatus status = getResult.bucketLoggingStatus;
if (status != null && status.loggingEnabled != null) {
string targetBucket = status.loggingEnabled.targetBucket;
string targetPrefix = status.loggingEnabled.targetPrefix;
}
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
```

```
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

存储桶标签

最近更新时间：2021-12-14 15:37:57

简介

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket tagging	设置存储桶标签	为已存在的存储桶设置标签
GET Bucket tagging	查询存储桶标签	查询指定存储桶下已有的存储桶标签
DELETE Bucket tagging	删除存储桶标签	删除指定的存储桶标签

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置存储桶标签

功能说明

PUT Bucket tagging 用于为已存在的存储桶设置标签。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
PutBucketTaggingRequest request = new PutBucketTaggingRequest(bucket);
string akey = "aTagKey";
string avalue = "aTagValue";
string bkey = "bTagKey";
string bvalue = "bTagValue";
request.AddTag(akey, avalue);
request.AddTag(bkey, bvalue);
```

版权所有：腾讯云计算（北京）有限责任公司

```
//执行请求
PutBucketTaggingResult result = cosXml.PutBucketTagging(request);

//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

查询存储桶标签

功能说明

GET Bucket tagging 用于查询指定存储桶下已有的存储桶标签。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
GetBucketTaggingRequest request = new GetBucketTaggingRequest(bucket);
//执行请求
GetBucketTaggingResult result = cosXml.GetBucketTagging(request);

//请求成功
Tagging tagging = result.tagging;
Console.WriteLine(tagging);
}
catch (COSXML.CosException.CosClientException clientEx)
```

```
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

删除存储桶标签

功能说明

DELETE Bucket tagging 用于删除指定存储桶下已有的存储桶标签。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
DeleteBucketTaggingRequest request = new DeleteBucketTaggingRequest(bucket);
//执行请求
DeleteBucketTaggingResult result = cosXml.DeleteBucketTagging(request);

//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
```

```
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

对象标签

最近更新时间：2022-05-25 15:36:35

简介

本文档提供关于对象标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Object tagging	设置对象标签	为已存在的对象设置标签
GET Object tagging	查询对象标签	查询指定对象已有标签
DELETE Object tagging	删除对象标签	删除指定的对象标签

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置对象标签

功能说明

PUT Object tagging 用于为已存在的对象设置标签。

示例代码

```
using COSXML.Model;
using COSXML.Model.Object;
using COSXML.Model.Tag;
using COSXML.Auth;
using System;
using COSXML;
using System.Linq;
namespace COSSnippet
{
public class ObjectTaggingModel {
private CosXml cosXml;
ObjectTaggingModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
```

```
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://intl.cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 设置对象标签, 此接口从 5.4.25 版本开始支持
public void PutObjectTagging()
{
//.cssg-snippet-body-start:[put-object-tagging]
try
{
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
PutObjectTaggingRequest request = new PutObjectTaggingRequest(bucket, key);
// 增加标签键值对
request.AddTag("tag1", "value1");
//执行请求
PutObjectTaggingResult result = cosXml.PutObjectTagging(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

}
// .cssg-methods-pragma
static void Main(string[] args)
```

```
{
ObjectTaggingModel m = new ObjectTaggingModel();
/// 设置对象标签
m.PutObjectTagging();
// .cssg-methods-pragma
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

查询存储桶标签

功能说明

GET Object tagging 用于查询指定对象下已有的对象标签。

示例代码

```
using COSXML.Model;
using COSXML.Model.Object;
using COSXML.Model.Tag;
using COSXML.Auth;
using System;
using COSXML;
using System.Linq;
namespace COSSnippet
{
public class ObjectTaggingModel {
private CosXml cosXml;
ObjectTaggingModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://intl.cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
```

```
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 获取对象标签，此接口从 5.4.25 版本开始支持
public void GetObjectTagging()
{
//.cssg-snippet-body-start:[get-object-tagging]
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
GetObjectTaggingRequest request = new GetObjectTaggingRequest(bucket, key);
// 执行请求
GetObjectTaggingResult result = cosXml.GetObjectTagging(request);
// 请求成功
Console.WriteLine(result.GetResultInfo());
// 遍历输出Tagging列表
for (int i = 0; i < result.tagging.tagSet.tags.Count; i++) {
Console.WriteLine(result.tagging.tagSet.tags[i].key);
Console.WriteLine(result.tagging.tagSet.tags[i].value);
}
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
//.cssg-snippet-body-end
}
// .cssg-methods-pragma
static void Main(string[] args)
{
ObjectTaggingModel m = new ObjectTaggingModel();
/// 获取对象标签
m.GetObjectTagging();
// .cssg-methods-pragma
}
```

版权所有：腾讯云计算（北京）有限责任公司

```
}
}
```

说明：
更多完整示例，请前往 GitHub 查看。

删除对象标签

功能说明

DELETE Bucket tagging 用于删除指定对象下已有的对象标签。

示例代码

```
using COSXML.Model;
using COSXML.Model.Object;
using COSXML.Model.Tag;
using COSXML.Auth;
using System;
using COSXML;
using System.Linq;
namespace COSSnippet
{
public class ObjectTaggingModel {
private CosXml cosXml;
ObjectTaggingModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://intl.cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
```

```
/// 删除对象标签，此接口从 5.4.25 版本开始支持
public void DeleteObjectTagging()
{
//.cssg-snippet-body-start:[delete-object-tagging]
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
DeleteObjectTaggingRequest request = new DeleteObjectTaggingRequest(bucket, key);
// 执行请求
DeleteObjectTaggingResult result = cosXml.DeleteObjectTagging(request);
// 请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
//.cssg-snippet-body-end
}
// .cssg-methods-pragma
static void Main(string[] args)
{
ObjectTaggingModel m = new ObjectTaggingModel();
/// 删除对象标签
m.DeleteObjectTagging();
// .cssg-methods-pragma
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

静态网站

最近更新时间：2022-01-11 12:32:40

简介

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket website	设置静态网站	设置存储桶的静态网站配置
GET Bucket website	查询静态网站配置	查询存储桶的静态网站配置
DELETE Bucket website	删除静态网站配置	删除存储桶的静态网站配置

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置静态网站

功能说明

PUT Bucket website 用于为存储桶配置静态网站。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
PutBucketWebsiteRequest putRequest = new PutBucketWebsiteRequest(bucket);
putRequest.SetIndexDocument("index.html");
putRequest.SetErrorDocument("erorror.html");
putRequest.SetRedirectAllRequestTo("index.html");
PutBucketWebsiteResult putResult = cosXml.PutBucketWebsite(putRequest);

//请求成功
Console.WriteLine(putResult.GetResultInfo());
```

```
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

查询静态网站配置

功能说明

GET Bucket website 用于查询与存储桶关联的静态网站配置信息。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
GetBucketWebsiteRequest request = new GetBucketWebsiteRequest(bucket);
//执行请求
GetBucketWebsiteResult result = cosXml.GetBucketWebsite(request);

//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
```

```
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

删除静态网站配置

功能说明

DELETE Bucket website 用于删除存储桶中的静态网站配置。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
DeleteBucketWebsiteRequest request = new DeleteBucketWebsiteRequest(bucket);
//执行请求
DeleteBucketWebsiteResult result = cosXml.DeleteBucketWebsite(request);

//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

清单

最近更新时间：2022-01-23 14:45:55

简介

本文档提供关于清单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket inventory	设置清单任务	设置存储桶的清单任务
GET Bucket inventory	查询清单任务	查询存储桶的清单任务
DELETE Bucket inventory	删除清单任务	删除存储桶的清单任务

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置清单任务

功能说明

PUT Bucket inventory 用于在存储桶中创建清单任务。

示例代码

```
try
{
string inventoryId = "aInventoryId";
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
PutBucketInventoryRequest putRequest = new PutBucketInventoryRequest(bucket, inventoryId);
putRequest.SetDestination("CSV", "100000000001", "examplebucket-1250000000", "ap-guangzhou","list1");
putRequest.IsEnable(true);
// 清单任务周期，枚举值：Daily、Weekly
putRequest.SetScheduleFrequency("Daily");
```

```
// 是否在清单中包含对象版本，枚举值：All、Current
putRequest.SetIncludedObjectVersions("All");
//执行请求
PutBucketInventoryResult putResult = cosXml.PutBucketInventory(putRequest);

//请求成功
Console.WriteLine(putResult.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

错误码说明

该请求可能会发生的一些常见的特殊错误如下：

错误码	描述	状态码
InvalidArgument	不合法的参数值	HTTP 400 Bad Request
TooManyConfigurations	清单数量已经达到1000条的上限	HTTP 400 Bad Request
AccessDenied	未授权的访问。您可能不具备访问该存储桶的权限	HTTP 403 Forbidden

查询清单任务

功能说明

GET Bucket inventory 用于查询存储桶中用户的清单任务信息。

示例代码

```
try
{
string inventoryId = "aInventoryId";
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
GetBucketInventoryRequest getRequest = new GetBucketInventoryRequest(bucket);
getRequest.SetInventoryId(inventoryId);

GetBucketInventoryResult getResult = cosXml.GetBucketInventory(getRequest);

InventoryConfiguration configuration = getResult.inventoryConfiguration;
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

删除清单任务

功能说明

DELETE Bucket inventory 用于删除存储桶中指定的清单任务。

示例代码

```
try
{
string inventoryId = "aInventoryId";
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
```

```
DeleteBucketInventoryRequest deleteRequest = new DeleteBucketInventoryRequest(buc
ket);
deleteRequest.SetInventoryId(inventoryId);
DeleteBucketInventoryResult deleteResult = cosXml.DeleteBucketInventory(deleteReq
uest);

//请求成功
Console.WriteLine(deleteResult.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

访问管理
跨域访问

最近更新时间：2021-12-16 14:59:55

简介

本文档提供关于跨域访问的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域名访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域名访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域名访问配置信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置跨域配置

功能说明

设置指定存储桶的跨域名访问配置信息（PUT Bucket cors）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
PutBucketCORSRequest request = new PutBucketCORSRequest(bucket);
//设置跨域访问配置 CORS
COSXML.Model.Tag.CORSConfiguration.CORSRule corsRule =
new COSXML.Model.Tag.CORSConfiguration.CORSRule();
corsRule.id = "corsconfigureId";
corsRule.maxAgeSeconds = 6000;
```

```
corsRule.allowedOrigins = new List<string>();
corsRule.allowedOrigins.Add("http://cloud.tencent.com");
corsRule.allowedMethods = new List<string>();
corsRule.allowedMethods.Add("PUT");
corsRule.allowedHeaders = new List<string>();
corsRule.allowedHeaders.Add("Host");
corsRule.exposeHeaders = new List<string>();
corsRule.exposeHeaders.Add("x-cos-meta-x1");
request.SetCORSRule(corsRule);
//执行请求
PutBucketCORSResult result = cosXml.PutBucketCORS(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

查询跨域配置

功能说明

查询指定存储桶的跨域名访问配置信息（GET Bucket cors）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
```

```
GetBucketCORSRequest request = new GetBucketCORSRequest(bucket);
//执行请求
GetBucketCORSResult result = cosXml.GetBucketCORS(request);
//存储桶的 CORS 配置信息
CORSConfiguration conf = result.corsConfiguration;
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

删除跨域配置

功能说明

删除指定存储桶的跨域名访问配置（DELETE Bucket cors）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
DeleteBucketCORSRequest request = new DeleteBucketCORSRequest(bucket);
//执行请求
DeleteBucketCORSResult result = cosXml.DeleteBucketCORS(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
```

```
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

自定义域名

最近更新时间：2021-12-16 14:59:55

简介

本文档提供关于自定义域名的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket domain	设置自定义域名	设置存储桶的自定义域名信息
GET Bucket domain	查询自定义域名	查询存储桶的自定义域名信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置自定义域名

功能说明

PUT Bucket domain 用于为存储桶配置自定义域名。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";

DomainConfiguration domain = new DomainConfiguration();
domain.rule = new DomainConfiguration.DomainRule();
domain.rule.Name = "www.qq.com";
domain.rule.Status = "ENABLED";
domain.rule.Type = "WEBSITE";

PutBucketDomainRequest request = new PutBucketDomainRequest(bucket, domain);
//执行请求
PutBucketDomainResult result = cosXml.PutBucketDomain(request);
```

```
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

返回错误码说明

该请求可能会发生的一些常见的特殊错误如下：

状态码	说明
HTTP 409 Conflict	该域名记录已存在，且请求中没有设置强制覆盖。或者该域名记录不存在，且请求中设置了强制覆盖
HTTP 451 Unavailable For Legal Reasons	该域名是中国境内域名，并且没有备案

查询自定义域名

功能说明

GET Bucket domain 用于查询存储桶的自定义域名信息。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
```

```
string bucket = "examplebucket-1250000000";
GetBucketDomainRequest request = new GetBucketDomainRequest(bucket);
//执行请求
GetBucketDomainResult result = cosXml.GetBucketDomain(request);

//请求成功
Console.WriteLine(result.domainConfiguration);
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

返回参数说明

参数名称	描述	类型
x-cos-domain-txt-verification	域名校验信息，该字段是一个 MD5 校验值，原串格式为：`cos[Region][BucketName-APPID][BucketCreateTime]`，其中 Region 为存储桶所在地域，BucketCreateTime 为存储桶 GMT 创建时间	String

访问控制

最近更新时间：2021-12-14 15:45:37

简介

本文档提供关于存储桶、对象的访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

存储桶 ACL

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶的访问权限控制列表（ACL）
GET Bucket acl	查询存储桶 ACL	查询指定存储桶的访问权限控制列表（ACL）

对象 ACL

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表
GET Object acl	查询对象 ACL	查询对象的访问控制列表

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

存储桶 ACL

设置存储桶 ACL

功能说明

设置指定存储桶的访问权限控制列表（ACL）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
```

```
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
PutBucketACLRequest request = new PutBucketACLRequest(bucket);
//设置私有读写权限
request.SetCosACL(CosACL.Private);
//授予1131975903账号读权限
COSXML.Model.Tag.GrantAccount readAccount = new COSXML.Model.Tag.GrantAccount();
readAccount.AddGrantAccount("1131975903", "1131975903");
request.SetXCosGrantRead(readAccount);
//执行请求
PutBucketACLResult result = cosXml.PutBucketACL(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

查询存储桶 ACL

功能说明

查询指定存储桶的访问权限控制列表（ACL）。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
GetBucketACLRequest request = new GetBucketACLRequest(bucket);
//执行请求
GetBucketACLResult result = cosXml.GetBucketACL(request);
```

```
//存储桶的 ACL 信息
AccessControlPolicy acl = result.accessControlPolicy;
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：
更多完整示例，请前往 GitHub 查看。

对象 ACL

设置对象 ACL

功能说明

设置存储桶中某个对象的访问控制列表（ACL）。

示例代码

```
// 因为存储桶 ACL 最多1000条，为避免 ACL 达到上限，
// 非必须情况不建议给对象单独设置 ACL(对象默认继承 bucket 权限).
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
PutObjectACLRequest request = new PutObjectACLRequest(bucket, key);
//设置私有读写权限
request.SetCosACL(CosACL.Private);
//授予1131975903账号读权限
COSXML.Model.Tag.GrantAccount readAccount = new COSXML.Model.Tag.GrantAccount();
readAccount.AddGrantAccount("1131975903", "1131975903");
request.SetXCosGrantRead(readAccount);
```

```
//执行请求
PutObjectACLResult result = cosXml.PutObjectACL(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

查询对象 ACL

功能说明

查询对象的访问控制列表。

示例代码

```
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
GetObjectACLRequest request = new GetObjectACLRequest(bucket, key);
//执行请求
GetObjectACLResult result = cosXml.GetObjectACL(request);
//对象的 ACL 信息
AccessControlPolicy acl = result.accessControlPolicy;
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
```

```
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

说明：

更多完整示例，请前往 GitHub 查看。

防盗链

最近更新时间：2021-12-16 14:59:56

简介

本文档提供关于存储桶 Referer 白名单或者黑名单的 API 概览和 SDK 示例代码。

API	操作名	操作描述
PUT Bucket referer	设置存储桶 Referer	设置存储桶 Referer 白名单或者黑名单
GET Bucket referer	查询存储桶 Referer	查询存储桶 Referer 白名单或者黑名单

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置存储桶 Referer

功能说明

使用 .NET SDK 为存储桶设置 Referer 白名单或者黑名单，并读取存储桶的 Referer 配置。

> 说明：
>
> - 存储桶防盗链配置从 5.4.24 版本开始支持，下载新版 SDK 前往 Releases 或参见 快速入门。
> - 查看版本更新日志，请前往 GitHub。

示例代码

```
using COSXML.Model.Tag;
using COSXML.Model.Bucket;
using COSXML.Auth;
using System;
using COSXML;
namespace COSSnippet
{
```

```
public class BucketRefererModel {
private CosXml cosXml;
BucketRefererModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencen
t.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://
console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 http
s://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 设置存储桶防盗链
public void PutBucketReferer()
{
//.cssg-snippet-body-start:[put-bucket-cors]
try
{
// 存储桶名称, 此处填入格式必须为 BucketName-APPID, 其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
PutBucketRefererRequest request = new PutBucketRefererRequest(bucket);
// 设置防盗链规则
RefererConfiguration configuration = new RefererConfiguration();
// 是否开启防盗链, 枚举值：Enabled、Disabled
configuration.Status = "Enabled";
// 防盗链类型, 枚举值：Black-List、White-List
configuration.RefererType = "White-List";
// 生效域名列表, 支持多个域名且为前缀匹配, 支持带端口的域名和 IP, 支持通配符*, 做二级域名或多
级域名的通配
configuration.domainList = new DomainList();
// 单条生效域名 例如www.qq.com/example, 192.168.1.2:8080, *.qq.com
configuration.domainList.AddDomain("*.domain1.com");
configuration.domainList.AddDomain("*.domain2.com");
// 是否允许空 Referer 访问, 枚举值：Allow、Deny, 默认值为 Deny
configuration.EmptyReferConfiguration = "Deny";
request.SetRefererConfiguration(configuration);
//执行请求
PutBucketRefererResult result = cosXml.PutBucketReferer(request);
//请求成功
```

```
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

//.cssg-snippet-body-end
}
/// 获取存储桶防盗链规则
public void GetBucketReferer()
{
//.cssg-snippet-body-start:[get-bucket-cors]
try
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
GetBucketRefererRequest request = new GetBucketRefererRequest(bucket);
// 执行请求
GetBucketRefererResult result = cosXml.GetBucketReferer(request);
Console.WriteLine(result.GetResultInfo());
// Status参数
Console.WriteLine(result.refererConfiguration.Status);
// Referer名单类型
Console.WriteLine(result.refererConfiguration.RefererType);
// 名单中的域名列表
foreach (string domain in result.refererConfiguration.domainList.domains)
{
Console.WriteLine(domain);
}
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
```

```
}

//.cssg-snippet-body-end
}

// .cssg-methods-pragma
static void Main(string[] args)
{
BucketRefererModel m = new BucketRefererModel();
/// 设置存储桶跨域规则
m.PutBucketReferer();
/// 获取存储桶跨域规则
m.GetBucketReferer();
// .cssg-methods-pragma
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

图片处理
基础图片处理

最近更新时间：2022-10-09 16:09:43

简介

本文档提供关于基础图片处理的相关的 API 概览以及 SDK 示例代码。

服务	功能	说明
基础图片处理服务	缩放	等比缩放、设定目标宽高缩放等多种方式
	裁剪	普通裁剪、缩放裁剪、内切圆、人脸智能裁剪
	旋转	自适应旋转、普通旋转
	格式转换	格式转换、GIF 格式优化、渐进显示
	质量变换	针对 JPG 和 WEBP 图片进行质量变换
	高斯模糊	对图片进行模糊处理
	锐化	对图片进行锐化处理
	添加水印	图片水印、文字水印
	获取图片信息	基本信息、EXIF 信息、主色调
	去除元信息	包括 EXIF 信息
	快速缩略模板	快速实现图片格式转换、缩略、剪裁等功能，生成缩略图

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

上传时使用图片处理

下面示例展示了如何在上传图片时自动实现图片处理。

版权所有：腾讯云计算（北京）有限责任公司

图片上传完成后，COS 会存储原始图片和已处理过的图片。后续用户可以通过普通的下载请求获取处理结果。

示例代码

```
PutObjectRequest request = new PutObjectRequest(bucket, key, srcPath);
JObject o = new JObject();
// 不返回原图
o["is_pic_info"] = 0;
JArray rules = new JArray();
JObject rule = new JObject();
rule["bucket"] = bucket;
rule["fileid"] = "desample_photo.jpg";
//处理参数，规则参见：https://cloud.tencent.com/document/product/436/44879
rule["rule"] = "imageMogr2/thumbnail/400x400";
rules.Add(rule);
o["rules"] = rules;
string ruleString = o.ToString(Formatting.None);
request.SetRequestHeader("Pic-Operations", ruleString);
//执行请求
PutObjectResult result = cosXml.PutObject(request);
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

对云上数据进行图片处理

下面示例展示了如何在对已存储在 COS 的图片进行相应处理操作，并将结果存入到 COS。

示例代码

```
JObject o = new JObject();
// 不返回原图
o["is_pic_info"] = 0;
JArray rules = new JArray();
JObject rule = new JObject();
rule["bucket"] = bucket;
rule["fileid"] = "desample_photo.jpg";
//处理参数，规则参见：https://cloud.tencent.com/document/product/436/44879
rule["rule"] = "imageMogr2/thumbnail/400x400";
rules.Add(rule);
o["rules"] = rules;
```

版权所有：腾讯云计算（北京）有限责任公司

```
string ruleString = o.ToString(Formatting.None);
ImageProcessRequest request = new ImageProcessRequest(bucket, key, ruleString);
ImageProcessResult result = cosXml.ImageProcess(request);
```

说明：
更多完整示例，请前往 GitHub 查看。

下载时进行图片处理

下面示例展示了如何在对已存储在 COS 的图片在下载时进行处理操作。

示例代码

```
GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, key, localDir, localFileName);
//处理参数，这里的实例是格式转换为 TPG 图片，规则参见：https://cloud.tencent.com/document/product/436/44879
getObjectRequest.SetQueryParameter("imageMogr2/format/tpg", null);
```

说明：
更多完整示例，请前往 GitHub 查看。

盲水印

最近更新时间：2022-10-09 16:09:43

简介

本文档提供关于盲水印相关的 API 概览以及 SDK 示例代码。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

添加盲水印

功能说明

盲水印支持在上传时添加以及下载时添加。

示例代码一：上传时添加盲水印

```
PutObjectRequest request = new PutObjectRequest(bucket, key, srcPath);
JObject o = new JObject();
// 不返回原图
o["is_pic_info"] = 0;
JArray rules = new JArray();
JObject rule = new JObject();
rule["bucket"] = bucket;
rule["fileid"] = key;
//处理参数，规则参见：https://cloud.tencent.com/document/product/436/46782
rule["rule"] = "watermark/3/type/<type>/image/<imageUrl>/text/<text>/level/<level
>";
rules.Add(rule);
o["rules"] = rules;
string ruleString = o.ToString(Formatting.None);
request.SetRequestHeader("Pic-Operations", ruleString);
//执行请求
PutObjectResult result = cosXml.PutObject(request);
```

说明：

更多完整示例，请前往 GitHub 查看。

示例代码二：下载时添加盲水印

```
GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, key, localDir, l
ocalFileName);
//处理参数，规则参见：https://cloud.tencent.com/document/product/436/46782
getObjectRequest.SetQueryParameter("watermark/3/type/<type>/image/<imageUrl>/tex
t/<text>", null);
GetObjectResult result = cosXml.GetObject(getObjectRequest);
```

说明：
更多完整示例，请前往 GitHub 查看。

内容审核
图片审核

最近更新时间：2022-12-20 14:15:16

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于图片审核的 API 概览以及 SDK 示例代码。

API	操作描述
图片审核	对对象存储（Cloud Object Storage，COS）存量数据进行涉黄、违法违规以及广告引导类图片的扫描

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

存量图片审核

功能说明

下面示例展示了如何在对已存储在 COS 的图片进行审核，实现对 COS 存量数据的涉黄、违法违规以及广告引导类图片的扫描。

示例代码

```
using COSXML.Model.CI;
using COSXML.Auth;
using COSXML;
namespace COSSnippet
{
public class PictureOperationModel {
private CosXml cosXml;
PictureOperationModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 图片审核
public void SensitiveContentRecognition()
{
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000";
string key = "exampleobject"; //对象键
//.cssg-snippet-body-start:[sensitive-content-recognition]
SensitiveContentRecognitionRequest request =
new SensitiveContentRecognitionRequest(bucket, key, "ads");
SensitiveContentRecognitionResult result = cosXml.SensitiveContentRecognition(request);
}

static void Main(string[] args)
{
PictureOperationModel m = new PictureOperationModel();
/// 图片审核
m.SensitiveContentRecognition();
}
}
}
```

说明：
更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

视频审核

最近更新时间：2022-12-20 14:15:16

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于视频审核的 API 概览和 SDK 示例代码。

API	操作描述
提交视频审核任务	用于提交一个视频审核任务
查询视频审核任务结果	用于查询指定的视频审核任务

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

提交视频审核任务

功能说明

视频审核功能为异步任务方式，您可以通过提交视频审核任务审核您的视频文件，然后通过查询视频审核任务接口查询审核结果。

下面示例展示了如何提交一个视频审核任务，然后通过 JobId 查询视频审核任务结果。

说明：

- 此操作需要存储桶开通数据万象相关功能。
- 视频审核接口从 5.4.24 版本开始支持，下载新版 SDK 前往 Releases 或参见 快速入门。
- 查看版本 Changelog 请前往 GitHub。

示例代码

```
using COSXML.Model.CI;
using COSXML.Auth;
using System;
using System.Threading;
using COSXML;
namespace COSSnippet
{
public class SubmitVideoCensorJobModel {
private CosXml cosXml;
SubmitVideoCensorJobModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencen
t.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://
console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 http
s://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 提交视频审核任务
public string SubmitVideoCensorJob()
{
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000"; // 注意：此操作需要 bucket 开通内容审核相
关功能
SubmitVideoCensorJobRequest request = new SubmitVideoCensorJobRequest(bucket);
request.SetCensorObject("video.mp4"); // 媒体文件的对象键, 需要替换成桶内存在的媒体文件的
对象键
```

```
// 审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads
request.SetDetectType("Porn,Ads");
// 视频画面的审核通过视频截帧能力截取出一定量的截图，通过对截图逐一审核而实现的，该参数用于指定视频截帧的配置。
request.SetSnapshotMode("Average"); // 截帧模式。Interval 表示间隔模式；Average 表示平均模式；Fps 表示固定帧率模式。
request.SetSnapshotCount("100"); // 视频截帧数量，范围为(0, 10000]
request.SetSnapshotTimeInterval("1.0"); // 视频截帧频率，范围为(0, 60]，单位为秒，支持 float 格式，执行精度精确到毫秒
// 执行请求
SubmitCensorJobResult result = cosXml.SubmitVideoCensorJob(request);
Console.WriteLine(result.GetResultInfo());
Console.WriteLine(result.censorJobsResponse.JobsDetail.JobId);
Console.WriteLine(result.censorJobsResponse.JobsDetail.State);
Console.WriteLine(result.censorJobsResponse.JobsDetail.CreationTime);
return result.censorJobsResponse.JobsDetail.JobId;
}
static void Main(string[] args)
{
SubmitVideoCensorJobModel m = new SubmitVideoCensorJobModel();
/// 提交审核任务，JobId唯一标识了本次提交的审核任务的结果
string JobId = m.SubmitVideoCensorJob();
/// 打印 JobId
Console.WriteLine("JobId : " + JobId);
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

查询视频审核任务结果

功能说明

用于查询视频审核任务执行情况和结果。

示例代码

```
using COSXML.Model.CI;
using COSXML.Auth;
```

```
using System;
using System.Threading;
using COSXML;
namespace COSSnippet
{
public class SubmitVideoCensorJobModel {
private CosXml cosXml;
SubmitVideoCensorJobModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的区域, COS 地域的简称请参照 https://cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 查询视频审核任务结果
public void GetVideoCensorJobResult(string JobId)
{
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000"; // 注意：此操作需要 bucket 开通内容审核相关功能
GetVideoCensorJobRequest request = new GetVideoCensorJobRequest(bucket, JobId);
// 执行请求
GetVideoCensorJobResult result = cosXml.GetVideoCensorJob(request);
Console.WriteLine(result.GetResultInfo());
// 读取审核结果
Console.WriteLine(result.resultStruct.JobsDetail.JobId);
Console.WriteLine(result.resultStruct.JobsDetail.State);
Console.WriteLine(result.resultStruct.JobsDetail.CreationTime);
Console.WriteLine(result.resultStruct.JobsDetail.Object);
Console.WriteLine(result.resultStruct.JobsDetail.SnapshotCount);
Console.WriteLine(result.resultStruct.JobsDetail.Result);
Console.WriteLine(result.resultStruct.JobsDetail.PornInfo.HitFlag);
Console.WriteLine(result.resultStruct.JobsDetail.PornInfo.Count);

Console.WriteLine(result.resultStruct.JobsDetail.AdsInfo.HitFlag);
Console.WriteLine(result.resultStruct.JobsDetail.AdsInfo.Count);
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 审核视频截图内容
for(int i = 0; i < result.resultStruct.JobsDetail.Snapshot.Count; i++)
{
Console.WriteLine(result.resultStruct.JobsDetail.Snapshot[i].Url);
Console.WriteLine(result.resultStruct.JobsDetail.Snapshot[i].Text);
Console.WriteLine(result.resultStruct.JobsDetail.Snapshot[i].SnapshotTime);
Console.WriteLine(result.resultStruct.JobsDetail.Snapshot[i].PornInfo.HitFlag);
Console.WriteLine(result.resultStruct.JobsDetail.Snapshot[i].PornInfo.Score);
Console.WriteLine(result.resultStruct.JobsDetail.Snapshot[i].PornInfo.Label);
Console.WriteLine(result.resultStruct.JobsDetail.Snapshot[i].PornInfo.SubLabel);
Console.WriteLine(result.resultStruct.JobsDetail.Snapshot[i].AdsInfo.HitFlag);
Console.WriteLine(result.resultStruct.JobsDetail.Snapshot[i].AdsInfo.Score);
Console.WriteLine(result.resultStruct.JobsDetail.Snapshot[i].AdsInfo.Label);
Console.WriteLine(result.resultStruct.JobsDetail.Snapshot[i].AdsInfo.SubLabel);
}
}
static void Main(string[] args)
{
SubmitVideoCensorJobModel m = new SubmitVideoCensorJobModel();
// 输入在提交审核任务时获取的JobId
string JobId = "xxx";
/// 查询审核任务
m.GetVideoCensorJobResult(JobId);
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

音频审核

最近更新时间：2022-12-20 14:15:16

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于音频审核的 API 概览和 SDK 示例代码。

API	操作描述
提交音频审核任务	用于提交一个音频审核任务
查询音频审核任务结果	用于查询指定的音频审核任务

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

提交音频审核任务

功能说明

本接口用于提交一个音频审核任务。音频审核功能为异步任务方式，您可以通过提交音频审核任务审核您的音频文件，然后通过查询音频审核任务接口查询审核结果。

下面示例展示了如何提交一个音频审核任务，然后通过 JobId 查询音频审核任务结果。

说明：

- 此操作需要存储桶开通数据万象相关功能。
- 音频审核接口从 5.4.24 版本开始支持，下载新版 SDK 前往 Releases 或参见 快速入门。
- 查看版本 Changelog 请前往 GitHub。

示例代码

```
using COSXML.Model.CI;
using COSXML.Auth;
using System;
using System.Threading;
using COSXML;
namespace COSSnippet
{
public class SubmitAudioCensorJobModel {
private CosXml cosXml;
SubmitAudioCensorJobModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 提交音频审核任务
public string SubmitAudioCensorJob()
{
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000"; // 注意：此操作需要 bucket 开通内容审核相关功能
SubmitAudioCensorJobRequest request = new SubmitAudioCensorJobRequest(bucket);
request.SetCensorObject("audio.mp3"); // 媒体文件的对象键, 需要替换成桶内存在的媒体文件的对象键
```

```
// 审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads
request.SetDetectType("Porn,Ads");
// 执行请求
SubmitCensorJobResult result = cosXml.SubmitAudioCensorJob(request);
Console.WriteLine(result.GetResultInfo());
Console.WriteLine(result.censorJobsResponse.JobsDetail.JobId);
Console.WriteLine(result.censorJobsResponse.JobsDetail.State);
Console.WriteLine(result.censorJobsResponse.JobsDetail.CreationTime);
return result.censorJobsResponse.JobsDetail.JobId;
}
static void Main(string[] args)
{
SubmitAudioCensorJobModel m = new SubmitAudioCensorJobModel();
/// 提交审核任务
string JobId = m.SubmitAudioCensorJob();
/// 打印唯一标识此次审核任务的 JobId
Console.WriteLine("JobId : " + JobId);
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

查询音频审核任务结果

功能说明

用于查询音频审核任务执行情况和结果。

示例代码

```
using COSXML.Model.CI;
using COSXML.Auth;
using System;
using System.Threading;
using COSXML;
namespace COSSnippet
{
public class SubmitAudioCensorJobModel {
private CosXml cosXml;
```

```
SubmitAudioCensorJobModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencen
t.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://
console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 http
s://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 查询音频审核任务结果
public void GetAudioCensorJobResult(string JobId)
{
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000"; // 注意：此操作需要 bucket 开通内容审核相
关功能
GetAudioCensorJobRequest request = new GetAudioCensorJobRequest(bucket, JobId);
// 执行请求
GetAudioCensorJobResult result = cosXml.GetAudioCensorJob(request);
Console.WriteLine(result.GetResultInfo());
// 读取审核结果
Console.WriteLine(result.resultStruct.JobsDetail.JobId);
Console.WriteLine(result.resultStruct.JobsDetail.State);
Console.WriteLine(result.resultStruct.JobsDetail.CreationTime);
Console.WriteLine(result.resultStruct.JobsDetail.Object);
Console.WriteLine(result.resultStruct.JobsDetail.Result);
Console.WriteLine(result.resultStruct.JobsDetail.AudioText);
Console.WriteLine(result.resultStruct.JobsDetail.PornInfo.HitFlag);
Console.WriteLine(result.resultStruct.JobsDetail.PornInfo.Score);
Console.WriteLine(result.resultStruct.JobsDetail.PornInfo.Label);
Console.WriteLine(result.resultStruct.JobsDetail.AdsInfo.HitFlag);
Console.WriteLine(result.resultStruct.JobsDetail.AdsInfo.Score);
Console.WriteLine(result.resultStruct.JobsDetail.AdsInfo.Label);
// 音频节选Section信息
for(int i = 0; i < result.resultStruct.JobsDetail.Section.Count; i++)
{
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].Url);
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].OffsetTime);
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].Duration);
```

版权所有：腾讯云计算（北京）有限责任公司

```
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].Text);
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].PornInfo.HitFlag);
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].PornInfo.Score);
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].AdsInfo.HitFlag);
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].AdsInfo.Score);
}
}
static void Main(string[] args)
{
SubmitAudioCensorJobModel m = new SubmitAudioCensorJobModel();
/// 输入提交审核任务时获取的 JobId
string JobId = "xxx";
/// 查询审核任务结果
m.GetAudioCensorJobResult(JobId);
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

文档审核

最近更新时间：2022-12-20 14:15:16

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于文档审核的 API 概览和 SDK 示例代码。

API	操作描述
提交文档审核任务	用于提交一个文档审核任务
查询文档审核任务结果	用于查询指定的文档审核任务

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

提交文档审核任务

功能说明

本接口用于提交一个文档审核任务，可审核您的文档文件是否存在敏感违规信息。文档审核结合了对象存储（Cloud Object Storage，COS）文档预览功能，通过预先将文档转成图片，结合图片内容审核、图片 OCR 审核等方式，进行文档审核。

下面示例展示了如何提交一个文档审核任务，然后何通过 JobId 查询文档审核任务结果。

版权所有：腾讯云计算（北京）有限责任公司

说明：

- 此操作需要存储桶开通数据万象相关功能。
- 文档审核接口从 5.4.24 版本开始支持，下载新版 SDK 前往 Releases 或参见 快速入门。
- 查看版本 Changelog 请前往 GitHub。

示例代码

```
using COSXML.Model.CI;
using COSXML.Auth;
using System;
using System.Threading;
using COSXML;
namespace COSSnippet
{
public class SubmitDocumentCensorJobModel {
private CosXml cosXml;
SubmitDocumentCensorJobModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域，COS 地域的简称请参照 https://cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId，获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey，获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长，单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 提交文档审核任务
public string SubmitDocumentCensorJob()
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000"; // 注意：此操作需要 bucket 开通内容审核相关功能
SubmitDocumentCensorJobRequest request = new SubmitDocumentCensorJobRequest(bucket);
request.SetUrl("url"); // 审核文档的URL，需要替换成具体需要审核的文档URL
// 审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例
```

```
如：Porn,Ads
request.SetDetectType("Porn,Ads");
// 执行请求
SubmitCensorJobResult result = cosXml.SubmitDocumentCensorJob(request);
Console.WriteLine(result.GetResultInfo());
Console.WriteLine(result.censorJobsResponse.JobsDetail.JobId);
Console.WriteLine(result.censorJobsResponse.JobsDetail.State);
Console.WriteLine(result.censorJobsResponse.JobsDetail.CreationTime);
return result.censorJobsResponse.JobsDetail.JobId;
}
static void Main(string[] args)
{
SubmitDocumentCensorJobModel m = new SubmitDocumentCensorJobModel();
/// 提交审核任务
string JobId = m.SubmitDocumentCensorJob();
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

查询文档审核任务

功能说明

用于查询文档审核任务执行情况和结果。

示例代码

```
using COSXML.Model.CI;
using COSXML.Auth;
using System;
using System.Threading;
using COSXML;
namespace COSSnippet
{
public class SubmitDocumentCensorJobModel {
private CosXml cosXml;
SubmitDocumentCensorJobModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencen
```

```
t.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://
console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 http
s://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialPr
ovider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 查询文档审核任务结果
public void GetDocumentCensorJobResult(string JobId)
{
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.
intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000"; // 注意：此操作需要 bucket 开通内容审核相
关功能
GetDocumentCensorJobRequest request = new GetDocumentCensorJobRequest(bucket, Job
Id);
// 执行请求
GetDocumentCensorJobResult result = cosXml.GetDocumentCensorJob(request);
Console.WriteLine(result.GetResultInfo());
// 读取审核结果
Console.WriteLine(result.resultStruct.JobsDetail.State);
Console.WriteLine(result.resultStruct.JobsDetail.JobId);
Console.WriteLine(result.resultStruct.JobsDetail.Suggestion);
Console.WriteLine(result.resultStruct.JobsDetail.CreationTime);
Console.WriteLine(result.resultStruct.JobsDetail.Url);
Console.WriteLine(result.resultStruct.JobsDetail.PageCount);
Console.WriteLine(result.resultStruct.JobsDetail.Labels);
Console.WriteLine(result.resultStruct.JobsDetail.Labels.PornInfo.HitFlag);
Console.WriteLine(result.resultStruct.JobsDetail.Labels.PornInfo.Score);
Console.WriteLine(result.resultStruct.JobsDetail.PageSegment.Results.Url);
Console.WriteLine(result.resultStruct.JobsDetail.PageSegment.Results.Text);
Console.WriteLine(result.resultStruct.JobsDetail.PageSegment.Results.PageNumber);
Console.WriteLine(result.resultStruct.JobsDetail.PageSegment.Results.PornInfo.Hit
Flag);
Console.WriteLine(result.resultStruct.JobsDetail.PageSegment.Results.PornInfo.Sub
Label);
Console.WriteLine(result.resultStruct.JobsDetail.PageSegment.Results.PornInfo.Sco
re);
}
static void Main(string[] args)
```

```
{
SubmitDocumentCensorJobModel m = new SubmitDocumentCensorJobModel();
/// 输入提交审核任务时返回的，唯一标识此次审核任务的 JobId
string JobId = "xxx";
/// 查询审核任务结果
m.GetDocumentCensorJobResult(JobId);
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

文本审核

最近更新时间：2022-12-20 14:15:16

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于文本审核的 API 概览和 SDK 示例代码。

API	操作描述
提交文本审核任务	用于提交一个文本审核任务
查询文本审核任务结果	用于查询指定的文本审核任务

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

提交文本审核任务

功能说明

本接口用于提交一个文本审核任务。您可以通过提交文本审核任务审核您的文本文件，然后通过查询文本审核任务接口查询审核结果。

下面示例展示了如何提交一个文本审核任务，然后通过 JobId 查询文本审核任务结果。

说明：

- 此操作需要存储桶开通数据万象相关功能。
- 文本审核接口从 5.4.24 版本开始支持，下载新版 SDK 前往 Releases 或参见 快速入门。
- 查看版本 Changelog 请前往 GitHub。

示例代码

```
using COSXML.Model.CI;
using COSXML.Auth;
using System;
using System.Threading;
using COSXML;
namespace COSSnippet
{
public class SubmitTextCensorJobModel {
private CosXml cosXml;
SubmitTextCensorJobModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域，COS 地域的简称请参照 https://cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId，获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey，获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长，单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 提交文本审核任务
public string SubmitTextCensorJob()
{
// 存储桶名称，此处填入格式必须为 bucketname-APPID，其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000"; // 注意：此操作需要 bucket 开通内容审核相关功能
SubmitTextCensorJobRequest request = new SubmitTextCensorJobRequest(bucket);
request.SetCensorObject("text.txt"); // 媒体文件的对象键，需要替换成桶内存在的媒体文件的对象键
```

```
// 审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads
request.SetDetectType("Porn,Ads");
// 执行请求
SubmitCensorJobResult result = cosXml.SubmitTextCensorJob(request);
Console.WriteLine(result.GetResultInfo());
Console.WriteLine(result.censorJobsResponse.JobsDetail.JobId);
Console.WriteLine(result.censorJobsResponse.JobsDetail.State);
Console.WriteLine(result.censorJobsResponse.JobsDetail.CreationTime);
return result.censorJobsResponse.JobsDetail.JobId;
//.cssg-snippet-body-end
}
static void Main(string[] args)
{
SubmitTextCensorJobModel m = new SubmitTextCensorJobModel();
/// 提交审核任务
string JobId = m.SubmitTextCensorJob();
/// 打印唯一标识了此次审核任务的 JobId
Console.WriteLine("JobId : " + JobId);
}
}
}
```

说明：
更多完整示例，请前往 GitHub 查看。

查询文本审核任务

功能说明

用于查询文本审核任务执行情况和结果。

示例代码

```
using COSXML.Model.CI;
using COSXML.Auth;
using System;
using System.Threading;
using COSXML;
namespace COSSnippet
{
public class SubmitTextCensorJobModel {
```

```
private CosXml cosXml;
SubmitTextCensorJobModel() {
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照 https://cloud.tencent.com/document/product/436/6224
.Build();

string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥请参照 https://console.intl.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new DefaultQCloudCredentialProvider(secretId,
secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}
/// 查询文本审核任务结果
public void GetTextCensorJobResult(string JobId)
{
// 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考 https://console.intl.cloud.tencent.com/developer
string bucket = "examplebucket-1250000000"; // 注意：此操作需要 bucket 开通内容审核相关功能
GetTextCensorJobRequest request = new GetTextCensorJobRequest(bucket, JobId);
// 执行请求
GetTextCensorJobResult result = cosXml.GetTextCensorJob(request);
Console.WriteLine(result.GetResultInfo());
// 读取审核结果
Console.WriteLine(result.resultStruct.JobsDetail.JobId);
Console.WriteLine(result.resultStruct.JobsDetail.State);
Console.WriteLine(result.resultStruct.JobsDetail.CreationTime);
Console.WriteLine(result.resultStruct.JobsDetail.Object);
Console.WriteLine(result.resultStruct.JobsDetail.SectionCount);
Console.WriteLine(result.resultStruct.JobsDetail.Result);
Console.WriteLine(result.resultStruct.JobsDetail.PornInfo.HitFlag);
Console.WriteLine(result.resultStruct.JobsDetail.PornInfo.Count);
Console.WriteLine(result.resultStruct.JobsDetail.AdsInfo.HitFlag);
Console.WriteLine(result.resultStruct.JobsDetail.AdsInfo.Count);
// 文本节选Section信息
for(int i = 0; i < result.resultStruct.JobsDetail.Section.Count; i++)
{
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].StartByte);
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].PornInfo.HitFlag);
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].PornInfo.Score);
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].PornInfo.Keywords);
```

```
}
}
static void Main(string[] args)
{
SubmitTextCensorJobModel m = new SubmitTextCensorJobModel();
// 输入提交文本审核任务时获取到的 JobId
string JobId = "xxx";
// 查询审核任务结果
m.GetTextCensorJobResult(JobId);
}
}
}
```

说明：

更多完整示例，请前往 GitHub 查看。

设置自定义头部

最近更新时间：2020-08-25 15:37:25

简介

本文档主要介绍 SDK 如何在请求时携带自定义头部。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

功能说明

COS 在上传对象时可以携带以 `x-cos-meta-` 开头的自定义头部，包括用户自定义元数据头部后缀和用户自定义元数据信息，这些头部将作为对象元数据保存。

如果您开通了万象服务，可以携带 `Pic-Operations` 头部，实现后台自动图片处理，详细的 API 说明请参考 数据万象持久化。

示例代码

```
PutObjectRequest request = new PutObjectRequest(bucket, key, srcPath);

request.SetRequestHeader("x-cos-meta-key", "value");
request.SetRequestHeader("Content-Disposition", "attachment");

COSXMLUploadTask uploadTask = new COSXMLUploadTask(request);
uploadTask.SetSrcPath(srcPath);
```

说明：

更多完整示例，请前往 GitHub 查看。

设置访问域名（CDN/全球加速）

最近更新时间：2022-01-10 14:55:57

简介

本文档提供关于如何使用非默认域名请求 COS 服务。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

自定义源站域名

关于如何设置自定义源站域名请参考 自定义源站域名。

以下代码展示了如何使用自定义源站域名访问 COS 服务。

示例代码

```
CosXmlConfig config = new CosXmlConfig.Builder()
.SetRegion("COS_REGION") //设置一个默认的存储桶地域
//请求域名为 your.domain.com
.SetHost("your.domain.com") //自定义域名
.Build();
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

全球加速域名

关于全球加速功能请参考 全球加速功能概述。

以下代码展示了如何使用全球加速域名访问 COS 服务。

示例代码

```
CosXmlConfig config = new CosXmlConfig.Builder()
.SetEndpointSuffix("cos.accelerate.myqcloud.com")
.Build();
```

版权所有：腾讯云计算（北京）有限责任公司

说明：
更多完整示例，请前往 GitHub 查看。

自定义 CDN 域名

关于如何开启 CDN 自定义加速域名请参考 开启自定义 CDN 加速域名。

以下代码展示了如何使用自定义加速域名访问 COS 服务。

示例代码

```
CosXmlConfig config = new CosXmlConfig.Builder()
.SetHost("exampledomain.com")
.Build();
```

说明：
更多完整示例，请前往 GitHub 查看。

默认 CDN 域名

关于如何开启默认加速域名请参考 开启默认 CDN 加速域名。

以下代码展示了如何使用默认加速域名访问 COS 服务。

示例代码

```
CosXmlConfig config = new CosXmlConfig.Builder()
.SetEndpointSuffix("file.myqcloud.com")
.Build();
```

说明：
更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

异常处理

最近更新时间：2022-01-04 12:17:06

简介

调用 SDK 接口请求对象存储（Cloud Object Storage，COS）服务失败时，系统将抛出 CosClientException（客户端异常）或者 CosServerException （服务端异常）。

- CosClientException 是由于客户端无法和 COS 服务端正常进行交互所引起。例如客户端无法连接到服务端，无法解析服务端返回的数据，读取本地文件发生 IO 异常等。
- CosServerException 是客户端和 COS 服务端交互正常，但操作 COS 资源失败。例如客户端访问一个不存在 Bucket，删除一个不存在的文件，没有权限进行某个操作等。

客户端异常

CosClientException 集成自 System.ApplicationException，使用方法同 System.ApplicationException，同时添加一个额外的成员 errorCode，如下：

成员	描述	类型
errorCode	客户端错误码，例如10000表示参数检验失败，详情请参见 SDK 错误码	int

服务端异常

CosServerException 包含了服务端返回的状态码、requestid 和出错明细等。捕获异常后，建议对整个异常进行打印，异常包含了必须的排查因素。以下是异常成员变量的描述：

成员	描述	类型
requestId	请求 ID，用于表示一个请求，对于排查问题十分重要	string
statusCode	response 的 status 状态码，4xx 是指请求因客户端而失败，5xx 是服务端异常导致的失败，详情请参见 COS 错误码	string
errorCode	请求失败时 body 返回的 Error Code，详情请参见 COS 错误码	string
errorMessage	请求失败时 body 返回的 Error Message，详情请参见 COS 错误码	string

版权所有：腾讯云计算（北京）有限责任公司

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。

   COS 自助诊断工具

   点击自助诊断

   输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**。
4. 稍候片刻，便能看到相应的智能诊断结果。

版权所有：腾讯云计算（北京）有限责任公司

向下兼容指南

最近更新时间：2021-11-10 12:32:38

描述

.NET SDK 从 5.4.23 版本开始，提供 .NET Framework 4.0 及以下版本的兼容性。具体表现为，增加 COSXML-Compatible.dll 库的方式，能够实现在 .NET Framework 4.0 及以下版本应用中运行。

注意：

- 兼容的环境可能不支持 NuGet 方式集成，具体集成方式见“添加引用”。
- 所有高级接口（高级上传、断点续传下载等）在 COSXML-Compatible.dll 均不可用。

添加引用

目前仅提供手动添加 .dll 文件的集成方法，集成方式如下：

1. Releases 中下载 COSXML-Compatible.dll 文件。
2. 在 Visual Studio 项目中选择**项目** > **添加引用** > **浏览** > **COSXML-Compatible.dll** 的方式添加 .NET SDK。

使用方式

使用方式与通用版 .NET SDK 基本相同，需要关注如下两点：

1. .NET Framework 4.0 及以下环境通过 HTTPS 访问 COS 可能存在问题，如因设置开启 HTTPS 导致请求失败，可关闭 HTTPS 后重试。
2. .NET 兼容版中不包含高级接口的支持，请使用其他接口访问 COS，具体不兼容接口如下表所示：

命名空间	类名	说明
COSXML.Transfer	COSXMLCopyTask	高级 Copy 接口
COSXML.Transfer	COSXMLDownloadTask	高级下载接口
COSXML.Transfer	COSXMLTask	高级接口抽象类

版权所有：腾讯云计算（北京）有限责任公司

命名空间	类名	说明
COSXML.Transfer	COSXMLUploadTask	高级上传接口
COSXML.Transfer	TransferManager	传输任务控制类

未知问题

目前兼容包处于预发布阶段，如存在其他未列出的使用问题，请向 GitHub 项目提 issue。

Flutter SDK 快速入门

最近更新时间：2023-04-06 16:51:16

相关资源

- SDK 源码下载
- 示例 Demo
- SDK 更新日志
- Pub 地址

准备工作

1. 您需要一个纯 Flutter 项目或 Flutter 原生混合项目，这个应用可以是您现有的工程，也可以是您新建的一个空的工程。
2. Flutter 版本要求：

```
sdk: ">=2.12.0 <3.0.0"
flutter: ">=2.5.0"
```

第一步：SDK 介绍

tencentcloud_cos_sdk_plugin 目前兼容支持 iOS、Android，是通过 Flutter Plugin 桥接原生 Android 和 iOS 的 COS SDK 实现。

第二步：集成 SDK

1. 运行此命令：

```
flutter pub add tencentcloud_cos_sdk_plugin
```

2. 这将向您的包的 pubspec.yaml 添加这样一行（并运行隐式 flutter pub get）

```
dependencies:
tencentcloud_cos_sdk_plugin: ^1.0.2
```

3. 在您的 Dart 代码中，您可以使用 import 进行导入，然后开始使用：

```
import 'package:tencentcloud_cos_sdk_plugin/cos.dart';
```

第三步：开始使用

注意：

- 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

1. 初始化密钥

实现获取临时密钥

实现一个 `IFetchCredentials` 的类，实现请求临时密钥并返回结果的过程。

```
import 'dart:convert';
import 'dart:io';

import 'package:tencentcloud_cos_sdk_plugin/fetch_credentials.dart';
import 'package:tencentcloud_cos_sdk_plugin/pigeon.dart';

class FetchCredentials implements IFetchCredentials{
@override
Future<SessionQCloudCredentials> fetchSessionCredentials() async {
// 首先从您的临时密钥服务器获取包含了密钥信息的响应，例如：
var httpClient = HttpClient();
try {
// 临时密钥服务器 url
var stsUrl = "http://stsservice.com/sts";
var request = await httpClient.getUrl(Uri.parse(stsUrl));
var response = await request.close();
if (response.statusCode == HttpStatus.OK) {
var json = await response.transform(utf8.decoder).join();
print(json);
```

```
// 然后解析响应，获取临时密钥信息
var data = jsonDecode(json);
// 最后返回临时密钥信息对象
return SessionQCloudCredentials(
secretId: data['credentials']['tmpSecretId'],// 临时密钥 SecretId
secretKey: data['credentials']['tmpSecretKey'],// 临时密钥 SecretKey
token: data['credentials']['sessionToken'],// 临时密钥 Token
startTime: data['startTime'],//临时密钥有效起始时间，单位是秒
expiredTime: data['expiredTime']//临时密钥有效截止时间戳，单位是秒
);
} else {
throw ArgumentError();
}
} catch (exception) {
throw ArgumentError();
}
}
}
```

这里假设类名为 `FetchCredentials` 。初始化一个实例，来给 SDK 提供密钥。

```
Cos().initWithSessionCredential(FetchCredentials());
```

使用永久密钥进行本地调试

您可以使用腾讯云的永久密钥来进行开发阶段的本地调试。**由于该方式存在泄漏密钥的风险，请务必在上线前替换为临时密钥的方式。**

```
String SECRET_ID = "SECRETID"; //永久密钥 secretId
String SECRET_KEY = "SECRETKEY"; //永久密钥 secretKey

Cos().initWithPlainSecret(SECRET_ID, SECRET_KEY);
```

2. 注册 COS 服务

```
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
String region = "COS_REGION";
// 创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
CosXmlServiceConfig serviceConfig = CosXmlServiceConfig(
region: region,
isDebuggable: true,
isHttps: true,
);
// 注册默认 COS Service
```

```
Cos().registerDefaultService(serviceConfig);

// 创建 TransferConfig 对象，根据需要修改默认的配置参数
// TransferConfig 可以设置智能分块阈值 默认对大于或等于2M的文件自动进行分块上传，可以通过如下代码修改分块阈值
TransferConfig transferConfig = TransferConfig(
forceSimpleUpload: false,
enableVerification: true,
divisionForUpload: 2097152, // 设置大于等于 2M 的文件进行分块上传
sliceSizeForUpload: 1048576, //设置默认分块大小为 1M
);
// 注册默认 COS TransferManger
Cos().registerDefaultTransferManger(serviceConfig, transferConfig);

// 也可以通过 registerService 和 registerTransferManger 注册其他实例， 用于后续调用
// 一般用 region 作为注册的 key
String newRegion = "NEW_COS_REGION";
Cos().registerService(newRegion, serviceConfig..region = newRegion);
Cos().registerTransferManger(newRegion, serviceConfig..region = newRegion, transferConfig);
```

参数说明

CosXmlServiceConfig 用于配置 COS 服务，其主要成员说明如下：

参数名称	描述	类型	默认值	支持平台
region	存储桶地域 地域和访问域名	String	null	Android 和iOS
connectionTimeout	连接超时时间（单位是毫秒）	Int	Android(15000) iOS(30000)	Android 和iOS
socketTimeout	读写超时时间（单位是毫秒）	Int	30000	Android
isHttps	是否使用https协议	Bool	true	Android 和iOS
host	设置除了 GetService 请求外的 host	String	null	Android 和iOS

参数名称	描述	类型	默认值	支持平台
hostFormat	设置 host 的格式化字符串，sdk 会将 ${bucket} 替换为真正的 bucket，${region} 替换为真正的 region 例如将 hostFormat 设置为 ${bucket}.${region}.tencent.com，并且您的存储桶和地域分别为 bucket-1250000000 和 ap-shanghai，那么最终的请求地址为 `bucket-1250000000.ap-shanghai.tencent.com` • 注意，这个设置不会影响 GetService 请求	String	null	Android
port	设置请求的端口	Int	null	Android
isDebuggable	是否是 debug 模式（debug 模式会打印 debug 日志）	Bool	false	Android
signInUrl	是否将签名放在 URL 中，默认放在 Header 中	Bool	false	Android
userAgent	ua 拓展参数	String	null	Android 和iOS
dnsCache	是否开启 DNS 解析缓存，开启后，将 DNS 解析的结果缓存在本地，当系统 DNS 解析失败后，会使用本地缓存的 DNS 结果	Bool	true	Android
accelerate	是否使用全球加速域名	Bool	false	Android 和iOS

TransferConfig 用于配置 COS 上传服务，其主要成员说明如下：

参数名称	描述	类型	默认值	支持平台
divisionForUpload	设置启用分块上传的最小对象大小	Int	2097152	Android和iOS
sliceSizeForUpload	设置分块上传时的分块大小	Int	1048576	Android和iOS
enableVerification	分片上传时是否整体校验	Bool	true	Android和iOS
forceSimpleUpload	是否强制使用简单上传	Bool	false	Android

版权所有：腾讯云计算（北京）有限责任公司

第四步：访问 COS 服务

上传对象

SDK 支持上传本地文件、二进制数据。下面以上传本地文件为例：

```
// 获取 TransferManager
CosTransferManger transferManager = Cos().getDefaultTransferManger();
//CosTransferManger transferManager = Cos().getTransferManger("newRegion");
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String srcPath = "本地文件的绝对路径"; //本地文件的绝对路径
//若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传；否则，赋值 null
String? _uploadId;

// 上传成功回调
successCallBack(result) {
// todo 上传成功后的逻辑
}
//上传失败回调
failCallBack(clientException, serviceException) {
// todo 上传失败后的逻辑
if (clientException != null) {
print(clientException);
}
if (serviceException != null) {
print(serviceException);
}
}
//上传状态回调，可以查看任务过程
stateCallback(state) {
// todo notify transfer state
}
//上传进度回调
progressCallBack(complete, target) {
// todo Do something to update progress...
}
//初始化分块完成回调
initMultipleUploadCallback(
String bucket, String cosKey, String uploadId) {
//用于下次续传上传的 uploadId
_uploadId = uploadId;
}
//开始上传
```

```
TransferTask transferTask = await transferManager.upload(bucket, cosPath,
filePath: srcPath,
uploadId: _uploadId,
resultListener: ResultListener(successCallBack, failCallBack),
stateCallback: stateCallback,
progressCallBack: progressCallBack,
initMultipleUploadCallback: initMultipleUploadCallback
);
//暂停任务
transferTask.pause();
//恢复任务
transferTask.resume();
//取消任务
transferTask.cancel();
```

下载对象

```
// 高级下载接口支持断点续传，所以会在下载前先发起 HEAD 请求获取文件信息。
// 如果您使用的是临时密钥或者使用子账号访问，请确保权限列表中包含 HeadObject 的权限。

// TransferManager 支持断点下载，您只需要保证 bucket、cosPath、savePath
// 参数一致，SDK 便会从上次已经下载的位置继续下载。

// 获取 TransferManager
CosTransferManger transferManager = Cos().getDefaultTransferManger();
//CosTransferManger transferManager = Cos().getTransferManger("newRegion");
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String downliadPath = "本地文件的绝对路径"; //保存到本地文件的绝对路径

// 下载成功回调
successCallBack(result) {
// todo 下载成功后的逻辑
}
//下载失败回调
failCallBack(clientException, serviceException) {
// todo 下载失败后的逻辑
if (clientException != null) {
print(clientException);
}
if (serviceException != null) {
print(serviceException);
}
}
```

```
//下载状态回调, 可以查看任务过程
stateCallback(state) {
// todo notify transfer state
}
//下载进度回调
progressCallBack(complete, target) {
// todo Do something to download progress...
}
//开始下载
TransferTask transferTask = await transferManager.download(bucket, cosPath, downl
iadPath,
resultListener: ResultListener(successCallBack, failCallBack),
stateCallback: stateCallback,
progressCallBack: progressCallBack
);
//暂停任务
transferTask.pause();
//恢复任务
transferTask.resume();
//取消任务
transferTask.cancel();
```

Flutter SDK 常见问题

最近更新时间：2023-03-21 14:21:52

客户端网络正常，但是通过 HTTP 访问 COS 非常慢，或者报错 Connection reset，该如何处理？

部分区域的运营商可能会对 COS 的域名进行劫持，因此尽量通过 HTTPS 来访问 COS。

上传进度到了 100%，最终还是回调了 failCallBack 接口，该如何处理？

上传进度这里只是代表 SDK 写入到网络中的进度，100% 并不表示上传完成，只有回调 successCallBack 接口才真正上传成功，如果在最后发送 Complete Multipart Upload 请求时产生了异常，那么会回调 failCallBack 接口。

通过 CosTransferManger 上传和下载报错权限问题，该如何处理？

CosTransferManger 下载文件时会先进行 Head 操作，因此需要同时授权 HeadObject 和 GetObject 两个权限，上传时则需要简单上传和分块上传所有接口的权限。

调用接口时，报错 lock timeout 或者 no credential for sign，或者签名过期等错误，该如何处理？

如果使用的是临时密钥的方式，请检查IFetchCredentials的实现类 密钥是否及时更新了，或者密钥是否有效，如果是临时密钥则需要带上 token。

上传时报错 calculate md5 error，该如何处理？

可能是您在上传的过程中修改了文件，导致文件的 MD5 值发生了变化，或者网络很差导致服务端收包产生错误。

请求返回 ServerError 错误，该如何处理？

可能是您通过代理访问 COS，但是代理没有做好转发，直接返回了不正确的回包，导致 SDK 解析错误，可以抓包查看客户端接收的回包是否正常。

调用接口报错 403 权限错误，该如何处理？

权限问题一般不是 SDK 的问题，请检查自己授权信息。您也可以 联系我们 处理。

版权所有：腾讯云计算（北京）有限责任公司

快速体验

最近更新时间：2023-03-21 14:21:52

背景

移动互联网时代，App 作为移动互联网服务的基础设施，往往需要上传和下载大量的数据，数据的安全性和可靠性尤为重要。现在开发者可以将数据存储相关的问题交给 腾讯云对象存储（Cloud Object Storage，COS）服务，而只需要关心自己应用的业务逻辑即可，可减少很多工作量，提升开发效率。本文主要介绍如何快速搭建一个基于 COS 的应用传输服务，在腾讯云 COS 上实现应用数据的上传下载，在您的服务器上只需要部署您自己的业务、生成和管理临时密钥。

腾讯云 COS 提供的体验 demo，您可以参照本文档来体验 COS 传输实践 demo。

相关资源

本文涉及的 Demo 都存放在 Github 仓库，用户可前往获取。

准备工作

- 系统版本：Android 4.1 及以上或 iOS 11 及以上。
- 临时密钥服务器或者永久密钥（SecretId、SecretKey 可在云 API 密钥 中获取）
- 腾讯云 APPID（如果需要体验添加 bucket 功能的话）。

搭建用户客户端

配置客户端

1. 在 Github 仓库 下载项目文件，然后用 IDE 打开。
2. 根据 example/lib/config/config.template 文件中指引创建 config.dart 并修改相关值。
3. 运行项目，体验 COS 传输实践 demo。

注意：

- 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。

版权所有：腾讯云计算（北京）有限责任公司

- 如果您一定要使用永久密钥，SecretId、SecretKey 明文不要暴露到不安全环境下，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

运行示例 Demo

查询存储桶列表

启动示例 App 后，将展示用户已创建的所有存储桶。

创建存储桶

点击右上角**加号（新建桶）**，在配置页面输入桶名称并选择存储桶的所属 地域。

查询对象列表

选择点击某个存储桶，将看到该存储桶内所有文件以及文件夹。

上传对象

在文件列表页面点击右上角的**上传**，然后选择文件进行上传。

下载对象

在对象列表中，点击文件下方的**下载**按钮，即可下载文件。

版权所有：腾讯云计算（北京）有限责任公司

存储桶操作

最近更新时间：2023-03-21 14:21:52

简介

本文档提供关于存储桶基本操作的相关 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Service（List Buckets）	查询存储桶列表	查询指定账号下所有的存储桶列表
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

查询存储桶列表

功能说明

用于查询指定账号下所有存储桶列表。

示例代码

```
try {
ListAllMyBuckets listAllMyBuckets = await Cos().getDefaultService().getService();
// 存储桶列表详情请查看 ListAllMyBuckets 类
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
print(e);
}
```

参数说明

无

返回结果说明

- 成功：返回 ListAllMyBuckets 包含：存储桶列表和存储桶持有者信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientException 或者 CosXmlServiceException。详情请参见 异常处理。

ListAllMyBuckets 响应包体具体数据内容如下：

参数名称	描述	类型
buckets	存储桶列表	List<Bucket>
owner	存储桶持有者信息	Owner

存储桶（Bucket）中包含如下内容：

参数名称	描述	类型
name	存储桶的名称	String
location	存储桶所在地域	String
createDate	存储桶的创建时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	String

存储桶持有者信息（Owner）中包含如下内容：

参数名称	描述	类型
id	完整 ID	String
disPlayName	存储桶持有者的名字	String

创建存储桶

功能说明

创建一个存储桶（PUT Bucket）。

示例代码

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
String region = "COS_REGION";
// 是否开启多 AZ
bool enableMAZ = false;
try {
await Cos().getDefaultService().putBucket(
bucket,
```

```
region: region,
enableMAZ: enableMAZ
);
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
print(e);
}
```

参数说明

参数名称	描述	类型	是否必选
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
enableMAZ	是否创建多 AZ 存储桶	String	否

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientException 或者 CosXmlServiceException。详情请参见 异常处理。

检索存储桶及其权限

功能说明

HEAD Bucket 请求可以确认该存储桶是否存在，是否有权限访问。有以下几种情况：

- 存储桶存在且有读取权限，返回 HTTP 状态码为200。
- 无存储桶读取权限，返回 HTTP 状态码为403。
- 存储桶不存在，返回 HTTP 状态码为404。

示例代码

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
String region = "COS_REGION";
try {
Map<String?, String?> header = await Cos().getDefaultService().headBucket(
bucket,
region: region
```

```
);
// HTTP 状态码为200, HTTP 头为 header
} catch (e) {
// e.statusCode 查看具体的 HTTP 状态码
print(e);
}
```

参数说明

参数名称	描述	类型
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String

返回结果说明

- 成功：返回 HTTP Header。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientException 或者 CosXmlServiceException。详情请参见 异常处理。

删除存储桶

功能说明

删除指定的存储桶（DELETE Bucket）。

> 注意：
> 删除存储桶前，请确保存储桶内的数据和未完成上传的分块数据已全部清空，否则会无法删除存储桶。

示例代码

```
// 存储桶名称, 由 bucketname-appid 组成, appid 必须填入, 可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
// 存储桶所在地域简称, 例如广州地区是 ap-guangzhou
String region = "COS_REGION";
try {
await Cos().getDefaultService().deleteBucket(
bucket,
region: region
);
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
```

```
print(e);
}
```

参数说明

参数名称	描述	类型
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientException 或者 CosXmlServiceException。详情请参见 异常处理。

对象操作
上传对象

最近更新时间：2023-03-21 14:21:52

简介

本文档提供关于对象的上传操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object	简单上传对象	上传一个对象至存储桶

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传操作
Upload Part	上传分块	分块上传对象
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

上传对象

封装了简单上传、分块上传接口，根据文件大小智能的选择上传方式，同时支持续传功能。

示例代码: 上传本地文件

```
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
String region = "COS_REGION";
// 创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
CosXmlServiceConfig serviceConfig = CosXmlServiceConfig(
region: region,
isDebuggable: true,
```

```
isHttps: true,
);
// 创建 TransferConfig 对象，根据需要修改默认的配置参数
// TransferConfig 可以设置智能分块阈值 默认对大于或等于2M的文件自动进行分块上传，可以通过如下代码修改分块阈值
TransferConfig transferConfig = TransferConfig(
forceSimpleUpload: false,
enableVerification: true,
divisionForUpload: 2097152, // 设置大于等于 2M 的文件进行分块上传
sliceSizeForUpload: 1048576, //设置默认分块大小为 1M
);
// 注册默认 COS TransferManger
await Cos().registerDefaultTransferManger(serviceConfig, transferConfig);

// 获取 TransferManager
CosTransferManger transferManager = Cos().getDefaultTransferManger();
//CosTransferManger transferManager = Cos().getTransferManger("newRegion");
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String srcPath = "本地文件的绝对路径"; //本地文件的绝对路径
//若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传；否则，赋值 null
String? _uploadId;

// 上传成功回调
successCallBack(result) {
// todo 上传成功后的逻辑
}
//上传失败回调
failCallBack(clientException, serviceException) {
// todo 上传失败后的逻辑
if (clientException != null) {
print(clientException);
}
if (serviceException != null) {
print(serviceException);
}
}
//上传状态回调，可以查看任务过程
stateCallback(state) {
// todo notify transfer state
}
//上传进度回调
progressCallBack(complete, target) {
// todo Do something to update progress...
}
```

```
//初始化分块完成回调
initMultipleUploadCallback(
String bucket, String cosKey, String uploadId) {
//用于下次续传上传的 uploadId
_uploadId = uploadId;
}
//开始上传
TransferTask transferTask = await transferManager.upload(bucket, cosPath,
filePath: srcPath,
uploadId: _uploadId,
resultListener: ResultListener(successCallBack, failCallBack),
stateCallback: stateCallback,
progressCallBack: progressCallBack,
initMultipleUploadCallback: initMultipleUploadCallback
);
//暂停任务
transferTask.pause();
//恢复任务
transferTask.resume();
//取消任务
transferTask.cancel();
```

参数说明

TransferConfig 用于配置 COS 上传服务，其主要成员说明如下：

参数名称	描述	类型	默认值	支持平台
divisionForUpload	设置启用分块上传的最小对象大小	Int	2097152	Android和iOS
sliceSizeForUpload	设置分块上传时的分块大小	Int	1048576	Android和iOS
enableVerification	分片上传时是否整体校验	Bool	true	Android和iOS
forceSimpleUpload	是否强制使用简单上传	Bool	false	Android

upload 方法参数

参数名称	描述	类型
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String

参数名称	描述	类型
cosPath	对象键 是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg	String
filePath	需要上传的本地文件路径（和 byteArr 必选一个）	String
byteArr	需要上传的字节数组（和 filePath 必选一个）	Uint8List
uploadId	若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传	String
stroageClass	存储类型	String
trafficLimit	单链接限速（限速值设置范围为 819200 - 838860800，单位为 bit/s，即100KB/s - 100MB/s）	Int
ResultListener	上传结果回调（包括成功回调和失败回调）	ResultListener
StateCallBack	上传状态回调	StateCallBack
ProgressCallBack	上传进度回调	ProgressCallBack
InitMultipleUploadCallback	初始化分块完成回调	InitMultipleUploadCallbac

返回结果说明

- 成功：返回 TransferTask，可以对 TransferTask 进行暂停、恢复、取消操作。
- 失败：发生错误（如身份认证失败），会在 ResultListener 的失败回调中抛出异常 CosXmlClientException 或者 CosXmlServiceException。详情请参见 异常处理。

下载对象

最近更新时间：2023-03-21 14:21:52

简介

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地
HEAD Object	查询对象元数据	查询对象的元数据信息

下载对象

下载接口支持暂停、恢复以及取消下载请求，同时支持断点下载功能。

示例代码: 下载对象

```
// 高级下载接口支持断点续传，所以会在下载前先发起 HEAD 请求获取文件信息。
// 如果您使用的是临时密钥或者使用子账号访问，请确保权限列表中包含 HeadObject 的权限。

// TransferManager 支持断点下载，您只需要保证 bucket、cosPath、savePath
// 参数一致，SDK 便会从上次已经下载的位置继续下载。

// 获取 TransferManager
CosTransferManger transferManager = Cos().getDefaultTransferManger();
//CosTransferManger transferManager = Cos().getTransferManger("newRegion");
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String downliadPath = "本地文件的绝对路径"; //保存到本地文件的绝对路径

// 下载成功回调
successCallBack(result) {
// todo 下载成功后的逻辑
}
//下载失败回调
failCallBack(clientException, serviceException) {
// todo 下载失败后的逻辑
if (clientException != null) {
print(clientException);
```

```
}
if (serviceException != null) {
print(serviceException);
}
}
//下载状态回调，可以查看任务过程
stateCallback(state) {
// todo notify transfer state
}
//下载进度回调
progressCallBack(complete, target) {
// todo Do something to download progress...
}
//开始下载
TransferTask transferTask = await transferManager.download(bucket, cosPath, downl
iadPath,
resultListener: ResultListener(successCallBack, failCallBack),
stateCallback: stateCallback,
progressCallBack: progressCallBack
);
//暂停任务
transferTask.pause();
//恢复任务
transferTask.resume();
//取消任务
transferTask.cancel();
```

参数说明

参数名称	描述	类型	是否必选
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
cosPath	对象键 是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg	String	是
savePath	下载保存到本地文件的绝对路径	String	是
versionId	指定要下载的版本 ID	String	否
trafficLimit	单链接限速（限速值设置范围为 819200 - 838860800，单位为 bit/s，即100KB/s - 100MB/s）	Int	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型	是否必选
ResultListener	下载结果回调（包括成功回调和失败回调）	ResultListener	否
StateCallBack	下载状态回调	StateCallBack	否
ProgressCallBack	下载进度回调	ProgressCallBack	否

返回结果说明

- 成功：返回 TransferTask，可以对 TransferTask 进行暂停、恢复、取消操作。
- 失败：发生错误（如身份认证失败），会在 ResultListener 的失败回调中抛出异常 CosXmlClientException 或者 CosXmlServiceException。详情请参见 异常处理。

列出对象

最近更新时间：2023-03-21 14:21:52

简介

本文档提供关于列出对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

示例代码一: 获取第一页数据

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
try {
BucketContents bucketContents = await Cos().getDefaultService().getBucket(
bucket,
prefix: "dir/", // 前缀匹配，用来规定返回的对象前缀地址
maxKeys: 100 // 单次返回最大的条目数量，默认1000
);
// 表示数据被截断，需要拉取下一页数据
var isTruncated = bucketContents.isTruncated;
// nextMarker 表示下一页的起始位置
var nextMarker = bucketContents.nextMarker;
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
print(e);
}
```

示例代码二：请求下一页数据

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
```

版权所有：腾讯云计算（北京）有限责任公司

```
String bucket = "examplebucket-1250000000";
// prevPageBucketContents 是上一页的返回结果，这里的 nextMarker 表示下一页的起始位置
String prevPageMarker = prevPageBucketContents.nextMarker;
try {
BucketContents bucketContents = await Cos().getDefaultService().getBucket(
bucket,
prefix: "dir/", // 前缀匹配，用来规定返回的对象前缀地址
marker: prevPageMarker, // 起始位置
maxKeys: 100 // 单次返回最大的条目数量，默认1000
);
// 表示数据被截断，需要拉取下一页数据
var isTruncated = bucketContents.isTruncated;
// nextMarker 表示下一页的起始位置
var nextMarker = bucketContents.nextMarker;
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
print(e);
}
```

示例代码三：获取对象列表与子目录

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
// 定界符为一个符号，如果有 Prefix，
// 则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，
// 然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始
var delimiter = "/";
try {
BucketContents bucketContents = await Cos().getDefaultService().getBucket(
bucket,
prefix: "dir/", // 前缀匹配，用来规定返回的对象前缀地址
delimiter: delimiter,
maxKeys: 100 // 单次返回最大的条目数量，默认1000
);
// 表示数据被截断，需要拉取下一页数据
var isTruncated = bucketContents.isTruncated;
// nextMarker 表示下一页的起始位置
var nextMarker = bucketContents.nextMarker;
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
print(e);
}
```

参数说明

参数名称	描述	类型	是否必选
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
prefix	对象键匹配前缀，限定响应中只包含指定前缀的对象键	String	否
delimiter	一个字符的分隔符，用于对对象键进行分组。所有对象键中从 prefix 或从头（如未指定 prefix）到首个 delimiter 之间相同的部分将作为 CommonPrefixes 下的一个 Prefix 节点。被分组的对象键不再出现在后续对象列表中，具体场景和用法可参考上面的实际案例	String	否
encodingType	规定返回值的编码方式，可选值：url，代表返回的对象键为 URL 编码（百分号编码）后的值，例如“腾讯云”将被编码为%E8%85%BE%E8%AE%AF%E4%BA%91	String	否
marker	起始对象键标记，从该标记之后（不含）按照 UTF-8 字典序返回对象键条目	String	否
maxKeys	单次返回最大的条目数量，默认值为1000，最大为1000 注意：该参数会限制每一次 List 操作返回的最大条目数，COS 在每次 List 操作中将返回不超过 max-keys 所设定数值的条目（即 CommonPrefixes 和 Contents 的总和），如果单次响应中未列出所有对象，COS 会返回 NextMarker 节点，其值作为您下次 List 请求的 marker 参数，以便您列出后续对象	Int	否

返回结果说明

- 成功：返回 BucketContents 包含：对象列表和列表信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientException 或者 CosXmlServiceException。详情请参见 异常处理。

BucketContents 响应包体具体数据内容如下：

参数名称	描述	类型
name	存储桶的名称，格式为<BucketName-APPID>，例如 examplebucket-1250000000	String
encodingType	编码格式，对应请求中的 encodingType 参数，且仅当请求中指定了 encodingType 参数才会返回该节点	String
prefix	对象键匹配前缀，对应请求中的 prefix 参数	String
marker	起始对象键标记，从该标记之后（不含）按照 UTF-8 字典序返回对象键条目，对应请求中的 marker 参数	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
maxKeys	单次响应返回结果的最大条目数量，对应请求中的 maxKeys 参数	Int
delimiter	分隔符，对应请求中的 delimiter 参数，且仅当请求中指定了 delimiter 参数才会返回该节点	String
isTruncated	响应条目是否被截断，布尔值，例如 true 或 false	Bool
nextMarker	仅当响应条目有截断（IsTruncated 为 true）才会返回该节点，该节点的值为当前响应条目中的最后一个对象键，当需要继续请求后续条目时，将该节点的值作为下一次请求的 marker 参数传入	String
contentsList	对象条目	List<Content>
commonPrefixesList	从 prefix 或从头（如未指定 prefix）到首个 delimiter 之间相同的部分，定义为 Common Prefix。仅当请求中指定了 delimiter 参数才有可能返回该节点	List<CommonPrefixes>

对象（Content）中包含如下内容：

参数名称	描述	类型
key	对象键	String
lastModified	对象最后修改时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	String
eTag	对象的实体标签（Entity Tag），是对象被创建时标识对象内容的信息标签，可用于检查对象的内容是否发生变化，例如 `“8e0b617ca298a564c3331da28dcb50df”`，此头部并不一定返回对象的 MD5 值，而是根据对象上传和加密方式而有所不同	String
size	对象大小，单位为 Byte	Int
owner	对象持有者信息	Owner
storageClass	对象存储类型。枚举值请参见 存储类型 文档，例如 STANDARD_IA，ARCHIVE	String

持有者信息（Owner） 中包含如下内容：

参数名称	描述	类型
id	对象持有者的 APPID	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
disPlayName	对象持有者的名称	String

CommonPrefixes 中包含如下内容：

参数名称	描述	类型
prefix	单条 Common Prefix 的前缀	String

删除对象

最近更新时间：2023-03-21 14:21:52

简介

本文档提供关于对象的删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象

删除单个对象

功能说明

删除指定的对象（DELETE Object）。

示例代码

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
//对象在存储桶中的位置标识符，即对象键
String cosPath = "exampleobject";
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
String region = "COS_REGION";
try {
await Cos().getDefaultService().deleteObject(
bucket, cosPath,
region: region
);
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
print(e);
}
```

参数说明

参数名称	描述	类型	是否必选

参数名称	描述	类型	是否必选
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
cosPath	对象键 是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg	String	是
versionId	指定要删除的版本 ID	String	否

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientException 或者 CosXmlServiceException。详情请参见 异常处理。

单链接限速

最近更新时间：2023-03-21 14:21:52

简介

本文档提供关于调用上传下载接口时对链接进行限速。

使用说明

限速值设置范围为 **819200 - 838860800**，单位默认为 bit/s，即100KB/s - 100MB/s，如果超出该范围将返回400错误。
通过调用 upload 和 download 方法时传入的 trafficLimit 参数进行限速。

示例代码一：上传时对单链接限速

```
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
String region = "COS_REGION";
// 创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
CosXmlServiceConfig serviceConfig = CosXmlServiceConfig(
region: region,
isDebuggable: true,
isHttps: true,
);
// 创建 TransferConfig 对象，根据需要修改默认的配置参数
// TransferConfig 可以设置智能分块阈值 默认对大于或等于2M的文件自动进行分块上传，可以通过如下代码修改分块阈值
TransferConfig transferConfig = TransferConfig(
forceSimpleUpload: false,
enableVerification: true,
divisionForUpload: 2097152, // 设置大于等于 2M 的文件进行分块上传
sliceSizeForUpload: 1048576, //设置默认分块大小为 1M
);
// 注册默认 COS TransferManger
await Cos().registerDefaultTransferManger(serviceConfig, transferConfig);

// 获取 TransferManager
CosTransferManger transferManager = Cos().getDefaultTransferManger();
//CosTransferManger transferManager = Cos().getTransferManger("newRegion");
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
```

```
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String srcPath = "本地文件的绝对路径"; //本地文件的绝对路径
//若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传；否则，赋值 null
String? _uploadId;

// 上传成功回调
successCallBack(result) {
// todo 上传成功后的逻辑
}
//上传失败回调
failCallBack(clientException, serviceException) {
// todo 上传失败后的逻辑
if (clientException != null) {
print(clientException);
}
if (serviceException != null) {
print(serviceException);
}
}
//上传状态回调，可以查看任务过程
stateCallback(state) {
// todo notify transfer state
}
//上传进度回调
progressCallBack(complete, target) {
// todo Do something to update progress...
}
//初始化分块完成回调
initMultipleUploadCallback(
String bucket, String cosKey, String uploadId) {
//用于下次续传上传的 uploadId
_uploadId = uploadId;
}
// 设置单链接限速，单位为 bit/s，示例设置为 1M/s
int _trafficLimit = 1024 * 1024 * 8;
//开始上传
TransferTask transferTask = await transferManager.upload(bucket, cosPath,
filePath: srcPath,
uploadId: _uploadId,
trafficLimit: _trafficLimit,
resultListener: ResultListener(successCallBack, failCallBack),
stateCallback: stateCallback,
progressCallBack: progressCallBack,
initMultipleUploadCallback: initMultipleUploadCallback
);
```

示例代码二：下载时对单链接限速

```
// 高级下载接口支持断点续传，所以会在下载前先发起 HEAD 请求获取文件信息。
// 如果您使用的是临时密钥或者使用子账号访问，请确保权限列表中包含 HeadObject 的权限。

// TransferManager 支持断点下载，您只需要保证 bucket、cosPath、savePath
// 参数一致，SDK 便会从上次已经下载的位置继续下载。

// 获取 TransferManager
CosTransferManger transferManager = Cos().getDefaultTransferManger();
//CosTransferManger transferManager = Cos().getTransferManger("newRegion");
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String downliadPath = "本地文件的绝对路径"; //保存到本地文件的绝对路径

// 下载成功回调
successCallBack(result) {
// todo 下载成功后的逻辑
}
//下载失败回调
failCallBack(clientException, serviceException) {
// todo 下载失败后的逻辑
if (clientException != null) {
print(clientException);
}
if (serviceException != null) {
print(serviceException);
}
}
//下载状态回调，可以查看任务过程
stateCallback(state) {
// todo notify transfer state
}
//下载进度回调
progressCallBack(complete, target) {
// todo Do something to download progress...
}
// 设置单链接限速，单位为 bit/s，示例设置为 1M/s
int _trafficLimit = 1024 * 1024 * 8;
//开始下载
TransferTask transferTask = await transferManager.download(bucket, cosPath, downl
iadPath,
trafficLimit: _trafficLimit,
resultListener: ResultListener(successCallBack, failCallBack),
stateCallback: stateCallback,
```

```
progressCallBack: progressCallBack
);
//暂停任务
transferTask.pause();
//恢复任务
transferTask.resume();
//取消任务
transferTask.cancel();
```

版权所有：腾讯云计算（北京）有限责任公司

生成预签名链接

最近更新时间：2023-06-16 15:35:53

简介

本文档提供关于生成对象预签名链接相关的 SDK 示例代码。

关于使用预签名 URL 上传的说明请参见 预签名授权上传， 使用预签名 URL 下载的说明请参见 预签名授权下载。

说明：

- 签名规则。请参考 请求签名。
- 建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄露目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。

生成对象下载预签名链接

功能说明

生成用于 COS 对象下载的预签名链接。

示例代码

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
//对象在存储桶中的位置标识符，即对象键
String cosPath = "exampleobject";
HashMap<String, String> parameters = HashMap();
parameters["test1k"] = "test1v";
parameters["test2k"] = "test2v";
try {
// 获取对象下载预签名链接
String objectUrl = await Cos().getDefaultService().getPresignedUrl(
bucket,
cosPath,
parameters: parameters
```

```
);
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
print(e);
}
```

参数说明

参数名称	描述	类型	是否必选
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
cosPath	对象键 是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg	String	是
signValidTime	设置签名有效期(单位为秒)，注意这里是签名有效期，您需要自行保证密钥有效期	Int	否
signHost	是否签入Header Host，不签可能导致请求失败或安全漏洞	Bool	否
parameters	http请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此HTTP请求的参数	Map	否

返回结果说明

- 成功：返回对象下载预签名链接。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientException 或者 CosXmlServiceException。详情请参见 异常处理。

异常处理

最近更新时间：2023-03-21 14:21:52

简介

调用 SDK 接口请求 COS 服务失败时，系统将抛出 CosXmlClientException（客户端异常）或者 CosXmlServiceException （服务端异常）。

- CosXmlClientException 是由于客户端无法和 COS 服务端正常进行交互所引起。如客户端无法连接到服务端，无法解析服务端返回的数据，读取本地文件发生 IO 异常等。
- CosXmlServiceException 是客户端和 COS 服务端交互正常，但操作 COS 资源失败。如客户端访问一个不存在 Bucket ，删除一个不存在的文件，没有权限进行某个操作等。

客户端异常

CosXmlClientException 例如本地文件发生 IO 异常等。CosXmlClientException 包含了定义的客户端错误码和具体的错误描述信息。捕获异常后，建议对整个异常进行打印，异常包含了必须的排查因素。以下是异常成员变量的描述：

成员	描述	类型
errorCode	客户端错误码，如10000表示参数检验失败，更多详情请参见 SDK 错误码	int
message	客户端错误信息	string

服务端异常

CosXmlServiceException 例如客户端访问一个不存在 Bucket，删除一个不存在的文件，没有权限进行某个操作，服务端故障异常等。CosXmlServiceException 包含了服务端返回的状态码，requestid，出错明细等。捕获异常后，建议对整个异常进行打印，异常包含了必须的排查因素。以下是异常成员变量的描述：

成员	描述	类型
requestId	请求 ID，用于表示一个请求，对于排查问题十分重要	string
statusCode	response 的 status 状态码，4xx 是指请求因客户端而失败，5xx 是服务端异常导致的失败，更多详情请参见 COS 错误信息	string

版权所有：腾讯云计算（北京）有限责任公司

成员	描述	类型
errorCode	请求失败时 body 返回的 Error Code，更多详情请参见 COS 错误信息	string
errorMessage	请求失败时 body 返回的 Error Message，更多详情请参见 COS 错误信息	string

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。
   COS 自助诊断工具
   点击自助诊断
   输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**。
4. 稍侯片刻，便能看到相应的智能诊断结果。

设置访问域名

最近更新时间：2023-03-21 14:21:52

简介

本文档提供关于如何使用非默认域名请求 COS 服务。

自定义源站域名

关于如何设置自定义源站域名请参见 自定义源站域名。

以下代码展示了如何使用自定义源站域名访问 COS 服务。

示例代码

```
// 存储桶 region 可以在 COS 控制台指定存储桶的概览页查看 https://console.intl.cloud.tencent.com/cos5/bucket/ , 关于地域的详情见 https://cloud.tencent.com/document/product/436/6224
String region = "ap-beijing"; // 您的存储桶地域
String customDomain = "exampledomain.com"; // 自定义域名

// 创建 CosXmlServiceConfig 对象, 根据需要修改默认的配置参数
CosXmlServiceConfig serviceConfig = CosXmlServiceConfig(
region: region,
isDebuggable: false,
isHttps: true,
hostFormat: customDomain // 修改请求的域名
);
// 注册默认 COS Service
Cos().registerDefaultService(serviceConfig);
```

全球加速域名

关于全球加速功能请参考 全球加速功能概述。

以下代码展示了如何使用全球加速域名访问 COS 服务。

示例代码

```
// 存储桶 region 可以在 COS 控制台指定存储桶的概览页查看 https://console.intl.cloud.tencent.com/cos5/bucket/ , 关于地域的详情见 https://cloud.tencent.com/document/product/436/6224
String region = "ap-beijing"; // 您的存储桶地域
// 是否开启全球加速
bool accelerate = true;
// 创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
CosXmlServiceConfig serviceConfig = CosXmlServiceConfig(
region: region,
isDebuggable: false,
isHttps: true,
accelerate: accelerate // 使用全球加速域名
);
// 注册默认 COS Service
Cos().registerDefaultService(serviceConfig);
```

版权所有：腾讯云计算（北京）有限责任公司

Go SDK
快速入门

最近更新时间：2023-02-08 14:38:13

下载与安装

相关资源

- 对象存储 COS 的 XML Go SDK 源码下载地址：XML Go SDK。
- 示例 Demo 下载地址：COS XML Go SDK 示例。
- 更多信息请参见 COS Go SDK API 文档。
- SDK 文档中的所有示例代码请参见 SDK 代码示例。
- SDK 更新日志请参见 ChangeLog。
- SDK 常见问题请参见：Go SDK 常见问题。

> 说明：
> 如果您在使用 SDK 时遇到函数或方法不存在等错误，请先将 SDK 升级到最新版再重试。

环境依赖

Golang：用于下载和安装 Go 编译运行环境，请前往 Golang 官网进行下载。

安装 SDK

执行以下命令安装 COS Go SDK：

```
go get -u github.com/tencentyun/cos-go-sdk-v5
```

开始使用

下面为您介绍如何使用 COS Go SDK 完成一个基础操作，如初始化客户端、创建存储桶、查询存储桶列表、上传对象、查询对象列表、下载对象和删除对象。

初始化

注意：

- 建议用户使用子账号密钥 + 环境变量的方式调用 SDK，提高 SDK 使用的安全性。为子账号授权时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

使用 COS 域名生成 COS GO 客户端 Client 结构。

方法原型

```
func NewClient(uri *BaseURL, httpClient *http.Client) *Client
```

参数说明

```
// BaseURL 访问各 API 所需的基础 URL
type BaseURL struct {
// 访问 bucket, object 相关 API 的基础 URL（不包含 path 部分）: https://examplebucket-1250000000.cos.<Region>.myqcloud.com
BucketURL *url.URL
// 访问 service API 的基础 URL（不包含 path 部分）: https://cos.<Region>.myqcloud.com
ServiceURL *url.URL
// 访问 Batch API 的基础 URL （不包含 path 部分）: https://<UIN>.cos-control.<Region>.myqcloud.com
BatchURL *url.URL
// 访问 CI 的基础 URL （不包含 path 部分）: https://examplebucket-1250000000.ci.<Region>.myqcloud.com
CIURL *url.URL
}
```

参数名称	参数描述	类型	是否必填
BucketURL	访问 bucket, object 相关 API 的基础 URL（不包含 path 部分）	string	是
ServiceURL	访问 service API 的基础 URL（不包含 path 部分）	string	否
BatchURL	访问 Batch API 的基础 URL （不包含 path 部分）	string	否
CIURL	访问 CI 的基础 URL （不包含 path 部分）	string	否

请求示例1：使用永久密钥

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为用户真实的信息
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
```

```
https://console.intl.cloud.tencent.com/cos5/bucket
// COS_REGION 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket，关于地域的详情见 https://intl.cloud.tencent.com/document/product/436/6224
u, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
// 用于 Get Service 查询，默认全地域 service.cos.myqcloud.com
su, _ := url.Parse("https://cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: u, ServiceURL: su}
// 1.永久密钥
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
},
})
```

请求示例2：使用临时密钥

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// COS_REGION 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket，关于地域的详情见 https://intl.cloud.tencent.com/document/product/436/6224
u, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
// 2.临时密钥
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
SecretID: "SECRETID",
SecretKey: "SECRETKEY",
SessionToken: "SECRETTOKEN",
},
})
if client != nil {
// 调用 COS 请求
}
```

说明：

临时密钥生成和使用可参见 临时密钥生成及使用指引。

请求示例3：设置域名

通过修改 BaseURL，可以直接使用自定义域名或者全球加速域名访问 COS。

```
// 使用全球加速域名访问COS
u, _ := url.Parse("http://<BucketName-APPID>.cos.accelerate.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
// 2.临时密钥
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
SecretID: "SECRETID",
SecretKey: "SECRETKEY",
SessionToken: "SECRETTOKEN",
},
})
```

请求示例4：CRC64校验

COS Go SDK 默认开启了文件上传 CRC64 检验。

注意：

- COS Go SDK 版本需要大于等于 v0.7.23。
- 强烈建议用户不关闭 CRC64 校验。

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
// 存储桶名称，由 bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。https://console.cloud.tencent.com/cos5/bucket
// COS_REGION 可以在控制台查看，https://console.cloud.tencent.com/cos5/bucket，关于地域的详情见 https://www.tencentcloud.com/document/product/436/6224
u, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
// 2.临时密钥
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
SecretID: "SECRETID",
SecretKey: "SECRETKEY",
SessionToken: "SECRETTOKEN",
```

版权所有：腾讯云计算（北京）有限责任公司

```
},
})
// 关闭 CRC64 校验
client.Conf.EnableCRC = false
```

创建存储桶

```
package main
import (
"context"
"net/http"
"net/url"
"os"
"github.com/tencentyun/cos-go-sdk-v5"
)
func main() {
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// COS_REGION 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket，关于地域的详情见 https://intl.cloud.tencent.com/document/product/436/6224
u, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
},
})
_, err := c.Bucket.Put(context.Background(), nil)
if err != nil {
panic(err)
}
}
```

查询存储桶列表

```
package main

import (
"context"
```

```
"fmt"
"net/http"
"os"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
c := cos.NewClient(nil, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
},
})

s, _, err := c.Service.Get(context.Background())
if err != nil {
panic(err)
}

for _, b := range s.Buckets {
fmt.Printf("%#v\n", b)
}
}
```

上传对象

```
package main

import (
"context"
"net/http"
"net/url"
"os"
"strings"
"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
```

```
// COS_REGION 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket，关于地域的详情见 https://intl.cloud.tencent.com/document/product/436/6224
u, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
},
})
// 对象键（Key）是对象在存储桶中的唯一标识。
// 例如，在对象的访问域名 `examplebucket-1250000000.cos.COS_REGION.myqcloud.com/test/objectPut.go` 中，对象键为 test/objectPut.go
name := "test/objectPut.go"
// 1.通过字符串上传对象
f := strings.NewReader("test")

_, err := c.Object.Put(context.Background(), name, f, nil)
if err != nil {
panic(err)
}
// 2.通过本地文件上传对象
_, err = c.Object.PutFromFile(context.Background(), name, "../test", nil)
if err != nil {
panic(err)
}
// 3.通过文件流上传对象
fd, err := os.Open("./test")
if err != nil {
panic(err)
}
defer fd.Close()
_, err = c.Object.Put(context.Background(), name, fd, nil)
if err != nil {
panic(err)
}
}
```

查询对象列表

```
package main
```

```
import (
"context"
"fmt"
"net/http"
"net/url"
"os"
"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// COS_REGION 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket，关于地域的详情见 https://intl.cloud.tencent.com/document/product/436/6224
u, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
},
})

opt := &cos.BucketGetOptions{
Prefix: "test",
MaxKeys: 3,
}
v, _, err := c.Bucket.Get(context.Background(), opt)
if err != nil {
panic(err)
}

for _, c := range v.Contents {
fmt.Printf("%s, %d\n", c.Key, c.Size)
}
}
```

下载对象

```
package main
```

```
import (
"context"
"fmt"
"io/ioutil"
"net/http"
"net/url"
"os"
"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// COS_REGION 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket，关于地域的详情见 https://intl.cloud.tencent.com/document/product/436/6224
u, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
},
})
// 1.通过响应体获取对象
name := "test/objectPut.go"
resp, err := c.Object.Get(context.Background(), name, nil)
if err != nil {
panic(err)
}
bs, _ := ioutil.ReadAll(resp.Body)
resp.Body.Close()
fmt.Printf("%s\n", string(bs))
// 2.获取对象到本地文件
_, err = c.Object.GetToFile(context.Background(), name, "exampleobject", nil)
if err != nil {
panic(err)
}
}
```

删除对象

```go
package main

import (
"context"
"net/http"
"net/url"
"os"
"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// COS_REGION 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket，关于地域的详情见 https://intl.cloud.tencent.com/document/product/436/6224
u, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://intl.cloud.tencent.com/document/product/598/32675
},
})
name := "test/objectPut.go"
_, err := c.Object.Delete(context.Background(), name)
if err != nil {
panic(err)
}
}
```

Go SDK 常见问题

最近更新时间：2023-02-08 15:42:53

使用 Go SDK 如何读取文件的所有属性？

使用 GET Object 接口，在 Response.Header 中获取文件属性。

Go SDK 支持上传文件流吗？

支持上传文件流或字节流，详情请参见 上传对象 文档。

Go SDK 报错“403 AccessDenied(Message: Request has expired, RequestId: xxx, TraceId: xxx)”，该如何处理？

由于签名过期导致，重新生成签名即可解决；若重新生成签名仍报相同的错误，可以再检查下机器的本地时间是否已设置为当地标准时间。

版权所有：腾讯云计算（北京）有限责任公司

存储桶操作
创建存储桶

最近更新时间：2023-05-30 17:04:28

简介

本文档提供关于创建存储桶的 API 概览以及 SDK 示例代码。

注意：

- 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄露目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

API	操作名	操作描述
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶

创建存储桶

功能说明

在指定账号下创建一个存储桶。同一用户账号下，可以创建多个存储桶，数量上限是200个（不区分地域），存储桶中的对象数量没有限制。创建存储桶是低频操作，一般建议在控制台创建 Bucket，在 SDK 进行 Object 的操作。

方法原型

```
func (s *BucketService) Put(ctx context.Context, opt *BucketPutOptions) (*Respons
e, error)
```

请求示例

```
package main

import (
"context"
```

```
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)

func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})

// case1：创建存储桶
opt := &cos.BucketPutOptions{
XCosACL: "private",
}
_, err := client.Bucket.Put(context.Background(), opt)
if err != nil {
panic(err)
}

// case2：创建 MAZ 存储桶
opt.CreateBucketConfiguration = &cos.CreateBucketConfiguration{
BucketAZConfig: "MAZ",
}
_, err = client.Bucket.Put(context.Background(), opt)
if err != nil {
panic(err)
```

```
}
}
```

参数说明

```
type BucketPutOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
CreateBucketConfiguration *CreateBucketConfiguration
}
type CreateBucketConfiguration struct {
BucketAZConfig string
}
```

参数名称	参数描述
XCosACL	设置 Bucket 的 ACL，如 private，public-read，public-read-write
XCosGrantFullControl	赋予指定账户对 Bucket 的读写权限。格式为 `id=" ",id=" "` 。当需要给子账户授权 `id="qcs::cam::uin/{OwnerUin}:uin/{SubUin}"` ，当需要给主账户授权时，`id="qcs::cam::uin/{OwnerUin}:uin/{OwnerUin}"` 。例如 `id="qcs::cam::uin/100000000001:uin/100000000011",id="qcs::cam`
XCosGrantRead	赋予指定账户对 Bucket 的读权限。格式为 `id=" ",id=" "` 。当需要给子账户授权时 `id="qcs::cam::uin/{OwnerUin}:uin/{SubUin}"` ，当需要给主账户授权时，`id="qcs::cam::uin/{OwnerUin}:uin/{OwnerUin}"` 。例如 `id="qcs::cam::uin/100000000001:uin/100000000011",id="qcs::cam`
XCosGrantWrite	赋予指定账户对 Bucket 的写权限。格式为 `id=" ",id=" "` 。当需要给子账户授权时 `id="qcs::cam::uin/{OwnerUin}:uin/{SubUin}"` ，当需要给主账户授权时，`id="qcs::cam::uin/{OwnerUin}:uin/{OwnerUin}"` 。例如 `id="qcs::cam::uin/100000000001:uin/100000000011",id="qcs::cam::`
BucketAZConfig	存储桶 AZ 配置，指定为 MAZ 以创建多 AZ 存储桶。多 AZ 存储类型的适用地域，请参见

删除存储桶

最近更新时间：2023-02-08 15:39:50

简介

本文档提供关于删除存储桶的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

删除存储桶

功能说明

删除指定账号下的空存储桶。

方法原型

```
func (s *BucketService) Delete(ctx context.Context) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
```

```
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, err := client.Bucket.Delete(context.Background())
if err != nil {
panic(err)
}
}
```

检索存储桶

最近更新时间：2023-02-08 15:39:50

简介

本文档提供关于检索存储桶的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问

检索存储桶及其权限

功能说明

检索存储桶是否存在且是否有权限访问。

方法原型

```
func (s *BucketService) Head(ctx context.Context) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
```

```
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, err := client.Bucket.Head(context.Background())
if err != nil {
panic(err)
}
}
```

查询存储桶列表

最近更新时间：2023-02-08 15:39:50

简介

本文档提供关于查询存储桶列表的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Service（List Buckets）	查询存储桶列表	查询指定账号下所有的存储桶列表

查询存储桶列表

功能说明

查询指定账号下所有的存储桶列表。

方法原型

```
func (s *ServiceService) Get(ctx context.Context) (*ServiceGetResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
```

```
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, _, err := client.Service.Get(context.Background())
if err != nil {
panic(err)
}
}
```

返回结果说明

通过 GetServiceResult 返回请求结果。

```
type ServiceGetResult struct {
Owner *Owner
Buckets []Bucket
}
type Owner struct {
ID string
DisplayName string
}
type Bucket struct {
Name string
Region string
CreationDate string
}
```

参数名称	参数描述	类型
ID	Bucket 所有者的 ID	string
DisplayName	Bucket 所有者的名字信息	string
Name	Bucket 的名称	string

参数名称	参数描述	类型
Region	Bucket 所在地域	string
CreationDate	Bucket 创建时间。ISO8601 格式，例如 2016-11-09T08:46:32.000Z	string

版权所有：腾讯云计算（北京）有限责任公司

判断存储桶是否存在

最近更新时间：2023-02-08 15:39:50

简介

本文档提供快捷查询某个存储桶是否存在的示例代码。

示例代码实际调用了 COS API HeadBucket，是该接口的简化版。

HeadBucket除了检查存储桶是否存在，还可以判断是否有权限访问存储桶，有以下几种情况：

- 存储桶存在且有读取权限，返回 HTTP 状态码为200。
- 无存储桶读取权限，返回 HTTP 状态码为403。
- 存储桶不存在，返回 HTTP 状态码为404。

检查存储桶是否存在

功能说明

您可以通过 SDK 提供的快捷接口来检查 Bucket 是否存在。

> 注意：
> COS Go SDK 版本需要大于等于 v0.7.33。

方法原型

```
func (s *BucketService) IsExist(ctx context.Context) (isExist bool, err error)
```

示例代码

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
```

```
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
ok, err := client.Bucket.IsExist(context.Background())
if err == nil && ok {
fmt.Printf("bucket exists\n")
} else if err != nil {
fmt.Printf("head bucket failed: %v\n", err)
} else {
fmt.Printf("bucket does not exist\n")
}
}
```

返回结果说明

参数名称	描述	类型
isExist	存储桶是否存在	bool
err	请求是否成功	struct

对象操作
上传对象

最近更新时间：2023-02-08 15:36:40

简介

本文档提供关于对象的上传操作的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object	简单上传对象	上传一个 Object（文件/对象）至 Bucket
APPEND Object	追加上传对象	使用分块追加的方式上传对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化 Multipart Upload 上传操作
Upload Part	上传分块	分块上传文件
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

高级接口（推荐）

上传对象

功能说明

上传接口根据用户文件的长度，自动切分数据， 降低用户的使用门槛，用户无需关心分块上传的每个步骤，当文件大小大于16MB，采用分块上传，用户可通过 PartSize 参数调整。

版权所有：腾讯云计算（北京）有限责任公司

方法原型

```
func (s *ObjectService) Upload(ctx context.Context, key string, filepath string, opt *MultiUploadOptions) (*CompleteMultipartUploadResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
key := "exampleobject"
_, _, err := client.Object.Upload(
context.Background(), key, "localfile", nil,
)
if err != nil {
panic(err)
```

```
}
}
```

参数说明

```
type MultiUploadOptions struct {
OptIni *InitiateMultipartUploadOptions
PartSize int64
ThreadPoolSize int
CheckPoint bool
}
```

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
filepath	本地文件名	string	是
opt	对象属性	Struct	否
OptIni	设置对象属性和 ACL，详情请参见 InitiateMultipartUploadOptions	Struct	否
PartSize	块大小，单位为 MB，如果用户不指定或者指定 partSize <= 0，由 Go SDK 自动切分，新版本默认大小为16MB	int	否
ThreadPoolSize	线程池大小，默认为1	int	否
CheckPoint	是否开启断点续传，默认为 false	bool	否

返回结果说明

```
type CompleteMultipartUploadResult struct {
Location string
Bucket string
Key string
ETag string
}
```

参数名称	参数描述	类型
Location	URL 地址	string

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
Bucket	存储桶名称，格式：BucketName-APPID。例如 examplebucket-1250000000	string
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string
ETag	合并后对象的唯一标签值，该值不是对象内容的 MD5 校验值，仅能用于检查对象唯一性。如需校验文件内容，可以在上传过程中校验单个分块的 ETag 值	string

简单操作

简单上传对象

功能说明

上传一个 Object（文件/对象）至存储桶（PUT Object）。最大支持5GB（含），5GB以上的对象请使用 分块上传 或 高级接口 上传。支持简单上传对象、创建文件夹、批量上传操作。

方法原型

```
func (s *ObjectService) Put(ctx context.Context, key string, r io.Reader, opt *ObjectPutOptions) (*Response, error)
func (s *ObjectService) PutFromFile(ctx context.Context, name string, filePath string, opt *ObjectPutOptions) (*Response, error)
```

请求示例1：上传对象

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"strings"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
```

```
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
// Case1 使用 Put 上传对象
key := "exampleobject"
f, err := os.Open("../test")
opt := &cos.ObjectPutOptions{
ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
ContentType: "text/html",
},
ACLHeaderOptions: &cos.ACLHeaderOptions{
// 如果不是必要操作，建议上传文件时不要给单个文件设置权限，避免达到限制。若不设置默认继承桶的权限。
XCosACL: "private",
},
}
_, err = client.Object.Put(context.Background(), key, f, opt)
if err != nil {
panic(err)
}
// Case 2 使用 PUtFromFile 上传本地文件到 COS
filepath := "./test"
_, err = client.Object.PutFromFile(context.Background(), key, filepath, opt)
if err != nil {
panic(err)
}
// Case 3 上传 0 字节文件，设置输入流长度为 0
_, err = client.Object.Put(context.Background(), key, strings.NewReader(""), nil)
if err != nil {
// ERROR
}
}
```

请求示例2：创建文件夹

COS 上可以将以 '/' 分隔的对象路径看做虚拟文件夹，根据此特性，可以上传一个空的流，并且命名以 '/' 结尾，可实现在 COS 上创建一个空文件夹。

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"strings"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
// 文件夹名称
name := "folder/"
// 传递大小为0的输入流
_, err := client.Object.Put(context.Background(), name, strings.NewReader(""), nil)
if err != nil {
// ERROR
```

```
}
}
```

请求示例3：上传到虚拟目录

上传由 '/' 分隔的对象名，自动创建包含文件的文件夹。想要在此文件夹中添加新文件时，只需要在上传文件至 COS 时，将 Key 填写为此目录前缀即可。

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"strings"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
dir := "exampledir/"
filename := "exampleobject"
key := dir + filename
f := strings.NewReader("test file")
_, err = client.Object.Put(context.Background(), key, f, nil)
```

```
if err != nil {
// ERROR
}
}
```

请求示例4：查看上传进度

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
type SelfListener struct {
}
// 自定义进度回调，需要实现 ProgressChangedCallback 方法
func (l *SelfListener) ProgressChangedCallback(event *cos.ProgressEvent) {
switch event.EventType {
case cos.ProgressDataEvent:
fmt.Printf("\r[ConsumedBytes/TotalBytes: %d/%d, %d%%]",
event.ConsumedBytes, event.TotalBytes, event.ConsumedBytes*100/event.TotalBytes)
case cos.ProgressFailedEvent:
fmt.Printf("\nTransfer Failed: %v", event.Err)
}
}
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
t/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
m")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.int
l.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权
限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/59
8/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.i
```

```
ntl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
// Case 1 通过默认回调查看上传进度
key := "exampleobject"
f, err := os.Open("test")
opt := &cos.ObjectPutOptions{
ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
ContentType: "text/html",
// 设置默认的进度回调函数
Listener: &cos.DefaultProgressListener{},
},
ACLHeaderOptions: &cos.ACLHeaderOptions{
// 如果不是必要操作，建议上传文件时不要给单个文件设置权限，避免达到限制。若不设置默认继承桶的权限。
XCosACL: "private",
},
}
_, err = client.Object.Put(context.Background(), key, f, opt)
if err != nil {
panic(err)
}
// Case 2 通过自定义方式查看上传进度
opt.Listener = &SelfListener{}
filepath := "./test"
_, err = client.Object.PutFromFile(context.Background(), key, filepath, opt)
if err != nil {
panic(err)
}
}
```

请求示例5：多线程批量上传

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"sync"
)
func upload(wg *sync.WaitGroup, c *cos.Client, files <-chan string) {
```

```
defer wg.Done()
for file := range files {
name := "folder/" + file
fd, err := os.Open(file)
if err != nil {
//ERROR
continue
}
_, err = c.Object.Put(context.Background(), name, fd, nil)
if err != nil {
//ERROR
}
}
}
func main() {
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
// 多线程批量上传文件
filesCh := make(chan string, 2)
filePaths := []string{"test1", "test2", "test3"}
var wg sync.WaitGroup
threadpool := 2
for i := 0; i < threadpool; i++ {
wg.Add(1)
go upload(&wg, c, filesCh)
}
for _, filePath := range filePaths {
filesCh <- filePath
}
close(filesCh)
wg.Wait()
}
```

请求示例6：MD5校验

Go SDK v0.7.23及以上版本, 上传文件时默认都已开启了CRC64校验，用户也可自行做MD5校验。

```
package main
import (
"context"
"crypto/md5"
"encoding/base64"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"io/ioutil"
"net/http"
"net/url"
"os"
"strings"
)
func calMD5Digest(msg []byte) []byte {
m := md5.New()
m.Write(msg)
return m.Sum(nil)
}
func main() {
u, _ := url.Parse("https://test-1259654469.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
// Go SDK v0.7.23及以上版本，上传文件时默认都开启了 CRC64校验
// c.Conf.EnableCRC = false // 手动关闭 CRC64校验，强烈不建议用户关闭 CRC64校验
// 计算 MD5
content := "test content"
bs1 := calMD5Digest([]byte(content))
md5str := fmt.Sprintf("%x", bs1)
name := "exampleobject"
f := strings.NewReader(content)
opt := &cos.ObjectPutOptions{
ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
ContentMD5: base64.StdEncoding.EncodeToString(bs1), // 上传时服务器根据收到的文件和该
```

```
值做校验
XCosMetaXXX: &http.Header{},
},
ACLHeaderOptions: nil,
}
opt.XCosMetaXXX.Add("x-cos-meta-md5", md5str) // 下载时业务可通过该值做文件校验
// 上传文件
_, err := c.Object.Put(context.Background(), name, f, opt)
if err != nil {
// ERROR
}
// 下载文件
resp, err := c.Object.Get(context.Background(), name, nil)
if err != nil {
// ERROR
}
defer resp.Body.Close()
meta_md5 := resp.Header.Get("x-cos-meta-md5")
body, _ := ioutil.ReadAll(resp.Body)
bs2 := calMD5Digest(body)
if fmt.Sprintf("%x", bs2) != meta_md5 {
fmt.Printf("md5 is not consistent\n")
}
}
```

参数说明

```
type ObjectPutOptions struct {
*ACLHeaderOptions
*ObjectPutHeaderOptions
}
type ACLHeaderOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
}
type ObjectPutHeaderOptions struct {
CacheControl string
ContentDisposition string
ContentEncoding string
ContentType string
ContentMD5 string
ContentLength int64
Expires string
// 自定义的 x-cos-meta-* header
```

版权所有：腾讯云计算（北京）有限责任公司

```
XCosMetaXXX *http.Header
XCosStorageClass string
XCosTrafficLimit int
Listener ProgressListener
}
```

参数名称	参数描述	类型	是否必填
r	上传文件的内容，可以为文件流或字节流，当 r 不是 `bytes.Buffer/bytes.Reader/strings.Reader` 时，必须指定 `opt.ObjectPutHeaderOptions.ContentLength`	io.Reader	是
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
XCosACL	设置文件的 ACL，例如 private，public-read，public-read-write	string	否
XCosGrantFullControl	赋予被授权者所有的权限。格式：id="[OwnerUin]"	string	否
XCosGrantRead	赋予被授权者读的权限。格式：id="[OwnerUin]"	string	否
XCosStorageClass	文件的存储类型，例如 STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD。更多存储类型，请参见 存储类型概述	string	否
Expires	设置 Content-Expires	string	否
CacheControl	缓存策略，设置 Cache-Control	string	否
ContentType	内容类型，设置 Content-Type	string	否
ContentMD5	RFC 1864中定义的请求体内容的16字节二进制 MD5 哈希值的 Base64 编码形式，用于完整性检查，验证请求体在传输过程中是否发生变化，最终的取值长度应为24个字符，请注意在编写代码时使用正确的方法和参数，例如 ZzD3iDJdrMAAb00lgLLeig==	string	否
ContentDisposition	文件名称，设置 Content-Disposition	string	否
ContentEncoding	编码格式，设置 Content-Encoding	string	否
ContentLength	设置传输长度	int64	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
XCosMetaXXX	用户自定义的文件元信息，必须以 x-cos-meta 开头，否则会被忽略	http.Header	否
XCosTrafficLimit	设置单链接限速	int	否
Listener	进度回调接口	Struct	否

返回结果说明

```
{
'ETag': 'string',
'x-cos-expiration': 'string'
}
```

通过返回结果 Response 获取。

```
resp, err := client.Object.Put(context.Background(), key, f, nil)
etag := resp.Header.Get("ETag")
exp := resp.Header.Get("x-cos-expiration")
```

参数名称	参数描述	类型
ETag	上传文件的 MD5 值	string
x-cos-expiration	设置生命周期后，返回文件过期规则	string

追加上传对象

功能说明

以分块追加的方式上传对象（APPEND Object）。

方法原型

```
func (s *ObjectService) Append(ctx context.Context, name string, position int, r
io.Reader, opt *ObjectPutOptions) (int, *Response, error)
```

请求示例

```
package main
import (
"context"
```

```
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"strings"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
name := "exampleobject"
pos, _, err := client.Object.Append(context.Background(), name, 0, strings.NewReader("test1"), nil)
if err != nil {
// ERROR
}
_, _, err = client.Object.Append(context.Background(), name, pos, strings.NewReader("test2"), nil)
if err != nil {
// ERROR
}
}
```

参数说明

参数名称	参数描述	类型

参数名称	参数描述	类型
name	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string
position	追加操作的起始点，单位为字节。首次追加则设置 Position=0，后续追加则设置 Position 为当前 Object 的 content-length	int
r	上传文件的内容，可以为文件流或字节流，当 r 不是 bytes.Buffer/bytes.Reader/strings.Reader时，必须指定 opt.ObjectPutHeaderOptions.ContentLength	io.Reader
opt	上传参数，详见 ObjectPutOptions	struct

返回结果说明

参数名称	参数描述	类型
x-cos-next-append-position	下一次追加操作的起始点，单位：字节	int

分块操作

查询分块上传

功能说明

查询指定存储桶中正在进行的分块上传信息（List Multipart Uploads）。

方法原型

```
func (s *BucketService) ListMultipartUploads(ctx context.Context, opt *ListMultipartUploadsOptions) (*ListMultipartUploadsResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
```

```
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, _, err := client.Bucket.ListMultipartUploads(context.Background(), nil)
if err != nil {
panic(err)
}
}
```

参数说明

```
type ListMultipartUploadsOptions struct {
Delimiter string
EncodingType string
Prefix string
MaxUploads int
KeyMarker string
UploadIDMarker string
}
```

参数名称	参数描述	类型	是否必填

参数名称	参数描述	类型	是否必填
Delimiter	定界符为一个符号，对 Object 名字包含指定前缀且第一次出现 delimiter 字符之间的 Object 作为一组元素：common prefix。如果没有 prefix，则从路径起点开始	string	否
EncodingType	规定返回值的编码格式，合法值：url	string	否
Prefix	限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	string	否
MaxUploads	设置最大返回的 multipart 数量，合法取值从1到1000，默认1000	int	否
KeyMarker	与 upload-id-marker 一起使用： • 当 upload-id-marker 未被指定时，ObjectName 字母顺序大于 key-marker 的条目将被列出 • 当 upload-id-marker 被指定时，ObjectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出	string	否
UploadIDMarker	与 key-marker 一起使用： • 当 key-marker 未被指定时，upload-id-marker 将被忽略 • 当 key-marker 被指定时，ObjectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出	string	否

返回结果说明

```
// ListMultipartUploadsResult 保存 ListMultipartUploads 的结果
type ListMultipartUploadsResult struct {
Bucket string
EncodingType string
KeyMarker string
UploadIDMarker string
NextKeyMarker string
NextUploadIDMarker string
MaxUploads int
IsTruncated bool
Uploads []struct {
Key string
UploadID string
StorageClass string
Initiator *Initiator
Owner *Owner
Initiated string
}
```

版权所有：腾讯云计算（北京）有限责任公司

```
Prefix string
Delimiter string
CommonPrefixes []string
}
// 与 Owner 使用同样的结构
type Initiator Owner
// Owner 定义了 Bucket/Object's 拥有者
type Owner struct {
ID string
DisplayName string
}
```

参数名称	参数描述	类型
Bucket	分块上传的目标 Bucket，格式为 BucketName，例如 examplebucket-1250000000	string
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	string
KeyMarker	列出条目从该 key 值开始	string
UploadIDMarker	列出条目从该 UploadId 值开始	string
NextKeyMarker	假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点	string
NextUploadIDMarker	假如返回条目被截断，则返回 UploadId 就是下一个条目的起点	string
MaxUploads	最多返回的分块的数量，默认为最大的1000	string
IsTruncated	表示返回的分块是否被截断	bool
Uploads	每个 Upload 的信息	Container
Key	Object 的名称	string
UploadID	标示本次分块上传的 ID	string
Key	表示返回的分块是否被截断	bool
StorageClass	用来表示分块的存储类型，例如 STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD。更多存储类型，请参见 存储类型概述	string
Initiator	用来表示本次上传发起者的信息	Container
Owner	用来表示这些分块所有者的信息	Container
Initiated	分块上传的起始时间	string

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
Prefix	限定返回的 Objectkey 必须以 Prefix 作为前缀，注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	struct
Delimiter	定界符为一个符号，对 object 名字包含指定前缀且第一次出现 delimiter 字符之间的 object 作为一组元素：common prefix。如果没有prefix，则从路径起点开始	string
CommonPrefixes	将 prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	string
ID	用户唯一的 CAM 身份 ID	string
DisplayName	用户身份 ID 的简称（UIN）	string

分块上传对象

分块上传对象可包括的操作：

- 分块上传对象：初始化分块上传，上传分块，完成分块上传。
- 删除已上传分块。

> 说明：
> 分块上传对象，您还可以使用 高级接口 上传（推荐）。

初始化分块上传

功能说明

初始化 Multipart Upload 上传操作，获取对应的 uploadId（Initiate Multipart Upload）。

方法原型

```
func (s *ObjectService) InitiateMultipartUpload(ctx context.Context, name string, opt *InitiateMultipartUploadOptions) (*InitiateMultipartUploadResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
```

```
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://cloud.tencent.com/document/product/598/37140
},
})
name := "exampleobject"
// 可选 opt,如果不是必要操作，建议上传文件时不要给单个文件设置权限，避免达到限制。若不设置默认继承桶的权限。
v, _, err := client.Object.InitiateMultipartUpload(context.Background(), name, nil)
if err != nil {
panic(err)
}
UploadID := v.UploadID
fmt.Println(UploadID)
}
```

参数说明

```
type InitiateMultipartUploadOptions struct {
*ACLHeaderOptions
*ObjectPutHeaderOptions
```

```
}
type ACLHeaderOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
}
type ObjectPutHeaderOptions struct {
CacheControl string
ContentDisposition string
ContentEncoding string
ContentType string
ContentLength int64
Expires string
// 自定义的 x-cos-meta-* header
XCosMetaXXX *http.Header
XCosStorageClass string
}
```

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
XCosACL	设置文件 ACL，例如 private，public-read	string	否
XCosGrantFullControl	赋予被授权者所有的权限。格式：id="[OwnerUin]"	string	否
XCosGrantRead	赋予被授权者读的权限。格式：id="[OwnerUin]"	string	否
XCosStorageClass	文件的存储类型，例如 STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD。更多存储类型，请参见 存储类型概述	string	否
Expires	设置 Content-Expires	string	否
CacheControl	缓存策略，设置 Cache-Control	string	否
ContentType	内容类型，设置 Content-Type	string	否
ContentDisposition	文件名称，设置 Content-Disposition	string	否
ContentEncoding	编码格式，设置 Content-Encoding	string	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
ContentLength	设置传输长度	int64	否
XCosMetaXXX	用户自定义的文件元信息，必须以 x-cos-meta 开头，否则会被忽略	http.Header	否

返回结果说明

```
type InitiateMultipartUploadResult struct {
Bucket string
Key string
UploadID string
}
```

参数名称	参数描述	类型
UploadId	标识分块上传的 ID	string
Bucket	Bucket 名称，由 bucket-appid 组成	string
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string

上传分块

分块上传对象（Upload Part）。

方法原型

```
func (s *ObjectService) UploadPart(ctx context.Context, key, uploadID string, partNumber int, r io.Reader, opt *ObjectUploadPartOptions) (*Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
```

```
"strings"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
name := "exampleobject"
// 可选 opt,如果不是必要操作，建议上传文件时不要给单个文件设置权限，避免达到限制。若不设置默认继承桶的权限。
v, _, err := client.Object.InitiateMultipartUpload(context.Background(), name, nil)
if err != nil {
panic(err)
}
UploadID := v.UploadID
fmt.Println(UploadID)
// 注意，上传分块的块数最多10000块
key := "exampleobject"
f := strings.NewReader("test hello")
// opt 可选
resp, err := client.Object.UploadPart(
context.Background(), key, UploadID, 1, f, nil,
)
if err != nil {
panic(err)
}
PartETag := resp.Header.Get("ETag")
```

```
fmt.Println(PartETag)
}
```

参数说明

```
type ObjectUploadPartOptions struct {
ContentLength int64
ContentMD5 string
}
```

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
UploadId	标识分块上传的 ID，由 InitiateMultipartUpload 生成	string	是
PartNumber	标识上传分块的序号	int	是
r	上传分块的内容，可以为本地文件流或输入流。当 r 不是 `bytes.Buffer/bytes.Reader/strings.Reader` 时，必须指定 opt.ContentLength	io.Reader	是
ContentLength	设置传输长度	int64	否
ContentMD5	RFC 1864中定义的请求体内容的16字节二进制 MD5 哈希值的 Base64 编码形式，用于完整性检查，验证请求体在传输过程中是否发生变化，最终的取值长度应为24个字符，请注意在编写代码时使用正确的方法和参数，例如 ZzD3iDJdrMAAb00lgLLeig==	string	否

返回结果说明

```
{
'ETag': 'string'
}
```

通过返回结果Response获取。

```
resp, err := client.Object.UploadPart(context.Background(), key, UploadID, 1, f,
nil)
etag := resp.Header.Get("ETag")
```

参数名称	参数描述	类型
ETag	上传分块的 MD5 值	string

查询已上传块

功能说明

查询特定分块上传操作中的已上传的块（List Parts）。

方法原型

```
func (s *ObjectService) ListParts(ctx context.Context, name, uploadID string, opt *ObjectListPartsOptions) (*ObjectListPartsResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
```

```
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://cloud.tencent.com/document/product/598/37140
},
})
key := "exampleobject"
v, _, err := client.Object.InitiateMultipartUpload(context.Background(), key, nil)
if err != nil {
panic(err)
}
UploadID := v.UploadID
fmt.Println(UploadID)
_, _, err = client.Object.ListParts(context.Background(), key, UploadID, nil)
if err != nil {
panic(err)
}
}
```

参数说明

```
type ObjectListPartsOptions struct {
EncodingType string
MaxParts string
PartNumberMarker string
}
```

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
UploadId	标识分块上传的 ID，由 InitiateMultipartUpload 生成	string	是
EncodingType	规定返回值的编码方式	string	否
MaxParts	单次返回最大的条目数量，默认1000	string	否
PartNumberMarker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	string	否

返回结果说明

```
type ObjectListPartsResult struct {
Bucket string
EncodingType string
Key string
UploadID string
Initiator *Initiator
Owner *Owner
StorageClass string
PartNumberMarker string
NextPartNumberMarker string
MaxParts string
IsTruncated bool
Parts []Object
}
type Initiator struct {
UIN string
ID string
DisplayName string
}
type Owner struct {
UIN string
ID string
DisplayName string
}
type Object struct {
Key string
ETag string
Size int
PartNumber int
LastModified string
StorageClass string
Owner *Owner
}
```

参数名称	参数描述	类型
Bucket	存储桶名称，格式：BucketName-APPID。例如 examplebucket-1250000000	string
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	string
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string

参数名称	参数描述	类型
UploadId	标识分块上传的 ID，由 InitiateMultipartUpload 生成	string
Initiator	分块上传的创建者，包括 DisplayName，UIN 和 ID	struct
Owner	文件拥有者的信息，包括 DisplayName ，UIN 和 ID	struct
StorageClass	文件的存储类型，例如 STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD。更多存储类型，请参见 存储类型概述	string
PartNumberMarker	默认为0，从第一块列出分块，从 PartNumberMarker 下一个分块开始列出	string
NextPartNumberMarker	指明下一次列出分块的起始位置	string
MaxParts	最多返回的分块的数量，默认为最大的1000	string
IsTruncated	表示返回的分块是否被截断	bool
Part	上传分块的相关信息，包括 ETag，PartNumber，Size，LastModified	struct

完成分块上传

功能说明

完成整个文件的分块上传（Complete Multipart Upload）。

方法原型

```
func (s *ObjectService) CompleteMultipartUpload(ctx context.Context, key, uploadID string, opt *CompleteMultipartUploadOptions) (*CompleteMultipartUploadResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"strings"
)
```

```
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
name := "exampleobject"
// 可选 opt,如果不是必要操作，建议上传文件时不要给单个文件设置权限，避免达到限制。若不设置默认继承桶的权限。
v, _, err := client.Object.InitiateMultipartUpload(context.Background(), name, nil)
if err != nil {
panic(err)
}
UploadID := v.UploadID
fmt.Println(UploadID)
// 注意，上传分块的块数最多10000块
key := "exampleobject"
f := strings.NewReader("test hello")
// opt可选
resp, err := client.Object.UploadPart(
context.Background(), key, UploadID, 1, f, nil,
)
if err != nil {
panic(err)
}
PartETag := resp.Header.Get("ETag")
fmt.Println(PartETag)
opt := &cos.CompleteMultipartUploadOptions{}
```

```
opt.Parts = append(opt.Parts, cos.Object{
PartNumber: 1, ETag: PartETag},
)
_, _, err = client.Object.CompleteMultipartUpload(
context.Background(), key, UploadID, opt,
)
if err != nil {
panic(err)
}
}
```

参数说明

```
type CompleteMultipartUploadOptions struct {
Parts []Object
}
type Object struct {
ETag string
PartNumber int
}
```

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
UploadId	标识分块上传的 ID，由 InitiateMultipartUpload 生成	string	是
CompleteMultipartUploadOptions	所有分块的 ETag 和 PartNumber 信息	struct	是

返回结果说明

```
type CompleteMultipartUploadResult struct {
Location string
Bucket string
Key string
ETag string
}
```

参数名称	参数描述	类型

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
Location	URL 地址	string
Bucket	存储桶名称，格式：BucketName-APPID。例如 examplebucket-1250000000	string
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string
ETag	合并后对象的唯一标签值，该值不是对象内容的 MD5 校验值，仅能用于检查对象唯一性。如需校验文件内容，可以在上传过程中校验单个分块的 ETag 值	string

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块（Abort Multipart Upload）。

方法原型

```
func (s *ObjectService) AbortMultipartUpload(ctx context.Context, key, uploadID string) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
```

```
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考 https://cloud.tencent.com/document/product/598/37140
},
})
key := "exampleobject"
v, _, err := client.Object.InitiateMultipartUpload(context.Background(), key, nil)
if err != nil {
panic(err)
}
UploadID := v.UploadID
// Abort
_, err = client.Object.AbortMultipartUpload(context.Background(), key, UploadID)
if err != nil {
panic(err)
}
}
```

参数说明

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
UploadId	标识分块上传的 ID	string	是

复制与移动对象

最近更新时间：2023-03-23 15:58:31

简介

本文档提供关于对象的复制、移动操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object - Copy	设置对象复制（修改对象属性）	复制文件到目标路径

分块操作

API	操作名	操作描述
Initiate Multipart Upload	初始化分块上传/复制	初始化分块上传/复制操作
Upload Part - Copy	复制分块	将其他对象复制为一个分块
Complete Multipart Upload	完成分块上传/复制	完成整个对象的分块上传/复制

简单操作

复制对象

PUT Object - Copy 请求创建一个已存在 COS 的对象的副本，即将一个对象从源路径（对象键）复制到目标路径（对象键）。在复制的过程中，对象元数据和访问控制列表（ACL）可以被修改。
用户可以通过该接口创建副本、修改对象元属性（源对象和目标文件的属性相同）、移动或重命名对象（先复制，再单独调用删除接口）。

方法原型

```
func (s *ObjectService) Copy(ctx context.Context, key, sourceURL string, opt *Obj
ectCopyOptions) (*ObjectCopyResult, *Response, error)
```

请求示例1：简单复制对象

```
package main

import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"strings"
)

func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
name := "exampleobject"
// 上传源对象
f := strings.NewReader("test")
_, err := client.Object.Put(context.Background(), name, f, nil)

sourceURL := fmt.Sprintf("examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/%s", name)
dest := "example_dest"
// 如果不是必要操作，建议上传文件时不要给单个文件设置权限，避免达到限制。若不设置默认继承桶的权限。
```

```
// opt := &cos.ObjectCopyOptions{}
_, _, err = client.Object.Copy(context.Background(), dest, sourceURL, nil)
if err != nil {
panic(err)
}
}
```

请求示例2：移动对象

```
package main

import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"strings"
)

func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
source := "test/oldfile"
```

```
f := strings.NewReader("test")
// 上传文件
_, err := client.Object.Put(context.Background(), source, f, nil)
if err != nil {
// Error
}

// 移动对象
dest := "test/newfile"
sourceURL := fmt.Sprintf("examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/%s", source)
_, _, err = client.Object.Copy(context.Background(), dest, sourceURL, nil)
if err == nil {
_, err = client.Object.Delete(context.Background(), source, nil)
if err != nil {
// Error
}
}
}
```

请求示例3：修改存储类型

> 注意：
> 标准存储可以修改为低频存储、智能分层存储、归档存储和深度归档存储等，如果希望将归档存储或深度归档存储的对象修改为其他存储类型，首先需要使用 PostRestore 将归档存储或深度归档存储的对象回热，才能使用该接口请求修改存储类型；关于存储类型的详细说明请参见 存储类型概述。

```
package main

import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"strings"
)

func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
```

```
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
name := "exampleobject"
// 上传源对象
f := strings.NewReader("test")
_, err := client.Object.Put(context.Background(), name, f, nil)

sourceURL := fmt.Sprintf("%s/%s", client.BaseURL.BucketURL.Host, name)
opt := &cos.ObjectCopyOptions{
&cos.ObjectCopyHeaderOptions{
XCosMetadataDirective: "Replaced",
XCosStorageClass: "Archive", // 修改成归档类型
},
nil,
}
_, _, err = client.Object.Copy(context.Background(), name, sourceURL, opt)
if err != nil {
panic(err)
}
}
```

参数说明

```
type ObjectCopyOptions struct {
*ObjectCopyHeaderOptions
*ACLHeaderOptions
```

```
}
type ACLHeaderOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
}
type ObjectCopyHeaderOptions struct {
XCosMetadataDirective string
XCosCopySourceIfModifiedSince string
XCosCopySourceIfUnmodifiedSince string
XCosCopySourceIfMatch string
XCosCopySourceIfNoneMatch string
XCosStorageClass string
// 自定义的 x-cos-meta-* header
XCosMetaXXX *http.Header
XCosCopySource string
}
```

参数名称	参数描述	类型	是否
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
sourceURL	描述拷贝源文件的 URL	string	是
XCosACL	设置文件的 ACL，例如 private，public-read，public-read-write	string	否
XCosGrantFullControl	赋予被授权者所有的权限。格式：id="[OwnerUin]"	string	否
XCosGrantRead	赋予被授权者读的权限。格式：id="[OwnerUin]"	string	否
XCosMetadataDirective	可选值为 Copy，Replaced： • 设置为 Copy 时，忽略设置的用户元数据信息直接复制 • 设置为 Replaced 时，按设置的元信息修改元数据 当目标路径和源路径一样时，必须设置为 Replaced	string	是

参数名称	参数描述	类型	是否
XCosCopySourceIfModifiedSince	当 Object 在指定时间后被修改，则执行操作，否则返回412。可与 XCosCopySourceIfNoneMatch 一起使用，与其他条件联合使用返回冲突	string	否
XCosCopySourceIfUnmodifiedSince	当 Object 在指定时间后未被修改，则执行操作，否则返回412。可与 XCosCopySourceIfMatch 一起使用，与其他条件联合使用返回冲突	string	否
XCosCopySourceIfMatch	当 Object 的 Etag 和给定一致时，则执行操作，否则返回412。可与 XCosCopySourceIfUnmodifiedSince 一起使用，与其他条件联合使用返回冲突	string	否
XCosCopySourceIfNoneMatch	当 Object 的 Etag 和给定不一致时，则执行操作，否则返回412。可与 XCosCopySourceIfModifiedSince 一起使用，与其他条件联合使用返回冲突	string	否
XCosStorageClass	设置文件的存储类型，STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD	string	否
XCosMetaXXX	用户自定义的文件元信息	http.Header	否
XCosCopySource	源文件 URL 路径，可以通过 versionid 子资源指定历史版本	string	否

返回结果说明

上传文件的属性：

```
type ObjectCopyResult struct {
ETag string
LastModified string
}
```

参数名称	参数描述	类型
ETag	拷贝文件的 MD5 值	string
LastModified	拷贝文件的最后一次修改时间	string

移动对象

功能说明

移动对象主要包括两个操作：复制源对象到目标位置，删除源对象。

由于 COS 通过存储桶名称（Bucket）和对象键（ObjectKey）来标识对象。移动对象也就意味着修改这个对象的标识，COS目前没有提供修改对象唯一标识名的单独接口，但是可以通过组合复制对象加上删除对象的基本操作，来达到修改对象标识的目的，从而实现移动对象。

例如将 mybucket-1250000000 这个存储桶中的 picture.jpg 这个对象移动到同个存储桶的 doc 路径下。首先可以复制 picture.jpg 对象到存储桶的 doc 路径下，即对象键设定为 doc/picture.jpg，复制完成后删除 picture.jpg 这个对象，来实现“移动”的效果。

同样的，如果想将 mybucket-1250000000 这个存储桶里的 picture.jpg 这个对象移动到 myanothorbucket-1250000000 这个存储桶里，可以先将对象复制到 myanothorbucket-1250000000 存储桶，然后删除原来的对象。

请求示例

```
package main

import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"strings"
)

func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/59
```

```
8/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
source := "test/oldfile"
f := strings.NewReader("test")
// 上传文件
_, err := client.Object.Put(context.Background(), source, f, nil)
if err != nil {
// Error
}
// 移动对象
dest := "test/newfile"
sourceURL := fmt.Sprintf("%s/%s", u.Host, source)
_, _, err = client.Object.Copy(context.Background(), dest, sourceURL, nil)
if err == nil {
_, err = client.Object.Delete(context.Background(), source, nil)
if err != nil {
// Error
}
}
}
```

分块操作

初始化分块复制

功能说明

Initiate Multipart Upload 请求实现初始化分块复制，成功执行此请求后会返回 Upload ID ，用于后续的 Upload Part - Copy 请求。

方法原型

```
func (s *ObjectService) InitiateMultipartUpload(ctx context.Context, name string, opt *InitiateMultipartUploadOptions) (*InitiateMultipartUploadResult, *Response, error)
```

请求示例

```
package main

import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)

func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
name := "exampleobject"
v, _, err := client.Object.InitiateMultipartUpload(context.Background(), name, nil)
if err != nil {
panic(err)
}
UploadID := v.UploadID
fmt.Println(UploadID)
}
```

版权所有：腾讯云计算（北京）有限责任公司

参数说明

```
type InitiateMultipartUploadOptions struct {
*ACLHeaderOptions
*ObjectPutHeaderOptions
}
type ACLHeaderOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
}
type ObjectPutHeaderOptions struct {
CacheControl string
ContentDisposition string
ContentEncoding string
ContentType string
ContentLength int64
Expires string
// 自定义的 x-cos-meta-* header
XCosMetaXXX *http.Header
XCosStorageClass string
}
```

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
XCosACL	设置文件 ACL，例如 private，public-read	string	否
XCosGrantFullControl	赋予被授权者所有的权限。格式：id="[OwnerUin]"	string	否
XCosGrantRead	赋予被授权者读的权限。格式：id="[OwnerUin]"	string	否
XCosStorageClass	设置文件的存储类型，STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD	string	否
Expires	设置 Content-Expires	string	否
CacheControl	缓存策略，设置 Cache-Control	string	否
ContentType	内容类型，设置 Content-Type	string	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
ContentDisposition	文件名称，设置 Content-Disposition	string	否
ContentEncoding	编码格式，设置 Content-Encoding	string	否
ContentLength	设置传输长度	int64	否
XCosMetaXXX	用户自定义的文件元信息， 必须以 x-cos-meta 开头，否则会被忽略	http.Header	否

返回结果说明

```
type InitiateMultipartUploadResult struct {
Bucket string
Key string
UploadID string
}
```

参数名称	参数描述	类型
UploadId	标识分块上传的 ID	string
Bucket	Bucket 名称，由 bucket-appid 组成	string
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string

复制分块

功能说明

Upload Part - Copy 接口请求实现将一个对象的分块内容从源路径复制到目标路径。

> 注意：
> 使用上传分块对象，必须先初始化分块上传。在初始化分块上传的响应中，会返回一个唯一的描述符（upload ID），您需要在分块上传请求中携带此 ID。

方法原型

```
func (s *ObjectService) CopyPart(ctx context.Context, key, uploadID string, partN
umber int, sourceURL string, opt *ObjectCopyPartOptions) (*CopyPartResult, *Respo
```

```
nse, error)
```

请求示例

```
package main

import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)

func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
// 初始化分块
name := "exampleobject"
v, _, err := client.Object.InitiateMultipartUpload(context.Background(), name, nil)
if err != nil {
// ERROR
}
uploadID := v.UploadID
```

```
// 复制分块
sourceUrl := "examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceobject"
res, _, err := client.Object.CopyPart(context.Background(), name, uploadID, 1, sourceUrl, nil)
if err != nil {
// ERROR
}

// 完成分块上传
completeOpt := &cos.CompleteMultipartUploadOptions{}
completeOpt.Parts = append(completeOpt.Parts, cos.Object{
PartNumber: 1,
ETag: res.ETag,
})
_, _, err = client.Object.CompleteMultipartUpload(context.Background(), name, uploadID, completeOpt)
if err != nil {
// ERROR
}
}
```

参数说明

```
type ObjectCopyPartOptions struct {
XCosCopySourceRange
XCosCopySourceIfModifiedSince
XCosCopySourceIfUnmodifiedSince
XCosCopySourceIfMatch
XCosCopySourceIfNoneMatch
}
```

参数名称	参数描述	类型	是否必填
key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	string	是
uploadID	使用上传分块文件，必须先初始化分块上传。在初始化分块上传的响应中，会返回一个唯一的描述符（upload ID），您需要在分块上传请求中携带此 ID	string	是
partNumber	分块拷贝的块号	int	是

参数名称	参数描述	类型	是否必填
sourceURL	源对象 URL 路径，`<bucketname-appid>.cos.ap-guangzhou.myqcloud.com/<source>`。可以通过 URL 参数 `?versionId=<versionid>` 参数指定指定历史版本。	string	是
opt	分块拷贝参数	struct	否
XCosCopySourceRange	源对象的字节范围，范围值必须使用 bytes=first-last 格式，first 和 last 都是基于0开始的偏移量。例如 bytes=0-9 表示您希望拷贝源对象的开头10个字节的数据，如果不指定，则表示拷贝整个对象	string	否
XCosCopySourceIfModifiedSince	当对象在指定时间后被修改，则执行操作，否则返回412，可与 x-cos-copy-source-If-None-Match 一起使用，与其他条件联合使用返回冲突	string	否
XCosCopySourceIfUnmodifiedSince	当对象在指定时间后未被修改，则执行操作，否则返回412，可与 x-cos-copy-source-If-Match 一起使用，与其他条件联合使用返回冲突	string	否
XCosCopySourceIfMatch	当对象的 Etag 和给定一致时，则执行操作，否则返回412，可与 x-cos-copy-source-If-Unmodified-Since 一起使用，与其他条件联合使用返回冲突	string	否
XCosCopySourceIfNoneMatch	当对象的 Etag 和给定不一致时，则执行操作，否则返回412，可与 x-cos-copy-source-If-Modified-Since 一起使用，与其他条件联合使用返回冲突	string	否

返回结果说明

```
type CopyPartResult struct {
ETag string
LastModified string
}
```

参数名称	参数描述	类型

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
ETag	文件的 MD5 算法校验值，例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	string
LastModified	返回对象最后修改时间，GMT 格式	string

完成分块复制

功能说明

Complete Multipart Upload 接口请求用来实现完成整个分块复制。当使用 Copy Parts 复制完所有块以后，必须调用该 API 来完成整个文件的分块复制。在使用该 API 时，您必须在请求 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。
由于分块复制完后需要合并，而合并需要数分钟时间，因而当合并分块开始的时候，COS 就立即返回200的状态码，在合并的过程中，COS 会周期性的返回空格信息来保持连接活跃，直到合并完成，COS 会在 Body 中返回合并后块的内容。

- 当上传块小于1MB的时候，在调用该 API 时，会返回400 EntityTooSmall。
- 当上传块编号不连续的时候，在调用该 API 时，会返回400 InvalidPart。
- 当请求 Body 中的块信息没有按序号从小到大排列的时候，在调用该 API 时，会返回400 InvalidPartOrder。
- 当 UploadId 不存在的时候，在调用该 API 时，会返回404 NoSuchUpload。

> 注意：
> 建议您及时完成分块复制或者舍弃分块复制，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

方法原型

```
func (s *ObjectService) CompleteMultipartUpload(ctx context.Context, key, uploadID string, opt *CompleteMultipartUploadOptions) (*CompleteMultipartUploadResult, *Response, error)
```

请求示例

```
package main

import (
"context"
```

```
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)

func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
// 初始化分块
name := "exampleobject"
v, _, err := client.Object.InitiateMultipartUpload(context.Background(), name, nil)
if err != nil {
// ERROR
}
uploadID := v.UploadID

// 复制分块
sourceUrl := "examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceobject"
res, _, err := client.Object.CopyPart(context.Background(), name, uploadID, 1, sourceUrl, nil)
if err != nil {
// ERROR
}
```

```
// 完成分块上传
completeOpt := &cos.CompleteMultipartUploadOptions{}
completeOpt.Parts = append(completeOpt.Parts, cos.Object{
PartNumber: 1,
ETag: res.ETag,
})
_, _, err = client.Object.CompleteMultipartUpload(context.Background(), name, upl
oadID, completeOpt)
if err != nil {
// ERROR
}
}
```

参数说明

```
type CompleteMultipartUploadOptions struct {
Parts []Object
}
type Object struct {
ETag string
PartNumber int
}
```

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
UploadId	标识分块上传的 ID，由 InitiateMultipartUpload 生成	string	是
CompleteMultipartUploadOptions	所有分块的 ETag 和 PartNumber 信息	struct	是

返回结果说明

```
type CompleteMultipartUploadResult struct {
Location string
Bucket string
Key string
ETag string
}
```

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
Location	URL 地址	string
Bucket	存储桶名称，格式：BucketName-APPID。例如 examplebucket-1250000000	string
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string
ETag	合并后对象的唯一标签值，该值不是对象内容的 MD5 校验值，仅能用于检查对象唯一性。如需校验文件内容，可以在上传过程中校验单个分块的 ETag 值	string

下载对象

最近更新时间：2023-02-08 15:36:40

简介

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Object	下载对象	下载一个 Object（文件/对象）至本地

高级接口（推荐）

下载对象

功能说明

分块下载接口根据用户对象的长度，自动使用 Range 下载数据，可以实现并发下载，当对象大于16MB时，采用 Range 方式下载文件，可通过 PartSize 参数调整。

方法原型

```
func (s *ObjectService) Download(ctx context.Context, name string, filepath string, opt *MultiDownloadOptions) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
```

```
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
key := "exampleobject"
file := "localfile"
opt := &cos.MultiDownloadOptions{
ThreadPoolSize: 5,
}
_, err := client.Object.Download(
context.Background(), key, file, opt,
)
if err != nil {
panic(err)
}
}
```

参数说明

```
type MultiDownloadOptions struct {
Opt *ObjectGetOptions
PartSize int64
ThreadPoolSize int
CheckPoint bool
CheckPointFile string
}
```

参数名称	参数描述	类型	是否必填

参数名称	参数描述	类型	是否必填
name	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
filepath	本地文件名	string	是
opt	下载对象参数	Struct	否
Opt	请求参数，详情请参见 ObjectGetOptions	Struct	否
PartSize	块大小，单位为 MB，如果用户不指定或者指定 partSize <= 0，由 Go SDK 自动切分，新版本中默认16MB	int64	否
ThreadPoolSize	线程池大小，默认为1	int	否
CheckPoint	是否开启断点续传，默认为false	bool	否
CheckPointFile	开启断点续传时，表示保存下载进度的文件路径，默认路径为 <filepath>.cosresumabletask，当下载完成后，该进度文件会被清理掉	string	否

返回结果说明

参数名称	参数描述	类型
*Response	http 响应, 通过该返回结果获取响应状态码和响应头部等信息	Struct
error	出错信息，正常返回为 nil	Struct

简单操作

下载对象

功能说明

下载一个 Object（文件/对象）至本地（GET Object）。支持简单下载、批量下载操作。

方法原型

```
func (s *ObjectService) Get(ctx context.Context, key string, opt *ObjectGetOptions) (*Response, error)
```

```
func (s *ObjectService) GetToFile(ctx context.Context, key, localfile string, opt *ObjectGetOptions) (*Response, error)
```

请求示例1：下载对象

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"io/ioutil"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
key := "exampleobject"
opt := &cos.ObjectGetOptions{
ResponseContentType: "text/html",
Range: "bytes=0-3", // 通过 range 下载0~3字节的数据
}
// opt 可选，无特殊设置可设为 nil
// 1. 从响应体中获取对象
resp, err := client.Object.Get(context.Background(), key, opt)
```

```
if err != nil {
panic(err)
}
ioutil.ReadAll(resp.Body)
resp.Body.Close()
// 2. 下载对象到本地文件
_, err = client.Object.GetToFile(context.Background(), key, "example.txt", nil)
if err != nil {
panic(err)
}
}
```

请求示例2：批量下载（从COS下载目录）

```
package main
import (
"context"
"fmt"
"net/http"
"net/url"
"os"
"path"
"strings"
"sync"
"github.com/tencentyun/cos-go-sdk-v5"
)
func download(wg *sync.WaitGroup, c *cos.Client, keysCh <-chan []string) {
defer wg.Done()
for keys := range keysCh {
key := keys[0]
filename := keys[1]
_, err := c.Object.GetToFile(context.Background(), key, filename, nil)
if err != nil {
fmt.Println(err)
}
}
}
func main() {
u, _ := url.Parse("https://test-1259654469.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// os.Getenv 表示获取环境变量
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
```

```
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
// 多线程执行
keysCh := make(chan []string, 3)
var wg sync.WaitGroup
threadpool := 3
for i := 0; i < threadpool; i++ {
wg.Add(1)
go download(&wg, c, keysCh)
}
isTruncated := true
prefix := "dir" // 下载 dir 目录下所有文件
marker := ""
localDir := "./local/"
for isTruncated {
opt := &cos.BucketGetOptions{
Prefix: prefix,
Marker: marker,
EncodingType: "url", // url 编码
}
// 列出目录
v, _, err := c.Bucket.Get(context.Background(), opt)
if err != nil {
fmt.Println(err)
break
}
for _, c := range v.Contents {
key, _ := cos.DecodeURIComponent(c.Key) //EncodingType: "url", 先对 key 进行 url decode
localfile := localDir + key
if _, err := os.Stat(path.Dir(localfile)); err != nil && os.IsNotExist(err) {
os.MkdirAll(path.Dir(localfile), os.ModePerm)
}
// 以/结尾的key（目录文件）不需要下载
if strings.HasSuffix(localfile, "/") {
continue
}
keysCh <- []string{key, localfile}
}
marker, _ = cos.DecodeURIComponent(v.NextMarker) // EncodingType: "url", 先对 NextMarker 进行 url decode
isTruncated = v.IsTruncated
}
close(keysCh)
```

```
wg.Wait()
}
```

请求示例3：单链接限速

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
key := "exampleobject"
opt := &cos.ObjectGetOptions{
// 限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，如果超出该范围将返回400错误
XCosTrafficLimit: 819200,
}
// opt 可选，无特殊设置可设为 nil
// 1. 从响应体中获取对象
_, err := client.Object.GetToFile(context.Background(), key, "example.txt", opt)
if err != nil {
```

```
panic(err)
}
}
```

请求示例4：获取下载进度

Go SDK 通过回调的方式获取下载进度，用户需要自行实现 cos.ProgressListener 接口，接口定义如下：

```
const (
// 数据开始传输
ProgressStartedEvent ProgressEventType = iota
// 数据传输中
ProgressDataEvent
// 数据传输完成，但不能表示对应 API 调用完成
ProgressCompletedEvent
// 只有在数据传输时发生错误才会返回
ProgressFailedEvent
)
type ProgressEvent struct {
EventType ProgressEventType
RWBytes int64 // 单次读写的字节
ConsumedBytes int64 // 已完成的字节
TotalBytes int64 // 总字节
Err error // 错误
}
type ProgressListener interface {
ProgressChangedCallback(event *ProgressEvent)
}
```

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"io/ioutil"
"net/http"
"net/url"
"os"
)
type SelfListener struct {
}
// 自定义进度回调，需要实现 ProgressChangedCallback 方法
func (l *SelfListener) ProgressChangedCallback(event *cos.ProgressEvent) {
switch event.EventType {
case cos.ProgressDataEvent:
fmt.Printf("\r[ConsumedBytes/TotalBytes: %d/%d, %d%%]",
```

```
event.ConsumedBytes, event.TotalBytes, event.ConsumedBytes*100/event.TotalBytes)
case cos.ProgressFailedEvent:
fmt.Printf("\nTransfer Failed: %v", event.Err)
}
}
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
key := "exampleobject"
opt := &cos.ObjectGetOptions{
ResponseContentType: "text/html",
// 使用默认方式查看进度
Listener: &cos.DefaultProgressListener{},
}
// opt 可选，无特殊设置可设为 nil
// 1. 从响应体中获取对象
resp, err := client.Object.Get(context.Background(), key, opt)
if err != nil {
panic(err)
}
ioutil.ReadAll(resp.Body)
resp.Body.Close()
// 2. 下载对象到本地文件
// 使用自定义的进度回调方法
opt.Listener = &SelfListener{}
_, err = client.Object.GetToFile(context.Background(), key, "example.txt", opt)
```

```
if err != nil {
panic(err)
}
}
```

参数说明

```
type ObjectGetOptions struct {
ResponseContentType string
ResponseContentLanguage string
ResponseExpires string
ResponseCacheControl string
ResponseContentDisposition string
ResponseContentEncoding string
Range string
IfModifiedSince string
XCosTrafficLimit int
Listener ProgressListener
}
```

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
localfile	对象保存至本地的文件名称	string	是
ResponseContentType	设置响应头部 Content-Type	string	否
ResponseContentLanguage	设置响应头部 Content-Language	string	否
ResponseExpires	设置响应头部 Content-Expires	string	否
ResponseCacheControl	设置响应头部 Cache-Control	string	否
ResponseContentDisposition	设置响应头部 Content-Disposition	string	否
ResponseContentEncoding	设置响应头部 Content-Encoding	string	否
Range	设置下载文件的范围，格式为 bytes=first-last	string	否
IfModifiedSince	在指定时间后被修改才返回	string	否
XCosTrafficLimit	设置单链接限速	int	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
Listener	进度回调接口	Struct	否

返回结果说明

```
{
'Body': '',
'Accept-Ranges': 'bytes',
'Content-Type': 'application/octet-stream',
'Content-Length': '16807',
'Content-Disposition': 'attachment; filename="filename.jpg"',
'Content-Range': 'bytes 0-16086/16087',
'ETag': '"9a4802d5c99dafe1c04da0a8e7e166bf"',
'Last-Modified': 'Wed, 28 Oct 2014 20:30:00 GMT',
'X-Cos-Request-Id': 'NTg3NzQ3ZmVfYmRjMzVfMzE5N182NzczMQ=='
}
```

通过返回结果Response获取。

```
resp, err := client.Object.Get(context.Background(), key, nil)
body, _ := ioutil.ReadAll(resp.Body)
contentType := resp.Header.Get("Content-Type")
contentLength := resp.Header.Get("Content-Length")
etag := resp.Header.Get("ETag")
reqid := resp.Header.Get("X-Cos-Request-Id")
```

参数名称	参数描述	类型
Body	下载文件的内容	StreamBody
文件元信息	下载文件的元信息，包括 Etag 和 X-Cos-Request-Id 等信息，也会返回设置的文件元信息	string

版权所有：腾讯云计算（北京）有限责任公司

列出对象

最近更新时间：2023-02-08 15:36:40

简介

本文档提供关于列出对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象
GET Bucket Object Versions	查询对象及其历史版本列表	查询存储桶下的部分或者全部对象及其历史版本信息

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

方法原型

```
func (s *BucketService) Get(ctx context.Context, opt *BucketGetOptions) (*BucketGetResult, *Response, error)
```

请求示例1：列出对象列表

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
```

```
t/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
opt := &cos.BucketGetOptions{
Prefix: "test",
MaxKeys: 100,
}
_, _, err := client.Bucket.Get(context.Background(), opt)
if err != nil {
panic(err)
}
}
```

请求示例2：列出目录下对象

对象存储中本身是没有文件夹和目录的概念的，为了满足用户使用习惯，用户可通过分隔符/来模拟“文件夹”。

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"fmt"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
```

```
intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
var marker string
opt := &cos.BucketGetOptions{
Prefix: "folder/", // prefix 表示要查询的文件夹
Delimiter: "/", // deliter 表示分隔符，设置为/表示列出当前目录下的 object，设置为空表示列出所有的 object
MaxKeys: 1000, // 设置最大遍历出多少个对象，一次 listobject 最大支持1000
}
isTruncated := true
for isTruncated {
opt.Marker = marker
v, _, err := client.Bucket.Get(context.Background(), opt)
if err != nil {
fmt.Println(err)
break
}
for _, content := range v.Contents {
fmt.Printf("Object: %v\n", content.Key)
}
// common prefix 表示表示被 delimiter 截断的路径，如 delimter 设置为/，common prefix 则表示所有子目录的路径
for _, commonPrefix := range v.CommonPrefixes {
fmt.Printf("CommonPrefixes: %v\n", commonPrefix)
}
isTruncated = v.IsTruncated // 是否还有数据
marker = v.NextMarker // 设置下次请求的起始 key
}
}
```

参数说明

```
type BucketGetOptions struct {
Prefix string
Delimiter string
EncodingType string
Marker string
MaxKeys int
}
```

参数名称	参数描述	类型	是否必填
Prefix	默认为空，对对象键进行筛选，匹配前缀 prefix 为相同值的 objects	string	否
Delimiter	默认为空，如需模拟文件夹，可设置分隔符 `/`	string	否
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	string	否
Marker	默认以 UTF-8 二进制顺序列出条目，标记返回 objects 的 list 的起点位置	string	否
MaxKeys	最多返回的 objects 数量，默认为最大的1000	int	否

返回结果说明

```
type BucketGetResult struct {
Name string
Prefix string
Marker string
NextMarker string
Delimiter string
MaxKeys int
IsTruncated bool
Contents []Object
CommonPrefixes []string
EncodingType string
}
```

参数名称	参数描述	类型
Name	存储桶名称，格式：BucketName-APPID，例如 examplebucket-1250000000	string
Prefix	默认为空，对对象键进行筛选，匹配前缀 prefix 为相同值的 objects	string
Marker	默认以 UTF-8 二进制顺序列出条目，标记返回 objects 的 list 的起点位置	string

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
NextMarker	当 IsTruncated 为 true 时，标记下一次返回 objects 的 list 的起点位置	string
Delimiter	默认为空，如需模拟文件夹，可设置分隔符 `/`	string
MaxKeys	最多返回的 objects 数量，默认为最大的1000	int
IsTruncated	表示返回的 objects 是否被截断	bool
Contents	包含所有 object 元信息的 list，每个 Object 类型包括 ETag，StorageClass，Key，Owner，LastModified，Size 等信息	[]Object
CommonPrefixes	所有以 Prefix 开头，以 Delimiter 结尾的 Key 被归到同一类	[]string
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	string

查询对象历史版本列表

功能说明

查询开启版本控制的存储桶下的部分或者全部对象。

方法原型

```
func (s *BucketService) GetObjectVersions(ctx context.Context, opt *BucketGetObjectVersionsOptions) (*BucketGetObjectVersionsResult, *Response, error)
```

示例代码：获取所有对象历史版本列表

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
```

```
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
keyMarker := ""
versionIdMarker := ""
isTruncated := true
opt := &cos.BucketGetObjectVersionsOptions{}
for isTruncated {
opt.KeyMarker = keyMarker
opt.VersionIdMarker = versionIdMarker
v, _, err := client.Bucket.GetObjectVersions(context.Background(), opt)
if err != nil {
// ERROR
break
}
for _, vc := range v.Version {
fmt.Printf("Version: %v, %v, %v, %v\n", vc.Key, vc.Size, vc.VersionId, vc.IsLatest)
}
for _, dc := range v.DeleteMarker {
fmt.Printf("DeleteMarker: %v, %v, %v\n", dc.Key, dc.VersionId, dc.IsLatest)
}
keyMarker = v.NextKeyMarker
versionIdMarker = v.NextVersionIdMarker
isTruncated = v.IsTruncated
}
}
```

参数说明

```
type BucketGetObjectVersionsOptions struct {
Prefix string
Delimiter string
EncodingType string
KeyMarker string
VersionIdMarker string
MaxKeys int
}
```

名称	描述	类型	是否必选
prefix	对象键匹配前缀，限定响应中只包含指定前缀的对象键	string	否
delimiter	一个字符的分隔符，用于对对象键进行分组。所有对象键中从 prefix 或从头（如未指定 prefix）到首个 delimiter 之间相同的部分将作为 CommonPrefixes 下的一个 Prefix 节点。被分组的对象键不再出现在后续对象列表中，具体场景和用法可参考下面的实际案例	string	否
encoding-type	规定返回值的编码方式，可选值：url，代表返回的对象键为 URL 编码（百分号编码）后的值，例如“腾讯云”将被编码为 `%E8%85%BE%E8%AE%AF%E4%BA%91`	string	否
key-marker	起始对象键标记，从该标记之后（不含）按照 UTF-8 字典序返回对象版本条目	string	否
version-id-marker	起始版本 ID 标记，从该标记之后（不含）返回对象版本条目，如果上一次 List 结果的 NextVersionIdMarker 为空，则该参数也指定为空	string	否
max-keys	单次返回最大的条目数量，默认值为1000，最大为1000 **注意：**该参数会限制每一次 List 操作返回的最大条目数，COS 在每次 List 操作中将返回不超过 max-keys 所设定数值的条目（CommonPrefixes、Version 和 DeleteMarker 合计），如果单次响应中未列出所有对象，COS 会返回 NextKeyMarker 和 NextVersionIdMarker 节点，其值分别为您下次 List 请求的 key-marker 和 version-id-marker 参数，以便您列出后续版本	int	否

返回结果说明

```
type BucketGetObjectVersionsResult struct {
Name string
EncodingType string
Prefix string
KeyMarker string
VersionIdMarker string
MaxKeys int
Delimiter string
```

版权所有：腾讯云计算（北京）有限责任公司

```
IsTruncated bool
NextKeyMarker string
NextVersionIdMarker string
CommonPrefixes []string
Version []ListVersionsResultVersion
DeleteMarker []ListVersionsResultDeleteMarker
}
type ListVersionsResultVersion struct {
Key string
VersionId string
IsLatest bool
LastModified string
ETag string
Size int
StorageClass string
Owner *Owner
}
type ListVersionsResultDeleteMarker struct {
Key string
VersionId string
IsLatest bool
LastModified string
Owner *Owner
}
```

具体的节点描述如下：

节点名称（关键字）	父节点	描述	类型
ListVersionsResult	无	保存 GET Bucket Object versions 结果的所有信息	Container

Container 节点 ListVersionsResult 的内容：

节点名称（关键字）	父节点	描述	类型
EncodingType	ListVersionsResult	编码格式，对应请求中的 encoding-type 参数，且仅当请求中指定了 encoding-type 参数才会返回该节点	string
Name	ListVersionsResult	存储桶的名称，格式为 `<bucektname-appid>`，例如 `examplebucket-1250000000`	string
Prefix	ListVersionsResult	对象键匹配前缀，对应请求中的 prefix 参数	string

节点名称（关键字）	父节点	描述	类型
KeyMarker	ListVersionsResult	起始对象键标记，从该标记之后（不含）按照 UTF-8 字典序返回对象版本条目，对应请求中的 key-marker 参数	string
VersionIdMarker	ListVersionsResult	起始版本 ID 标记，从该标记之后（不含）返回对象版本条目，对应请求中的 version-id-marker 参数	string
MaxKeys	ListVersionsResult	单次响应返回结果的最大条目数量，对应请求中的 max-keys 参数	integer
IsTruncated	ListVersionsResult	响应条目是否被截断，布尔值，例如 true 或 false	boolean
NextKeyMarker	ListVersionsResult	仅当响应条目有截断（IsTruncated 为 true）才会返回该节点，该节点的值为当前响应条目中的最后一个对象键，当需要继续请求后续条目时，将该节点的值作为下一次请求的 key-marker 参数传入	string
NextVersionIdMarker	ListVersionsResult	仅当响应条目有截断（IsTruncated 为 true）才会返回该节点，该节点的值为当前响应条目中的最后一个对象的版本 ID，当需要继续请求后续条目时，将该节点的值作为下一次请求的 version-id-marker 参数传入。该节点的值可能为空，此时下一次请求的 version-id-marker 参数也需要指定为空。	string
Delimiter	ListVersionsResult	分隔符，对应请求中的 delimiter 参数，且仅当请求中指定了 delimiter 参数才会返回该节点	string
CommonPrefixes	ListVersionsResult	从 prefix 或从头（如未指定 prefix）到首个 delimiter 之间相同的部分，定义为 Common Prefix。仅当请求中指定了 delimiter 参数才有可能返回该节点	Container
Version	ListVersionsResult	对象版本条目	Container
DeleteMarker	ListVersionsResult	对象删除标记条目	Container

Container 节点 CommonPrefixes 的内容：

节点名称（关键字）	父节点	描述	类型

节点名称（关键字）	父节点	描述	类型
Prefix	ListVersionsResult.CommonPrefixes	单条 Common Prefix 的前缀	string

Container 节点 Version 的内容：

节点名称（关键字）	父节点	描述	类型
Key	ListVersionsResult.Version	对象键	string
VersionId	ListVersionsResult.Version	对象的版本 ID 当未启用版本控制时，该节点的值为空字符串当启用版本控制时，启用版本控制之前的对象，其版本 ID 为 null 当暂停版本控制时，新上传的对象其版本 ID 为 null，且同一个对象最多只存在一个版本 ID 为 null 的对象版本	string
IsLatest	ListVersionsResult.Version	当前版本是否为该对象的最新版本	boolean
LastModified	ListVersionsResult.Version	当前版本的最后修改时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	date
ETag	ListVersionsResult.Version	对象的实体标签（Entity Tag），是对象被创建时标识对象内容的信息标签，可用于检查对象的内容是否发生变化 例如"8e0b617ca298a564c3331da28dcb50df"。此头部并不一定返回对象的 MD5 值，而是根据对象上传和加密方式而有所不同	string
Size	ListVersionsResult.Version	对象大小，单位为 Byte	integer
StorageClass	ListVersionsResult.Version	对象存储类型。枚举值请参见 存储类型 文档，例如 STANDARD_IA，ARCHIVE	Enum
StorageTier	ListVersionsResult.Version	当对象存储类型为智能分层存储时，指示对象当前所处的存储层，枚举值：FREQUENT（标准层），INFREQUENT（低频层）。仅当 StorageClass 为 INTELLIGENT_TIERING（智能分层）时才会返回该节点	Enum
Owner	ListVersionsResult.Version	对象持有者信息	Container

Container 节点 Version.Owner 的内容：

节点名称（关键字）	父节点	描述	类型

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型
ID	ListVersionsResult.Version.Owner	对象持有者的 APPID	string
DisplayName	ListVersionsResult.Version.Owner	对象持有者的名称	string

Container 节点 DeleteMarker 的内容：

节点名称（关键字）	父节点	描述	类型
Key	ListVersionsResult.DeleteMarker	对象键	string
VersionId	ListVersionsResult.DeleteMarker	对象的删除标记的版本 ID	string
IsLatest	ListVersionsResult.DeleteMarker	当前删除标记是否为该对象的最新版本	boolean
LastModified	ListVersionsResult.DeleteMarker	当前删除标记的删除时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	date
Owner	ListVersionsResult.DeleteMarker	对象持有者信息	Container

Container 节点 DeleteMarker.Owner 的内容：

节点名称（关键字）	父节点	描述	类型
ID	ListVersionsResult.DeleteMarker.Owner	对象持有者的 APPID	string
DisplayName	ListVersionsResult.DeleteMarker.Owner	对象持有者的名称	string

删除对象

最近更新时间：2023-02-08 15:36:40

简介

本文档提供关于对象的删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

删除单个对象

功能说明

在 Bucket 中删除指定 Object （对象/文件夹）。

方法原型

```
func (s *ObjectService) Delete(ctx context.Context, key string) (*Response, erro
r)
```

请求示例1：删除对象

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
t/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
```

```
m")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.int
l.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权
限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/59
8/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.i
ntl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最
小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/produc
t/598/37140
},
})
key := "exampleobject"
_, err := client.Object.Delete(context.Background(), key)
if err != nil {
panic(err)
}
}
```

请求示例2：删除文件夹

该请求不会删除文件夹内的对象，只会删除指定的 key。

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
t/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
m")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
```

```
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
key := "folder/"
_, err := client.Object.Delete(context.Background(), key)
if err != nil {
panic(err)
}
}
```

参数说明

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是

删除多个对象

功能说明

在 Bucket 中删除多个 Object （文件/对象）。单次请求最大支持批量删除1000个 Object。

方法原型

```
func (s *ObjectService) DeleteMulti(ctx context.Context, opt *ObjectDeleteMultiOptions) (*ObjectDeleteMultiResult, *Response, error)
```

请求示例1：删除多个指定对象

```
package main
import (
```

```
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
var objects []string
objects = append(objects, []string{"a", "b", "c"}...)
obs := []cos.Object{}
for _, v := range objects {
obs = append(obs, cos.Object{Key: v})
}
opt := &cos.ObjectDeleteMultiOptions{
Objects: obs,
// 布尔值，这个值决定了是否启动 Quiet 模式
// 值为 true 启动 Quiet 模式，值为 false 则启动 Verbose 模式，默认值为 false
// Quiet: true,
}
_, _, err := client.Object.DeleteMulti(context.Background(), opt)
if err != nil {
panic(err)
}
}
```

请求示例2：删除文件夹及其文件

COS 上的文件夹概念是以 '/' 分隔对象名，形成类似文件系统的路径，从而模拟出来的。所以删除文件夹的操作，在 COS 上相当于删除一批有着同样前缀的对象。例如：文件夹 'prefix/' ，代表的是以 'prefix/' 为前缀的所有对象，所以删除 'prefix/'，意味着删除以 'prefix/' 为前缀的所有对象。
目前 COS Go SDK 没有提供一个接口去支持这样的操作，但是可以通过基本操作的组合，达到同样的效果。

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
dir := "exampledir/"
var marker string
opt := &cos.BucketGetOptions{
Prefix: dir,
MaxKeys: 1000,
}
```

```
isTruncated := true
for isTruncated {
opt.Marker = marker
v, _, err := client.Bucket.Get(context.Background(), opt)
if err != nil {
// Error
break
}
for _, content := range v.Contents {
_, err = client.Object.Delete(context.Background(), content.Key)
if err != nil {
// Error
}
}
isTruncated = v.IsTruncated
marker = v.NextMarker
}
}
```

参数说明

```
type ObjectDeleteMultiOptions struct {
Quiet bool
Objects []Object
}
// Object保存了对象的元信息
type Object struct {
Key string
// 其他参数与这个API不相关
}
```

参数名称	参数描述	类型	是否必填
Quiet	布尔值，这个值决定了是否启动 Quiet 模式。对于响应结果，COS 提供 Verbose 和 Quiet 两种模式： • Verbose 模式将返回每个 Object 的删除结果 • Quiet 模式只返回报错的 Object 信息 值为 true 启动 Quiet 模式，值为 false 则启动 Verbose 模式，默认值为 false	Boolean	否
Objects	说明每个将要删除的目标 Object 信息	Container	是
Key	目标 Object 文件名称	String	是

返回结果说明

上传文件的属性：

```
// ObjectDeleteMultiResult 保存 DeleteMulti 的结果
type ObjectDeleteMultiResult struct {
DeletedObjects []Object
Errors []struct {
Key string
Code string
Message string
}
}
```

参数名称	参数描述	类型
DeletedObjects	说明每个将要删除的目标 Object 信息	Container
Errors	说明本次删除的失败 Object 信息	Container
Key	删除失败的 Object 的名称	string
Code	删除失败的错误代码	string
Message	删除失败的错误信息	string

版权所有：腾讯云计算（北京）有限责任公司

对象访问 URL

最近更新时间：2023-02-08 15:36:40

简介

本文档提供获取对象访问 URL 的代码示例。

获取对象访问 URL

功能说明

获取对象访问 URL 用于匿名下载或分发。

> 注意：
> COS Go SDK 版本需要大于等于 v0.7.26。

方法原型

```
func (s *ObjectService) GetObjectURL(key string) *url.URL
```

请求示例

```
package main
import (
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
```

```
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
key := "exampleobject"
ourl := client.Object.GetObjectURL(key)
fmt.Println(ourl)
}
```

参数说明

参数名称	参数描述	类型	是否必填
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是

返回结果说明

该方法返回值为对象访问的 URL。

恢复归档对象

最近更新时间：2023-02-08 15:36:40

简介

本文档提供关于恢复归档对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
POST Object restore	恢复归档对象	将归档类型的对象取回访问

恢复归档对象

功能说明

将归档类型的对象取回访问（POST Object restore）。

方法原型

```
func (s *ObjectService) PostRestore(ctx context.Context, key string, opt *ObjectRestoreOptions) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
```

```
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
key := "example_restore"
f, err := os.Open("/test")
if err != nil {
panic(err)
}
opt := &cos.ObjectPutOptions{
ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
ContentType: "text/html",
XCosStorageClass: "ARCHIVE", //归档类型
},
ACLHeaderOptions: &cos.ACLHeaderOptions{
// 如果不是必要操作，建议上传文件时不要给单个文件设置权限，避免达到限制。若不设置默认继承桶的权限。
XCosACL: "private",
},
}
// 归档直传
_, err = client.Object.Put(context.Background(), key, f, opt)
if err != nil {
panic(err)
}
opts := &cos.ObjectRestoreOptions{
Days: 2,
Tier: &cos.CASJobParameters{
// Standard, Expedited and Bulk
Tier: "Expedited",
},
}
// 归档恢复
_, err = client.Object.PostRestore(context.Background(), key, opts)
if err != nil {
panic(err)
```

```
}
}
```

参数说明

```
type ObjectRestoreOptions struct {
Days int
Tier *CASJobParameters
}
type CASJobParameters struct {
Tier string
}
```

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
ObjectRestoreOptions	描述取回的临时文件的规则	struct	是
Days	描述临时文件的过期时间	int	是
CASJobParameters	描述恢复类型的配置信息	struct	否
Tier	描述取回临时文件的模式。 • 若恢复的是归档存储类型数据，可选值为 Expedited、Standard、Bulk，分别对应快速取回模式、标准取回模式以及批量取回模式这三种模式； • 若恢复的是深度归档存储类型数据，则可选值为 Standard、Bulk。	string	否

查询对象元数据

最近更新时间：2023-02-08 15:36:40

简介

本文档提供关于查询对象元数据操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

查询对象元数据

功能说明

查询 Object 的 Meta 信息（HEAD Object）。

方法原型

```
func (s *ObjectService) Head(ctx context.Context, key string, opt *ObjectHeadOpti
ons) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
t/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
m")
b := &cos.BaseURL{BucketURL: u}
```

```
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
key := "exampleobject"
_, err := client.Object.Head(context.Background(), key, nil)
if err != nil {
panic(err)
}
}
```

参数说明

```
type ObjectHeadOptions struct {
IfModifiedSince string
}
```

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
IfModifiedSince	在指定时间后被修改才返回	string	否

返回结果说明

```
{
'Content-Type': 'application/octet-stream',
'Content-Length': '16807',
'ETag': '"9a4802d5c99dafe1c04da0a8e7e166bf"',
'Last-Modified': 'Wed, 28 Oct 2014 20:30:00 GMT',
```

版权所有：腾讯云计算（北京）有限责任公司

```
'X-Cos-Request-Id': 'NTg3NzQ3ZmVfYmRjMzVfMzE5N182NzczMQ=='
}
```

通过返回结果 Response 获取。

```
resp, err := client.Object.Head(context.Background(), key, nil)
contentType := resp.Header.Get("Content-Type")
contentLength := resp.Header.Get("Content-Length")
etag := resp.Header.Get("ETag")
reqid := resp.Header.Get("X-Cos-Request-Id")
```

参数名称	参数描述	类型
文件元信息	获取文件的元信息，包括 Etag 和 X-Cos-Request-Id 等信息，也会包含设置的文件元信息	string

判断对象是否存在

最近更新时间：2023-02-08 15:36:40

简介

本文档提供快捷查询存储桶中某个对象是否存在的示例代码。

示例代码实际调用了 COS API HeadObject，是该接口的简化版。

HeadObject 除了检查对象是否存在，主要功能为返回对象元数据。包含了 HeadObject 完整功能的 SDK 接口，可参见 查询对象元数据。

检查对象是否存在

功能说明

您可以通过 SDK 提供的快捷接口来检查对象是否存在。

> 注意：
> COS Go SDK 版本需要大于等于 v0.7.33。

方法原型

```
func (s *ObjectService) IsExist(ctx context.Context, key string, id ...string) (i
sExist bool, err error)
```

示例代码

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
```

```
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
key := "exampleobject"
ok, err := client.Object.IsExist(context.Background(), key)
if err == nil && ok {
fmt.Printf("object exists\n")
} else if err != nil {
fmt.Printf("head object failed: %v\n", err)
} else {
fmt.Printf("object does not exist\n")
}
}
```

参数说明

参数名称	描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
id	对象 VersionId。	string	否

返回结果说明

参数名称	描述	类型

参数名称	描述	类型
isExist	对象是否存在	bool
err	请求是否成功	struct

版权所有：腾讯云计算（北京）有限责任公司

预签名 URL

最近更新时间：2023-02-08 15:42:53

简介

Go SDK 提供获取请求预签名 URL 接口，详细操作请查看本文示例。

说明：

- 建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。

获取请求预签名 URL

```
func (s *ObjectService) GetPresignedURL(ctx context.Context, httpMethod, key, ak, sk string, expired time.Duration, opt interface{}, signHost ...bool) (*url.URL, error)
```

参数说明

```
type PresignedURLOptions struct {
Query *url.Values
Header *http.Header
}
```

参数名称	类型	描述
httpMethod	string	HTTP 请求方法
key	string	对象键（Key）是对象在存储桶中的唯一标识，详情请参见 对象键（**注意：用户无需对 key 进行编码操作**）
ak	string	SecretId
sk	string	SecretKey

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述
expired	time.Duration	签名有效时长
opt	interface{}	扩展项，建议填写 *PresignedURLOptions 类型的参数。可填nil
PresignedURLOptions	struct	指定签入的请求参数和请求头部。
Query	struct	签名中要签入的请求参数。
Header	struct	签名中要签入的请求头部。
signHost	bool	可选参数，默认为true，获取签名时是否签入Header Host；您也可以选择不签入 Header Host，但可能导致请求失败或安全漏洞。

永久密钥预签名请求示例

上传请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"strings"
"time"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/59
```

```
8/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
ak := "SECRETID"
sk := "SECRETKEY"
name := "exampleobject"
ctx := context.Background()
f := strings.NewReader("test")
// 1. 通过普通方式上传对象
_, err := client.Object.Put(ctx, name, f, nil)
if err != nil {
panic(err)
}
// 获取预签名 URL
presignedURL, err := client.Object.GetPresignedURL(ctx, http.MethodPut, name, ak, sk, time.Hour, nil)
if err != nil {
panic(err)
}
// 2. 通过预签名方式上传对象
data := "test upload with presignedURL"
f = strings.NewReader(data)
req, err := http.NewRequest(http.MethodPut, presignedURL.String(), f)
if err != nil {
panic(err)
}
// 用户可自行设置请求头部
req.Header.Set("Content-Type", "text/html")
_, err = http.DefaultClient.Do(req)
if err != nil {
panic(err)
}
}
```

下载请求示例

```
package main
import (
"bytes"
"context"
"errors"
```

```
"github.com/tencentyun/cos-go-sdk-v5"
"io/ioutil"
"net/http"
"net/url"
"os"
"time"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
ak := "SECRETID"
sk := "SECRETKEY"
name := "exampleobject"
ctx := context.Background()
// 1. 通过普通方式下载对象
resp, err := client.Object.Get(ctx, name, nil)
if err != nil {
panic(err)
}
bs, _ := ioutil.ReadAll(resp.Body)
resp.Body.Close()
// 获取预签名 URL
presignedURL, err := client.Object.GetPresignedURL(ctx, http.MethodGet, name, ak, sk, time.Hour, nil)
if err != nil {
panic(err)
```

```
}
// 2. 通过预签名 URL下载对象
resp2, err := http.Get(presignedURL.String())
if err != nil {
panic(err)
}
bs2, _ := ioutil.ReadAll(resp2.Body)
resp2.Body.Close()
if bytes.Compare(bs2, bs) != 0 {
panic(errors.New("content is not consistent"))
}
}
```

临时密钥预签名请求示例

上传请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"time"
"strings"
)
// 通过 tag 的方式，用户可以将请求参数或者请求头部放进签名中。
type URLToken struct {
SessionToken string `url:"x-cos-security-token,omitempty" header:"-"`
}
func main() {
// 替换成您的临时密钥
tak := os.Getenv("SECRETID") // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
tsk := os.Getenv("SECRETKEY") // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
token := &URLToken{
SessionToken: "<token>",
}
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
```

```
m")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{})
name := "exampleobject"
ctx := context.Background()
// 方法1 通过 PresignedURLOptions 设置 x-cos-security-token
// PresignedURLOptions 提供用户添加请求参数和请求头部
opt := &cos.PresignedURLOptions{
Query: &url.Values{},
Header: &http.Header{},
}
opt.Query.Add("x-cos-security-token", "<token>")
// 获取预签名
presignedURL, err := c.Object.GetPresignedURL(ctx, http.MethodPut, name, tak, ts
k, time.Hour, opt)
if err != nil {
fmt.Printf("Error: %v\n", err)
return
}
// 通过预签名方式上传对象
data := "test upload with presignedURL"
f := strings.NewReader(data)
req, err := http.NewRequest(http.MethodPut, presignedURL.String(), f)
if err != nil {
fmt.Printf("Error: %v\n", err)
}
_, err = http.DefaultClient.Do(req)
if err != nil {
fmt.Printf("Error: %v\n", err)
}
// 方法2 通过 tag 设置 x-cos-security-token
// 获取预签名
presignedURL, err = c.Object.GetPresignedURL(ctx, http.MethodPut, name, tak, tsk,
time.Hour, token)
if err != nil {
fmt.Printf("Error: %v\n", err)
return
}
f = strings.NewReader(data)
req, err = http.NewRequest(http.MethodPut, presignedURL.String(), f)
if err != nil {
fmt.Printf("Error: %v\n", err)
}
_, err = http.DefaultClient.Do(req)
if err != nil {
fmt.Printf("Error: %v\n", err)
```

```
}
}
```

下载请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"time"
)
// 通过 tag 的方式，用户可以将请求参数或者请求头部放进签名中。
type URLToken struct {
SessionToken string `url:"x-cos-security-token,omitempty" header:"-"`
}
func main() {
// 替换成您的临时密钥
tak := os.Getenv("SECRETID") // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
tsk := os.Getenv("SECRETKEY") // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
token := &URLToken{
SessionToken: "<token>",
}
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{})
name := "exampleobject"
ctx := context.Background()
// 方法1 通过 PresignedURLOptions 设置 x-cos-security-token
// PresignedURLOptions 提供用户添加请求参数和请求头部
opt := &cos.PresignedURLOptions{
Query: &url.Values{},
Header: &http.Header{},
}
opt.Query.Add("x-cos-security-token", "<token>")
// 获取预签名
presignedURL, err := c.Object.GetPresignedURL(ctx, http.MethodGet, name, tak, tsk, time.Hour, opt)
```

```
if err != nil {
fmt.Printf("Error: %v\n", err)
return
}
// 通过预签名访问对象
resp, err := http.Get(presignedURL.String())
if err != nil {
fmt.Printf("Error: %v\n", err)
}
defer resp.Body.Close()
fmt.Println(presignedURL.String())
fmt.Printf("resp:%v\n", resp)
// 方法2 通过 tag 设置 x-cos-security-token
// 获取预签名
presignedURL, err = c.Object.GetPresignedURL(ctx, http.MethodGet, name, tak, tsk,
time.Hour, token)
if err != nil {
fmt.Printf("Error: %v\n", err)
return
}
// 通过预签名访问对象
resp, err = http.Get(presignedURL.String())
if err != nil {
fmt.Printf("Error: %v\n", err)
}
defer resp.Body.Close()
fmt.Println(presignedURL.String())
fmt.Printf("resp:%v\n", resp)
}
```

自定义域名生成预签名示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"time"
)
func main() {
// 替换成您的密钥
```

```
tak := os.Getenv("SECRETID") // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
tsk := os.Getenv("SECRETKEY") // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 修改成用户的自定义域名
u, _ := url.Parse("https://<自定义域名>")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{})
name := "exampleobject"
ctx := context.Background()
// 获取预签名
presignedURL, err := c.Object.GetPresignedURL(ctx, http.MethodGet, name, tak, tsk, time.Hour, nil)
if err != nil {
fmt.Printf("Error: %v\n", err)
return
}
// 通过预签名访问对象
resp, err := http.Get(presignedURL.String())
if err != nil {
fmt.Printf("Error: %v\n", err)
}
defer resp.Body.Close()
fmt.Println(presignedURL.String())
fmt.Printf("resp:%v\n", resp)
}
```

添加请求参数或请求头部

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"time"
)
func main() {
// 替换成您的临时密钥
tak := os.Getenv("SECRETID") // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指
```

```
引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
tsk := os.Getenv("SECRETKEY") // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{})
name := "exampleobject"
ctx := context.Background()
// PresignedURLOptions 提供用户添加请求参数和请求头部
opt := &cos.PresignedURLOptions{
// http 请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此 HTTP 请求的参数
Query: &url.Values{},
// http 请求头部，传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的 HTTP 请求头部
Header: &http.Header{},
}
// 添加请求参数，返回的预签名 url 将包含该参数
opt.Query.Add("x-cos-security-token", "<token>")
// 添加请求头部，返回的预签名 url 只是将请求头部设置到签名里，请求时还需要自行设置对应的 header。
opt.Header.Add("Content-Type", "text/html")
// SDK 默认签入 Header Host，不传递 signHost 参数或者 SignHost = true 时，表示签入 Header Host。
// signHost = false 时，表示不签入 Header Host，不签入 Header Host 可能导致请求失败或安全漏洞。
var signHost bool = true
// 获取预签名，签名中携带host。
presignedURL, err := c.Object.GetPresignedURL(ctx, http.MethodPut, name, tak, tsk, time.Hour, opt, signHost)
if err != nil {
fmt.Printf("Error: %v\n", err)
return
}
// 通过预签名访问对象
req, _ := http.NewRequest(http.MethodPut, presignedURL.String(), strings.NewReader("test"))
// 请求时需要设置对应 header
req.Header.Set("Content-Type", "text/html")
resp, err := http.DefaultClient.Do(req)
if err != nil {
fmt.Printf("Error: %v\n", err)
}
defer resp.Body.Close()
fmt.Println(presignedURL.String())
```

```
fmt.Printf("resp:%v\n", resp)
}
```

版权所有：腾讯云计算（北京）有限责任公司

客户端加密

最近更新时间：2023-04-12 14:28:24

简介

Go SDK 支持客户端加密，将文件加密后再进行上传，并在下载时进行解密，适用于存储敏感数据的客户。

客户端加密只支持以下方式：
KMS 服务托管密钥：用户只需提供 KMS 服务的用户主密钥 ID（即 CMK ID）给 SDK。使用这种方式需要用户开通 KMS 服务，更多 KMS 服务信息参见 腾讯密钥管理系统。

注意事项

在对加密数据进行复制或者迁移时，您需要对加密元信息的完整性和正确性负责。因您维护不当导致加密元信息出错或丢失，从而导致加密数据无法解密所引起的一切损失和后果均由您自行承担。

上传加密流程

- 每次上传一个文件对象前，将随机生成一个对称加密密钥。随机生成的密钥通过 KMS 服务进行加密，将加密后的结果 base64 编码存储在对象的元数据中。

2. 进行文件对象的上传，上传时在内存使用 AES256 算法加密。

下载解密流程

- 获取文件元数据中加密必要的信息，Base64 解码后使用 KMS 服务（或者用户提供的密钥）进行解密，得到当时加密数据的密钥。

2. 使用密钥对下载输入流进行使用 AES256 解密，得到解密后的文件输入流。

请求示例

完整的示例请参见 客户端加密 - KMS 加密完整示例。

版权所有：腾讯云计算（北京）有限责任公司

请求示例1：简单上传和下载

```
// 创建原始客户端
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})

// Case1 上传对象
name := "test/example"

// 该标识信息唯一确认一个主加密密钥，解密时，需要传入相同的标识信息
// KMS加密时，该信息设置成 EncryptionContext，最大支持1024字符，如果 Encrypt 指定了该参数，则在 Decrypt 时需要提供同样的参数
materialDesc := make(map[string]string)
//materialDesc["desc"] = "material information of your master encrypt key"

// 创建 KMS 客户端
kmsclient, _ := coscrypto.NewKMSClient(c.GetCredential(), "ap-guangzhou")
// 创建 KMS 主加密密钥，标识信息 materialDesc 和主密钥一一对应
kmsID := os.Getenv("KMSID")
masterCipher, _ := coscrypto.CreateMasterKMS(kmsclient, kmsID, materialDesc)
// 创建加密客户端
client := coscrypto.NewCryptoClient(c, masterCipher)

contentLength := 1024*1024*10 + 1
originData := make([]byte, contentLength)
_, err := rand.Read(originData)
f := bytes.NewReader(originData)
// 加密上传
_, err = client.Object.Put(context.Background(), name, f, nil)
log_status(err)

// 解密下载
resp, err := client.Object.Get(context.Background(), name, nil)
log_status(err)
defer resp.Body.Close()
decryptedData, _ := ioutil.ReadAll(resp.Body)
```

```
if bytes.Compare(decryptedData, originData[rangeStart:rangeEnd+1]) != 0 {
fmt.Println("Error: encryptedData != originData")
}
```

请求示例2：上传文件和下载对象到文件

```
// 创建原始客户端
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})

// Case1 上传对象
name := "test/example"

// 该标识信息唯一确认一个主加密密钥，解密时，需要传入相同的标识信息
// KMS 加密时，该信息设置成 EncryptionContext，最大支持1024字符，如果 Encrypt 指定了该参数，则在 Decrypt 时需要提供同样的参数
materialDesc := make(map[string]string)
//materialDesc["desc"] = "material information of your master encrypt key"

// 创建KMS客户端
kmsclient, _ := coscrypto.NewKMSClient(c.GetCredential(), "ap-guangzhou")
// 创建 KMS 主加密密钥，标识信息 materialDesc 和主密钥一一对应
kmsID := os.Getenv("KMSID")
masterCipher, _ := coscrypto.CreateMasterKMS(kmsclient, kmsID, materialDesc)
// 创建加密客户端
client := coscrypto.NewCryptoClient(c, masterCipher)

// 加密上传
_, err = client.Object.PutFromFile(context.Background(), name, filepath, nil)
if err != nil {
//ERROR
}

// 解密下载
_, err = client.Object.GetToFile(context.Background(), name, "./test.download", n
```

```
il)
if err != nil {
//ERROR
}
```

请求示例3：分块上传

```
// 创建原始客户端
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})

// Case1 上传对象
name := "test/example"

// 该标识信息唯一确认一个主加密密钥，解密时，需要传入相同的标识信息
// KMS 加密时，该信息设置成 EncryptionContext，最大支持1024字符，如果 Encrypt 指定了该参数，则在 Decrypt 时需要提供同样的参数
materialDesc := make(map[string]string)
//materialDesc["desc"] = "material information of your master encrypt key"

// 创建 KMS 客户端
kmsclient, _ := coscrypto.NewKMSClient(c.GetCredential(), "ap-guangzhou")
// 创建 KMS 主加密密钥，标识信息 materialDesc 和主密钥一一对应
kmsID := os.Getenv("KMSID")
masterCipher, _ := coscrypto.CreateMasterKMS(kmsclient, kmsID, materialDesc)
// 创建加密客户端
client := coscrypto.NewCryptoClient(c, masterCipher)

filepath := "test"
stat, err := os.Stat(filepath)
if err != nil {
// ERROR
}
contentLength := stat.Size()
```

```
// 分块上传
// 每个分块需要16字节对齐，且大于等于1MB
partSize := (contentLength / 16 / 3) * 16
if partSize < int64(1024*1024) {
partSize = int64(1024*1024)
}
cryptoCtx := coscrypto.CryptoContext{
DataSize: contentLength,
// 每个分块需要16字节对齐
PartSize: partSize,
}
// 切分数据
_, chunks, _, err := cos.SplitFileIntoChunks(filepath, cryptoCtx.PartSize)
if err != nil {
// ERROR
}

// init mulitupload
v, _, err := client.Object.InitiateMultipartUpload(context.Background(), name, ni
l, &cryptoCtx)
if err != nil {
// ERROR
}

// part upload
optcom := &cos.CompleteMultipartUploadOptions{}
for _, chunk := range chunks {
fd, err := os.Open(filepath)
if err != nil {
// ERROR
}
opt := &cos.ObjectUploadPartOptions{
ContentLength: chunk.Size,
}
fd.Seek(chunk.OffSet, os.SEEK_SET)
resp, err := client.Object.UploadPart(context.Background(), name, v.UploadID, chu
nk.Number, cos.LimitReadCloser(fd, chunk.Size), opt, &cryptoCtx)
if err != nil {
// ERROR
}
optcom.Parts = append(optcom.Parts, cos.Object{
PartNumber: chunk.Number, ETag: resp.Header.Get("ETag"),
})
}
// complete upload
_, _, err = client.Object.CompleteMultipartUpload(context.Background(), name, v.U
ploadID, optcom)
```

```
if err != nil {
// ERROR
}

_, err = client.Object.GetToFile(context.Background(), name, "test.download", nil
)
if err != nil {
// ERROR
}
```

版权所有：腾讯云计算（北京）有限责任公司

服务端加密

最近更新时间：2023-04-12 14:28:24

如果需要对上传的对象进行加密，我们支持以下加密方式。

使用 COS 托管加密密钥的服务端加密（SSE-COS）保护数据

由腾讯云 COS 托管主密钥和管理数据。COS 会帮助您在数据写入数据中心时自动加密，并在您取用该数据时自动解密。目前支持使用 COS 主密钥对数据进行 AES-256 加密。

GO SDK 通过设置 `ObjectPutHeaderOptions` 的 `XCosServerSideEncryption` 成员来完成。

```
// 下载对象，下载对象需要用户提供密钥 // 下载对象，下载对象需要用户提供密钥 package main

import (
"context"
"errors"
"io/ioutil"
"net/http"
"net/url"
"os"
"strings"

"github.com/tencentyun/cos-go-sdk-v5"
"github.com/tencentyun/cos-go-sdk-v5/debug"
)

func main() {
// 替换成您的<Bucketname-APPID>
u, _ := url.Parse("https://<Bucketname-APPID>.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考见 https://cloud.tencent.com/document/product/598/37140
Transport: &debug.DebugRequestTransport{
RequestHeader: true,
// Notice when put a large file and set need the request body, might happend out of memory error.
RequestBody: false,
ResponseHeader: true,
```

```
ResponseBody: true,
},
},
})
opt := &cos.ObjectPutOptions{
ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
ContentType: "text/html",
XCosServerSideEncryption: "AES256",
},
ACLHeaderOptions: &cos.ACLHeaderOptions{},
}
name := "PutFromGoWithSSE-COS"
content := "Put Object From Go With SSE-COS"
f := strings.NewReader(content)
_, err := c.Object.Put(context.Background(), name, f, opt)
if err != nil {
panic(err)
}
// 下载对象
getopt := &cos.ObjectGetOptions{}
var resp *cos.Response
resp, err = c.Object.Get(context.Background(), name, getopt)
if err != nil {
panic(err)
}
// 验证
bodyBytes, _ := ioutil.ReadAll(resp.Body)
bodyContent := string(bodyBytes)
if bodyContent != content {
panic(errors.New("Content inconsistency"))
}
}
```

使用客户提供的加密密钥的服务端加密 （SSE-C）保护数据

加密密钥由用户自己提供，用户在上传对象时，COS 将使用用户提供的加密密钥对用户的数据进行 AES-256 加密。GO SDK 通过设置 `ObjectPutHeaderOptions` 的 `XCosSSECustomer*` 成员来完成。

注意：

- 该加密所运行的服务需要使用 HTTPS 请求。
- customerKey：用户提供的密钥，传入一个32字节的字符串，支持数字、字母、字符的组合，不支持中文。

- 如果上传的源文件调用了该方法，那么在使用 GET（下载）、HEAD（查询）时对源对象操作的时候也要调用该方法。

```
package main

import (
"context"
"net/url"
"os"
"strings"
"errors"
"io/ioutil"
"net/http"

"github.com/tencentyun/cos-go-sdk-v5"
"github.com/tencentyun/cos-go-sdk-v5/debug"
)

func main() {
// 替换成您的<Bucketname-APPID>
u, _ := url.Parse("https://<Bucketname-APPID>.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
Transport: &debug.DebugRequestTransport{
RequestHeader: true,
// Notice when put a large file and set need the request body, might happend out of memory error.
RequestBody: false,
ResponseHeader: true,
ResponseBody: true,
},
},
})
opt := &cos.ObjectPutOptions{
ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
ContentType: "text/html",
XCosSSECustomerAglo: "AES256",
XCosSSECustomerKey: "MDEyMzQ1Njc4OUFCQ0RFRjAxMjM0NTY3ODlBQkNERUY=",
```

```
XCosSSECustomerKeyMD5: "U5L61r7jcwdNvT7frmUG8g==",
},
ACLHeaderOptions: &cos.ACLHeaderOptions{},
}
name := "PutFromGoWithSSE-C"
content := "Put Object From Go With SSE-C"
f := strings.NewReader(content)
_, err := c.Object.Put(context.Background(), name, f, opt)
if err != nil {
panic(err)
}
// 下载对象，下载对象需要用户提供密钥
getopt := &cos.ObjectGetOptions{
XCosSSECustomerAglo: "AES256",
XCosSSECustomerKey: "MDEyMzQ1Njc4OUFCQ0RFRjAxMjM0NTY3ODlBQkNERUY=",
XCosSSECustomerKeyMD5: "U5L61r7jcwdNvT7frmUG8g==",
}
var resp *cos.Response
resp, err = c.Object.Get(context.Background(), name, getopt)
if err != nil {
panic(err)
}
// 验证
bodyBytes, _ := ioutil.ReadAll(resp.Body)
bodyContent := string(bodyBytes)
if bodyContent != content {
panic(errors.New("Content inconsistency"))
}
}
```

异地容灾
版本控制

最近更新时间：2023-02-08 14:38:13

简介

本文档提供关于版本控制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

设置版本控制

功能说明

设置指定存储桶的版本控制功能（PUT Bucket versioning）。

方法原型

```
func (s *BucketService) PutVersioning(ctx context.Context, opt *BucketPutVersionOptions) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
```

```
t/436/6224 。
u, _ := url.Parse("https://examplebucket-12500000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
opt := &cos.BucketPutVersionOptions{
// Enabled 或者 Suspended，版本控制配置一旦开启就不能删除，只能暂停
Status: "Enabled",
}
_, err := client.Bucket.PutVersioning(context.Background(), opt)
if err != nil {
panic(err)
}
}
```

参数说明

```
type BucketPutVersionOptions struct {
Status string
}
```

参数名称	描述	类型
BucketPutVersionOptions	版本控制策略	struct
Status	说明版本是否开启，枚举值：Suspended（暂停版本控制）、Enabled（开启版本控制）	string

查询版本控制

功能说明

查询指定存储桶的版本控制信息（GET Bucket versioning）。

方法原型

```
func (s *BucketService) GetVersioning(ctx context.Context) (*BucketGetVersionResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-12500000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, _, err := client.Bucket.GetVersioning(context.Background())
if err != nil {
panic(err)
```

```
}
}
```

返回结果说明

```
type BucketGetVersionResult struct {
Status string
}
```

参数名称	描述	类型
BucketGetVersionResult	版本控制策略	struct
Status	说明版本是否开启，枚举值：Suspended（暂停版本控制）、Enabled（开启版本控制）	string

存储桶复制

最近更新时间：2023-02-08 14:38:13

简介

本文档提供关于存储桶复制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket replication	设置存储桶复制	对已启用版本控制的存储桶配置存储桶复制规则
GET Bucket replication	查询存储桶复制	查询存储桶的存储桶复制规则
DELETE Bucket replication	删除存储桶复制	删除存储桶的存储桶复制规则

设置存储桶复制

功能说明

设置指定存储桶的存储桶复制规则（PUT Bucket replication）。

方法原型

```
func (s *BucketService) PutBucketReplication(ctx context.Context, opt *PutBucketReplicationOptions) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
```

```
t/436/6224 。
u, _ := url.Parse("https://examplebucket-12500000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
opt := &cos.PutBucketReplicationOptions{
// qcs::cam::uin/[UIN]:uin/[Subaccount]
Role: "qcs::cam::uin/100000000001:uin/100000000001",
Rule: []cos.BucketReplicationRule{
{
ID: "1",
// Enabled or Disabled
Status: "Enabled",
Destination: &cos.ReplicationDestination{
// qcs::cos:[Region]::[Bucketname-Appid]
Bucket: "qcs::cos:ap-beijing::destinationbucket-1250000000",
},
},
},
}
_, err := client.Bucket.PutBucketReplication(context.Background(), opt)
if err != nil {
panic(err)
}
}
```

参数说明

```
type PutBucketReplicationOptions struct {
Role string
Rule []BucketReplicationRule
}
```

```
type BucketReplicationRule struct {
ID string
Status string
Prefix string
Destination *ReplicationDestination
}
type ReplicationDestination struct {
Bucket string
StorageClass string
}
```

参数名称	描述	类型
PutBucketReplicationOptions	存储桶复制规则	struct
Role	发起者身份标示：`qcs::cam::uin/<owneruin>:uin/<subuin>`	string
Rule	具体配置信息，最多支持1000个，所有策略只能指向一个目标存储桶	struct
ID	用来标注具体 Rule 的名称	string
Status	标识 Rule 是否生效，枚举值：Enabled, Disabled	string
Prefix	前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空	string
Destination	目标存储桶信息	struct
Bucket	资源标识符：`qcs::cos:[region]::[bucketname-AppId]`	string
StorageClass	存储级别，枚举值：STANDARD，STANDARD_IA。更多类型请参见上方 API 概览。默认值：原存储类型	string

查询存储桶复制

功能说明

查询指定存储桶的存储桶复制规则（GET Bucket replication）。

方法原型

```
func (s *BucketService) GetBucketReplication(ctx context.Context) (*GetBucketReplicationResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-12500000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, _, err := client.Bucket.GetBucketReplication(context.Background())
if err != nil {
panic(err)
}
}
```

返回结果说明

```
type GetBucketReplicationResult struct {
Role string
Rule []BucketReplicationRule
}
```

版权所有：腾讯云计算（北京）有限责任公司

```
type BucketReplicationRule struct {
ID string
Status string
Prefix string
Destination *ReplicationDestination
}
type ReplicationDestination struct {
Bucket string
StorageClass string
}
```

参数名称	描述	类型
GetBucketReplicationResult	存储桶复制规则	struct
Role	发起者身份标示：`qcs::cam::uin/<owneruin>:uin/<subuin>`	string
Rule	具体配置信息，最多支持1000个，所有策略只能指向一个目标存储桶	struct
ID	用来标注具体 Rule 的名称	string
Status	标识 Rule 是否生效，枚举值：Enabled，isabled	string
Prefix	前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空	string
Destination	目标存储桶信息	struct
Bucket	资源标识符：`qcs::cos:[region]::[bucketname-AppId]`	string
StorageClass	存储级别，枚举值：STANDARD，STANDARD_IA。更多类型请参见上方 API 概览。默认值：原存储类型	string

删除存储桶复制

功能说明

删除指定存储桶的存储桶复制规则（DELETE Bucket replication）。

方法原型

```
func (s *BucketService) DeleteBucketReplication(ctx context.Context) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-12500000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, err := client.Bucket.DeleteBucketReplication(context.Background())
if err != nil {
panic(err)
}
}
```

数据管理
生命周期

最近更新时间：2023-04-12 14:28:24

简介

本文档提供关于生命周期的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket lifecycle	设置生命周期	设置存储桶的生命周期管理的配置
GET Bucket lifecycle	查询生命周期	查询存储桶生命周期管理的配置
DELETE Bucket lifecycle	删除生命周期	删除存储桶生命周期管理的配置

设置生命周期

功能说明

设置指定存储桶的生命周期配置信息（PUT Bucket lifecycle）。

方法原型

```
func (s *BucketService) PutLifecycle(ctx context.Context, opt *BucketPutLifecycleOptions) (*Response, error)
```

请求示例

```
package main

import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)

func main() {
```

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
lc := &cos.BucketPutLifecycleOptions{
Rules: []cos.BucketLifecycleRule{
{
ID: "1234",
Filter: &cos.BucketLifecycleFilter{Prefix: "test"},
Status: "Enabled",
Transition: []cos.BucketLifecycleTransition{
{
Days: 10,
StorageClass: "Standard",
},
},
},
{
ID: "123422",
Filter: &cos.BucketLifecycleFilter{Prefix: "gg"},
Status: "Disabled",
Expiration: &cos.BucketLifecycleExpiration{
Days: 10,
},
},
},
}
_, err := client.Bucket.PutLifecycle(context.Background(), lc)
```

```
if err != nil {
panic(err)
}
}
```

参数说明

```
type BucketPutLifecycleOptions struct {
XMLName xml.Name
Rules []BucketLifecycleRule
}
type BucketLifecycleRule struct {
ID string
Status string
Filter *BucketLifecycleFilter
Transition []BucketLifecycleTransition
Expiration *BucketLifecycleExpiration
AbortIncompleteMultipartUpload *BucketLifecycleAbortIncompleteMultipartUpload
NoncurrentVersionTransition []BucketLifecycleNoncurrentVersion
NoncurrentVersionExpiration *BucketLifecycleNoncurrentVersion
}
type BucketLifecycleFilter struct {
Prefix string
Tag *BucketTaggingTag
And *BucketLifecycleAndOperator
}
type BucketLifecycleTransition struct {
Date string
Days int
StorageClass string
}
type BucketLifecycleExpiration struct {
Date string
Days int
ExpiredObjectDeleteMarker bool
}
type BucketLifecycleNoncurrentVersion struct {
NoncurrentDays int
StorageClass string
}
type BucketLifecycleAbortIncompleteMultipartUpload struct {
DaysAfterInitiation int
}
```

参数名称	父节点	参数描述	类

参数名称	父节点	参数描述	类
LifecycleConfiguration	无	生命周期配置	Co
Rule	LifecycleConfiguration	规则描述	Co
ID	LifecycleConfiguration.Rule	用于唯一地标识规则，长度不能超过255个字符	St
Filter	LifecycleConfiguration.Rule	Filter 用于描述规则影响的 Object 集合	Co
And	LifecycleConfiguration.Rule .Filter	对象筛选器中的一个子集，仅当需要指定多种筛选规则时才需要此元素，例如：同时指定 Prefix 和 Tag 筛选，或同时指定多个 Tag 筛选。	Co
Prefix	LifecycleConfiguration.Rule .Filter.And	指定规则所适用的前缀。匹配前缀的对象受该规则影响，Prefix 最多只能有一个	St
Tag	LifecycleConfiguration.Rule .Filter.And	标签集合，最多支持10个标签	Co
Key	LifecycleConfiguration.Rule .Filter.And.Tag	标签的 Key，长度不超过128字节，支持英文字母、数字、空格、加号、 减号、下划线、等号、点号、冒号、斜线	St
Value	LifecycleConfiguration.Rule .Filter.And.Tag	标签的 Value，长度不超过256字节, 支持英文字母、数字、空格、加号、 减号、下划线、等号、点号、冒号、斜线	St
Status	LifecycleConfiguration.Rule	指明规则是否启用，枚举值：Enabled，Disabled	St
Expiration	LifecycleConfiguration.Rule	规则过期属性	Co
Transition	LifecycleConfiguration.Rule	规则转换属性，用于描述对象何时进行存储类型的转换和转换的存储类型	Co

版权所有：腾讯云计算（北京）有限责任公司

参数名称	父节点	参数描述	类
Days	LifecycleConfiguration.Rule .Transition 或 Expiration	指明规则对应的动作在对象最后的修改日期过后多少天操作：如果是 Transition，该字段有效值是非负整数 如果是 Expiration，该字段有效值为正整数，最大支持3650天	Int
Date	LifecycleConfiguration.Rule .Transition 或 Expiration	指明规则对应的动作在何时操作，支持 `2007-12-01T12:00:00.000Z` 和 `2007-12-01T00:00:00+08:00` 这两种格式	St
ExpiredObjectDeleteMarker	LifecycleConfiguration.Rule .Expiration	删除过期对象删除标记，枚举值 true，false	St
AbortIncompleteMultipartUpload	LifecycleConfiguration.Rule	设置允许分片上传保持运行的最长时间	Co
DaysAfterInitiation	LifecycleConfiguration.Rule .AbortIncompleteMultipartUpload	指明分片上传开始后多少天内必须完成上传	Int
NoncurrentVersionExpiration	LifecycleConfiguration.Rule	指明非当前版本对象何时过期	Co
NoncurrentVersionTransition	LifecycleConfiguration.Rule	指明非当前版本对象何时进行存储类型的转换和转换的存储类型	Co
NoncurrentDays	LifecycleConfiguration.Rule .NoncurrentVersionExpiration 或 NoncurrentVersionTransition	指明规则对应的动作在对象变成非当前版本多少天后执行 如果是 Transition，该字段有效值是非负整数 如果是 Expiration，该字段有效值为正整数，最大支持3650天	Int
StorageClass	LifecycleConfiguration.Rule .Transition 或 NoncurrentVersionTransition	指定 Object 沉降后的存储类型，枚举值：STANDARD_IA，MAZ_STANDARD_IA，INTELLIGENT_TIERING，MAZ_INTELLIGENT_TIERING，ARCHIVE，DEEP_ARCHIVE。关于存储类型的介绍，请参见 存储类型概述	St

版权所有：腾讯云计算（北京）有限责任公司

查询生命周期

功能说明

查询存储桶的生命周期管理配置（GET Bucket lifecycle）。

方法原型

```
func (s *BucketService) GetLifecycle(ctx context.Context) (*BucketGetLifecycleResult, *Response, error)
```

请求示例

```
package main

import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)

func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
```

```
})
_, _, err := client.Bucket.GetLifecycle(context.Background())
if err != nil {
panic(err)
}
}
```

返回结果说明

通过 GetBucketLifecycleResult 返回请求结果。

```
type BucketLifecycleRule struct {
ID string
Status string
Filter *BucketLifecycleFilter
Transition *BucketLifecycleTransition
Expiration *BucketLifecycleExpiration
AbortIncompleteMultipartUpload *BucketLifecycleAbortIncompleteMultipartUpload
}
type BucketLifecycleFilter struct {
Prefix string
}
type BucketLifecycleTransition struct {
Date string
Days int
StorageClass string
}
type BucketLifecycleExpiration struct {
Date string
Days int
}
type BucketLifecycleAbortIncompleteMultipartUpload struct {
DaysAfterInitiation string
}
```

参数名称	参数描述	类型	是否必填
BucketLifecycleRule	设置对应的规则，包括 ID，Filter，Status，Expiration，Transition，AbortIncompleteMultipartUpload	List	是
ID	规则的唯一 ID	String	否
Status	设置 Rule 是否启用，可选值为 Enabled 或者 Disabled	String	是

参数名称	参数描述	类型	是否必填
Filter	用于描述规则影响的 Object 集合，如需设置 Bucket 中的所有 objects，请设置 Prefix 为空	Struct	是
Transition	设置 Object 转换存储类型规则，可以指定天数 Days 或者指定日期 Date，Date 的格式必须是 GMT ISO 8601。StorageClass 可选 Standard_IA，Archive，可以同时设置多条此类规则	struct	否
Expiration	设置 Object 过期规则，可以指定天数 Days 或者指定日期 Date，Date 的格式必须是 GMT ISO 8601	Struct	否
AbortIncompleteMultipartUpload	指明分块上传开始后多少天内必须完成上传	Struct	否

删除生命周期

功能说明

删除存储桶生命周期管理的配置（DELETE Bucket lifecycle）。

方法原型

```
func (s *BucketService) DeleteLifecycle(ctx context.Context) (*Response, error)
```

请求示例

```
package main

import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)

func main() {
// 存储桶名称，由bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
```

```
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, err := client.Bucket.DeleteLifecycle(context.Background())
if err != nil {
panic(err)
}
}
```

日志管理

最近更新时间：2023-02-08 14:40:10

简介

本文档提供关于日志管理的 API 概览以及 Go SDK 示例代码。

API	操作名	操作描述
PUT Bucket logging	设置日志管理	为源存储桶开启日志记录
GET Bucket logging	查询日志管理	查询源存储桶的日志配置信息

设置日志管理

功能说明

PUT Bucket logging 用于为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中。

方法原型

```
func (s *BucketService) PutLogging(ctx context.Context, opt *BucketPutLoggingOpti
ons) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
t/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
```

版权所有：腾讯云计算（北京）有限责任公司

```
m")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
opt := &cos.BucketPutLoggingOptions{
LoggingEnabled: &cos.BucketLoggingEnabled{
TargetBucket: "TargetBucket-1250000000",
},
}
_, err := client.Bucket.PutLogging(context.Background(), opt)
if err != nil {
// ERROR
}
}
```

参数说明

```
type BucketLoggingEnabled struct {
TargetBucket string
TargetPrefix string
}
// BucketPutLoggingOptions is the options of PutBucketLogging
type BucketPutLoggingOptions struct {
XMLName xml.Name
LoggingEnabled *BucketLoggingEnabled
}
```

参数名称	描述	类型
BucketPutLoggingOptions	日志管理配置参数	Struct
LoggingEnabled	日志管理配置	Struct

参数名称	描述	类型
TargetBucket	存放日志的目标存储桶，可以是同一个存储桶（但不推荐），或同一账户下、同一地域的存储桶	String
TargetPrefix	日志存放在目标存储桶的指定路径	String

查询日志管理

功能说明

GET Bucket logging 用于查询指定存储桶的日志配置信息。

方法原型

```
func (s *BucketService) GetLogging(ctx context.Context) (*BucketGetLoggingResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权
```

```
限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
v, _, err := client.Bucket.GetLogging(context.Background())
if err != nil {
// ERROR
}
fmt.Println(v)
}
```

返回结果说明

```
type BucketGetLoggingResult BucketPutLoggingOptions
```

参数名称	描述	类型
BucketGetLoggingResult	日志管理配置参数	Struct
LoggingEnabled	日志管理配置	Struct
TargetBucket	存放日志的目标存储桶，可以是同一个存储桶（但不推荐），或同一账户下、同一地域的存储桶	String
TargetPrefix	日志存放在目标存储桶的指定路径	String

存储桶标签

最近更新时间：2023-02-08 14:40:10

简介

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket tagging	设置存储桶标签	为已存在的存储桶设置标签
GET Bucket tagging	查询存储桶标签	查询指定存储桶下已有的存储桶标签
DELETE Bucket tagging	删除存储桶标签	删除指定的存储桶标签

设置存储桶标签

功能说明

PUT Bucket tagging 用于为已存在的存储桶设置标签。

方法原型

```
func (s *BucketService) PutTagging(ctx context.Context, opt *BucketPutTaggingOpti
ons) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
```

```
t/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
opt := &cos.BucketPutTaggingOptions{
TagSet: []cos.BucketTaggingTag{
{
Key: "testk1",
Value: "testv1",
},
{
Key: "testk2",
Value: "testv2",
},
},
}
_, err := client.Bucket.PutTagging(context.Background(), opt)
if err != nil {
// ERROR
}
}
```

参数说明

```
type BucketTaggingTag struct {
Key string
Value string
}
type BucketPutTaggingOptions struct {
XMLName xml.Name
```

```
TagSet []BucketTaggingTag
}
```

参数名称	描述	类型
BucketPutTaggingOptions	存储桶标签配置参数	Struct
TagSet	存储桶标签配置信息	Struct
key	标签的 Key，长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String
value	标签的 Value，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String

查询存储桶标签

功能说明

GET Bucket tagging 用于查询指定存储桶下已有的存储桶标签。

方法原型

```
func (s *BucketService) GetTagging(ctx context.Context) (*BucketGetTaggingResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶region可以在COS控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
```

```
m")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
v, _, err := client.Bucket.GetTagging(context.Background())
if err != nil {
fmt.Println(err)
}
fmt.Println(v)
}
```

返回结果说明

```
type BucketTaggingTag struct {
Key string
Value string
}
type BucketGetTaggingResult struct {
XMLName xml.Name
TagSet []BucketTaggingTag
}
```

参数名称	描述	类型
BucketGetTaggingResult	存储桶标签配置参数	Struct
TagSet	存储桶标签配置信息	Struct
key	标签的 Key，长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String
value	标签的 Value，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String

版权所有：腾讯云计算（北京）有限责任公司

删除存储桶标签

功能说明

DELETE Bucket tagging 用于删除指定存储桶下已有的存储桶标签。

方法原型

```
func (s *BucketService) DeleteTagging(ctx context.Context) (*Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶region可以在COS控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
resp, err := client.Bucket.DeleteTagging(context.Background())
```

```
if err != nil {
fmt.Println(err)
}
fmt.Println(resp.Header)
}
```

版权所有：腾讯云计算（北京）有限责任公司

对象标签

最近更新时间：2023-02-08 14:40:10

简介

本文档提供关于对象标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Object tagging	设置对象标签	为已上传的对象设置标签
GET Object tagging	查询对象标签	查询指定对象下已有的对象标签
DELETE Object tagging	删除对象标签	删除指定对象下已有的对象标签

设置对象标签

功能说明

COS 支持为已存在的对象设置标签。PUT Object tagging 接口通过为对象添加键值对作为对象标签，可以协助您分组管理已有的对象资源，详情请参见 对象标签概述。

方法原型

```
func (s *ObjectService) PutTagging(ctx context.Context, name string, opt *ObjectPutTaggingOptions, id ...string) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
"strings"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
```

```
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
// Case1 通过 PutTagging 设置云上对象标签
opt := &cos.ObjectPutTaggingOptions{
TagSet: []cos.ObjectTaggingTag{
{
Key: "test_k2",
Value: "test_v2",
},
{
Key: "test_k3",
Value: "test_v3",
},
},
}
name := "example"
_, err := client.Object.PutTagging(context.Background(), name, opt)
if err != nil {
//ERROR
}
// Case2 上传时设置对象标签
name = "test/example"
f := strings.NewReader("test")
popt := &cos.ObjectPutOptions{
ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
XOptionHeader: &http.Header{},
},
}
```

```
popt.XOptionHeader.Add("x-cos-tagging", "Key1=Value1&Key2=Value2")
_, err = client.Object.Put(context.Background(), name, f, popt)
}
```

参数说明

```
type ObjectPutTaggingOptions struct {
TagSet []ObjectTaggingTag
}
type BucketTaggingTag struct {
Key string
Value string
}
```

参数名称	参数描述	类型	是否必填
name	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
TagSet	标签集合，最多支持10个标签	Array	是
Key	标签键，长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String	是
Value	标签值，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String	是

查询对象标签

功能说明

GET Object tagging 接口用于查询指定对象下已有的对象标签。

方法原型

```
func (s *ObjectService) GetTagging(ctx context.Context, name string, id ...string
) (*ObjectGetTaggingResult, *Response, error)
```

请求示例

```
package main
import (
```

```
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
name := "example"
res, _, err := client.Object.GetTagging(context.Background(), name)
if err != nil {
//ERROR
}
fmt.Println(res)
}
```

结果说明

```
type ObjectGetTaggingResult struct {
TagSet []ObjectTaggingTag
}
type BucketTaggingTag struct {
Key string
```

```
Value string
}
```

参数名称	参数描述	类型
TagSet	标签集合，最多支持10个标签	Array
Key	标签键，长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String
Value	标签值，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String

删除对象标签

功能说明

DELETE Object tagging 接口用于删除指定对象下已有的对象标签。

方法原型

```
func (s *ObjectService) DeleteTagging(ctx context.Context, name string, id ...str
ing) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
t/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
m")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
```

```
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
name := "example"
_, err := client.Object.DeleteTagging(context.Background(), name)
if err != nil {
//ERROR
}
}
```

静态网站

最近更新时间：2023-02-08 14:40:10

简介

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket website	设置静态网站	设置存储桶的静态网站配置
GET Bucket website	查询静态网站配置	查询存储桶的静态网站配置
DELETE Bucket website	删除静态网站配置	删除存储桶的静态网站配置

设置静态网站

功能说明

PUT Bucket website 用于为存储桶配置静态网站。

方法原型

```
func (s *BucketService) PutWebsite(ctx context.Context, opt *BucketPutWebsiteOpti
ons) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
```

版权所有：腾讯云计算（北京）有限责任公司

```
t/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
opt := &cos.BucketPutWebsiteOptions{
Index: "index.html",
Error: &cos.ErrorDocument{"index_backup.html"},
RoutingRules: &cos.WebsiteRoutingRules{
[]cos.WebsiteRoutingRule{
{
ConditionErrorCode: "404",
RedirectProtocol: "https",
RedirectReplaceKey: "404.html",
},
{
ConditionPrefix: "docs/",
RedirectProtocol: "https",
RedirectReplaceKeyPrefix: "documents/",
},
},
},
}
_, err := client.Bucket.PutWebsite(context.Background(), opt)
if err != nil {
// ERROR
}
}
```

参数说明

```
type WebsiteRoutingRule struct {
ConditionErrorCode string
ConditionPrefix string
RedirectProtocol string
RedirectReplaceKey string
RedirectReplaceKeyPrefix string
}
type WebsiteRoutingRules struct {
Rules []WebsiteRoutingRule
}
type ErrorDocument struct {
Key string
}
type RedirectRequestsProtocol struct {
Protocol string
}
type BucketPutWebsiteOptions struct {
XMLName xml.Name
Index string
RedirectProtocol *RedirectRequestsProtocol
Error *ErrorDocument
RoutingRules *WebsiteRoutingRules
}
```

参数名称	描述	类型
BucketPutWebsiteOptions	静态网站配置参数	Struct
Index	指定索引文档	String
RedirectProtocol	全站重定向的协议	Struct
Protocol	指定全站重定向的协议，只能设置为 https	String
Error	错误文档	Struct
Key	指定通用错误返回	String
RoutingRules	设置重定向规则，最多设置100条 RoutingRule	Struct
ConditionErrorCode	指定重定向错误码，只支持配置4XX返回码，优先级高于 Error.Key	String
ConditionPrefix	指定前缀重定向的路径，替换指定的 folder/	String
RedirectProtocol	指定重定向规定的协议，只能设置为 https	String
RedirectReplaceKey	替换整个 Key 为指定的内容	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
RedirectReplaceKeyPrefix	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String

查询静态网站配置

功能说明

GET Bucket website 用于查询与存储桶关联的静态网站配置信息。

方法原型

```
func (s *BucketService) GetWebsite(ctx context.Context) (*BucketGetWebsiteResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
```

```
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
res, _, err := client.Bucket.GetWebsite(context.Background())
if err != nil {
// ERROR
}
fmt.Println(res)
}
```

返回结果说明

```
type BucketGetWebsiteResult BucketPutWebsiteOptions
```

参数名称	描述	类型
BucketGetWebsiteResult	静态网站配置参数	Struct
Index	指定索引文档	String
RedirectProtocol	全站重定向的协议	Struct
Protocol	指定全站重定向的协议，只能设置为 https	String
Error	通用错误返回	Struct
Key	指定通用错误返回	String
RoutingRules	设置重定向规则，最多设置100条 RoutingRule	Struct
ConditionErrorCode	指定重定向错误码，只支持配置4XX返回码，优先级高于 Error.Key	String
ConditionPrefix	指定前缀重定向的路径，替换指定的 folder/	String
RedirectProtocol	指定重定向规定的协议，只能设置为 https	String
RedirectReplaceKey	替换整个 Key 为指定的内容	String
RedirectReplaceKeyPrefix	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String

版权所有：腾讯云计算（北京）有限责任公司

删除静态网站配置

功能说明

DELETE Bucket website 用于删除存储桶中的静态网站配置。

方法原型

```
func (s *BucketService) DeleteWebsite(ctx context.Context) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, err := client.Bucket.DeleteWebsite(context.Background())
if err != nil {
```

```
// ERROR
}
}
```

清单

最近更新时间：2023-02-08 14:40:10

简介

本文档提供关于清单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket inventory	设置清单任务	设置存储桶的清单任务
GET Bucket inventory	查询清单任务	查询存储桶的清单任务
List Bucket Inventory Configurations	查询所有清单	查询存储桶的所有清单任务
DELETE Bucket inventory	删除清单任务	删除存储桶的清单任务

设置清单任务

功能说明

PUT Bucket inventory 用于在存储桶中创建清单任务。

方法原型

```
func (s *BucketService) PutInventory(ctx context.Context, id string, opt *BucketPutInventoryOptions) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
```

```
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://intl.cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
},
})
opt := &cos.BucketPutInventoryOptions{
ID: "test_id",
// True or False
IsEnabled: "True",
IncludedObjectVersions: "All",
Filter: &cos.BucketInventoryFilter{
Prefix: "test",
},
OptionalFields: &cos.BucketInventoryOptionalFields{
BucketInventoryFields: []string{
"Size", "LastModifiedDate",
},
},
Schedule: &cos.BucketInventorySchedule{
// Weekly or Daily
Frequency: "Daily",
},
Destination: &cos.BucketInventoryDestination{
Bucket: "dest_bucket-1250000000",
Format: "CSV",
},
}
_, err := client.Bucket.PutInventory(context.Background(), "test_id", opt)
if err != nil {
// ERROR
```

```
}
}
```

参数说明

```
type BucketInventoryFilter struct {
Prefix string
}
type BucketInventoryOptionalFields struct {
BucketInventoryFields []string
}
type BucketInventorySchedule struct {
Frequency string
}
type BucketInventoryEncryption struct {
SSECOS string
}
type BucketInventoryDestination struct {
Bucket string
AccountId string
Prefix string
Format string
Encryption *BucketInventoryEncryption
}
// BucketPutInventoryOptions ...
type BucketPutInventoryOptions struct {
XMLName xml.Name
ID string
IsEnabled string
IncludedObjectVersions string
Filter *BucketInventoryFilter
OptionalFields *BucketInventoryOptionalFields
Schedule *BucketInventorySchedule
Destination *BucketInventoryDestination
}
```

参数名称	描述	类型
BucketPutInventoryOptions	存储桶清单配置信息	Struct
ID	清单的名称，与请求参数中的 ID 对应	String
IsEnabled	清单是否启用的标识： • 如果设置为 true，清单功能将生效 • 如果设置为 false，将不生成任何清单	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
IncludedObjectVersions	是否在清单中包含对象版本： • 如果设置为 All，清单中将会包含所有对象版本，并在清单中增加 VersionId，IsLatest，DeleteMarker 这几个字段 • 如果设置为 Current，则清单中不包含对象版本信息	String
Filter	过滤器	Struct
Prefix	需要分析的对象的前缀	String
OptionalFields	设置清单结果中应包含的分析项目	Struct
BucketInventoryFields	清单结果中可选包含的分析项目名称，可选字段包括：Size，LastModifiedDate，StorageClass，ETag，IsMultipartUploaded，ReplicationStatus	[]String
Schedule	清单任务周期	Struct
Frequency	清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly	String
Destination	描述存放清单结果的信息	Struct
Bucket	清单分析结果的存储桶名	String
AccountId	存储桶的所有者 ID，例如100000000001	String
Prefix	清单分析结果的前缀	String
Format	清单分析结果的文件形式，可选项为 CSV 格式	String
Encryption	为清单结果提供服务端加密的选项	Struct
SSECOS	使用 SSE-COS 加密	String

错误码说明

该请求可能会发生的一些常见的特殊错误如下：

错误码	描述	状态码
InvalidArgument	不合法的参数值	HTTP 400 Bad Request
TooManyConfigurations	清单数量已经达到1000条的上限	HTTP 400 Bad Request
AccessDenied	未授权的访问。您可能不具备访问该存储桶的权限	HTTP 403 Forbidden

版权所有：腾讯云计算（北京）有限责任公司

查询清单任务

功能说明

GET Bucket inventory 用于查询存储桶中用户的清单任务信息。

方法原型

```
func (s *BucketService) GetInventory(ctx context.Context, id string) (*BucketGetInventoryResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://intl.cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
},
})
```

```
v, _, err := client.Bucket.GetInventory(context.Background(), "test_id")
if err != nil {
// ERROR
}
fmt.Println(v)
}
```

返回结果说明

```
type BucketGetInventoryResult BucketPutInventoryOptions
```

参数名称	描述	类型
BucketPutInventoryOptions	存储桶清单配置信息	Struct
ID	清单的名称，与请求参数中的 ID 对应	String
IsEnabled	清单是否启用的标识： • 如果设置为 true，清单功能将生效 • 如果设置为 false，将不生成任何清单	String
IncludedObjectVersions	是否在清单中包含对象版本： • 如果设置为 All，清单中将会包含所有对象版本，并在清单中增加 VersionId，IsLatest，DeleteMarker 这几个字段 • 如果设置为 Current，则清单中不包含对象版本信息	String
Filter	过滤器	Struct
Prefix	需要分析的对象的前缀	String
OptionalFields	设置清单结果中应包含的分析项目	Struct
BucketInventoryFields	清单结果中可选包含的分析项目名称，可选字段包括：Size，LastModifiedDate，StorageClass，ETag，IsMultipartUploaded，ReplicationStatus	[]String
Schedule	清单任务周期	Struct
Frequency	清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly	String
Destination	描述存放清单结果的信息	Struct
Bucket	清单分析结果的存储桶名	String
AccountId	存储桶的所有者 ID，例如100000000001	String
Prefix	清单分析结果的前缀	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
Format	清单分析结果的文件形式，可选项为 CSV 格式	String
Encryption	为清单结果提供服务端加密的选项	Struct
SSECOS	使用 SSE-COS 加密	String

查询所有清单

功能说明

List Bucket Inventory Configurations 用于请求返回一个存储桶中的所有清单任务。每一个存储桶中最多配置1000条清单任务。

方法原型

```
func (s *BucketService) ListInventoryConfigurations(ctx context.Context, token string) (*ListBucketInventoryConfigResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶region可以在COS控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://intl.cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.int
```

```
l.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
},
})
v, _, err := client.Bucket.ListInventoryConfigurations(context.Background(), "")
if err != nil {
// ERROR
}
fmt.Println(v)
}
```

返回结果说明

```
type BucketListInventoryConfiguartion BucketPutInventoryOptions
type ListBucketInventoryConfigResult struct {
XMLName xml.Name
InventoryConfigurations []BucketListInventoryConfiguartion
IsTruncated bool
ContinuationToken string
NextContinuationToken string
}
```

参数名称	描述	类型
ListBucketInventoryConfigResult	存储桶所有清单配置信息	Struct
InventoryConfigurations	清单配置信息	Struct
IsTruncated	是否已列出所有清单任务信息的标识。如果已经展示完则为 false，否则为 true	Bool
ContinuationToken	当页清单列表的标识，可理解为页数。该标识与请求中的 continuation-token 参数对应	String
NextContinuationToken	下一页清单列表的标识。如果该参数中有值，则可将该值作为 continuation-token 参数并发起 GET 请求以获取下一页清单任务信息	String

删除清单任务

功能说明

DELETE Bucket inventory 用于删除存储桶中指定的清单任务。

方法原型

```
func (s *BucketService) DeleteInventory(ctx context.Context, id string) (*Respons
e, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
intl.cloud.tencent.com/ ，关于地域的详情见 https://intl.cloud.tencent.com/document/p
roduct/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
m")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.int
l.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权
限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/produ
ct/598/32675
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.i
ntl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最
小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/p
roduct/598/32675
},
})
_, err := client.Bucket.DeleteInventory(context.Background(), "test_id")
```

```
if err != nil {
// ERROR
}
}
```

参数说明

参数名称	描述	类型
id	清单的名称	String

访问管理
跨域访问

最近更新时间：2023-02-08 14:36:23

简介

本文档提供关于跨域访问的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域名访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域名访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域名访问配置信息

设置跨域配置

功能说明

设置指定存储桶的跨域名访问配置信息（PUT Bucket cors）。

方法原型

```
func (s *BucketService) PutCORS(ctx context.Context, opt *BucketPutCORSOptions)
(*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
```

```
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
opt := &cos.BucketPutCORSOptions{
Rules: []cos.BucketCORSRule{
{
AllowedOrigins: []string{"http://www.qq.com"},
AllowedMethods: []string{"PUT", "GET"},
AllowedHeaders: []string{"x-cos-meta-test", "x-cos-xx"},
MaxAgeSeconds: 500,
ExposeHeaders: []string{"x-cos-meta-test1"},
},
{
ID: "1234",
AllowedOrigins: []string{"http://www.baidu.com", "twitter.com"},
AllowedMethods: []string{"PUT", "GET"},
MaxAgeSeconds: 500,
},
},
}
_, err := client.Bucket.PutCORS(context.Background(), opt)
if err != nil {
panic(err)
}
}
```

参数说明

```
type BucketCORSRule struct {
ID string
AllowedMethods []string
AllowedOrigins []string
AllowedHeaders []string
MaxAgeSeconds int
ExposeHeaders []string
}
```

参数名称	参数描述	类型	是否必填
BucketCORSRule	设置对应的跨域规则，包括 ID，MaxAgeSeconds，AllowedOrigin，AllowedMethod，AllowedHeader，ExposeHeader	struct	是
ID	设置规则的 ID	string	否
AllowedMethods	设置允许的方法，如 GET，PUT，HEAD，POST，DELETE	[]string	是
AllowedOrigins	设置允许的访问来源，如 `"http://cloud.tencent.com"`，支持通配符 *	[]string	是
AllowedHeaders	设置请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *	[]string	否
MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	int	否
ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	[]string	否

查询跨域配置

功能说明

查询指定存储桶的跨域名访问配置信息（GET Bucket cors）。

方法原型

```
func (s *BucketService) GetCORS(ctx context.Context) (*BucketGetCORSResult, *Response, error)
```

请求示例

```
package main
import (
"context"
```

```
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, _, err := client.Bucket.GetCORS(context.Background())
if err != nil {
panic(err)
}
}
```

返回结果说明

通过 GetBucketCORSResult 返回请求结果。

```
type BucketCORSRule struct {
ID string
AllowedMethods []string
AllowedOrigins []string
AllowedHeaders []string
MaxAgeSeconds int
```

版权所有：腾讯云计算（北京）有限责任公司

```
ExposeHeaders []string
}
```

参数名称	参数描述	类型	是否必填
BucketCORSRule	设置对应的跨域规则，包括 ID，MaxAgeSeconds，AllowedOrigin，AllowedMethod，AllowedHeader，ExposeHeader	struct	是
ID	设置规则的 ID	string	否
AllowedMethods	设置允许的方法，如 GET，PUT，HEAD，POST，DELETE	[]string	是
AllowedOrigins	设置允许的访问来源，如 `"http://cloud.tencent.com"`，支持通配符 *	[]string	是
AllowedHeaders	设置请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *	[]string	否
MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	int	否
ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	[]string	否

删除跨域配置

功能说明

删除指定存储桶的跨域名访问配置（DELETE Bucket cors）。

方法原型

```
func (s *BucketService) DeleteCORS(ctx context.Context) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
```

```
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, err := client.Bucket.DeleteCORS(context.Background())
if err != nil {
panic(err)
}
}
```

存储桶策略

最近更新时间：2023-02-08 14:36:23

简介

本文档提供关于存储桶策略的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket policy	设置存储桶策略	设置指定存储桶的权限策略
GET Bucket policy	查询存储桶策略	查询指定存储桶的权限策略
DELETE Bucket policy	删除存储桶策略	删除指定存储桶的权限策略

设置存储桶策略

功能说明

PUT Bucket policy 请求可以向 Bucket 写入权限策略，当存储桶已存在权限策略时，该请求上传的策略将覆盖原有的权限策略。

方法原型

```
func (s *BucketService) PutPolicy(ctx context.Context, opt *BucketPutPolicyOption
s) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
```

```
intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
opt := &cos.BucketPutPolicyOptions{
Version: "2.0",
Statement: []cos.BucketStatement{
{
Principal: map[string][]string{
"qcs": []string{
"qcs::cam::uin/100000000001:uin/100000000011", //替换成您想授予权限的账户uin
},
},
Action: []string{
"name/cos:GetObject",
},
Effect: "allow",
Resource: []string{
//这里改成允许的路径前缀，可以根据自己网站的用户登录态判断允许上传的具体路径，例子： a.jpg 或者 a/* 或者 * (使用通配符*存在重大安全风险，请谨慎评估使用)
"qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/exampleobject",
},
Condition: map[string]map[string]interface{}{
"ip_not_equal": map[string]interface{}{
"qcs:ip": []string{
"192.168.1.1",
},
},
},
},
},
```

```
}
_, err := client.Bucket.PutPolicy(context.Background(), opt)
if err != nil {
// ERROR
}
}
```

参数说明

```
type BucketStatement struct {
Principal map[string][]string
Action []string
Effect string
Resource []string
Condition map[string]map[string]interface{}
}
type BucketPutPolicyOptions struct {
Statement []BucketStatement
Version string
Principal map[string][]string
}
```

参数名称	描述	类型
Statement	描述一条或多条权限的详细信息	Struct
Version	策略语法版本	Struct
Principal	描述策略授权的实体，详情请参见 访问策略语言概述	String
Action	此处是指 COS API，根据需求指定一个或者一序列操作的组合或所有操作(*)，例如 action 为 name/cos:GetService，**请注意区分英文大小写**	Array
Effect	有 allow（允许）和 deny（显式拒绝）两种情况	String
Resource	授权操作的具体数据，可以是任意资源、指定路径前缀的资源、指定绝对路径的资源或它们的组合	Array
Condition	约束条件，可以不填，具体说明请参见 condition 说明	Struct

查询存储桶策略

功能说明

GET Bucket policy 请求可以向 Bucket 读取权限策略。

方法原型

```
func (s *BucketService) GetPolicy(ctx context.Context) (*BucketGetPolicyResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
res, _, err := client.Bucket.GetPolicy(context.Background())
if err != nil {
// ERROR
}
fmt.Println(res)
}
```

返回结果说明

```
type BucketGetPolicyResult BucketPutPolicyOptions
// 详情见 BucketPutPolicyOptions
```

删除存储桶策略

功能说明

DELETE Bucket policy 请求可以向 Bucket 删除权限策略。

方法原型

```
func (s *BucketService) DeletePolicy(ctx context.Context) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
```

```
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, err := client.Bucket.DeletePolicy(context.Background())
if err != nil {
// ERROR
}
}
```

自定义域名

最近更新时间：2023-02-08 14:36:23

简介

本文档提供关于自定义域名的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket domain	设置自定义域名	设置存储桶的自定义域名信息
GET Bucket domain	查询自定义域名	查询存储桶的自定义域名信息
DELETE Bucket domain	删除自定义域名	删除存储桶的自定义域名信息

设置自定义域名

功能说明

PUT Bucket domain 用于为存储桶配置自定义域名。

方法原型

```
func (s *BucketService) PutDomain(ctx context.Context, opt *BucketPutDomainOptions) (*Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
```

```
intl.cloud.tencent.com/ , 关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
opt := &cos.BucketPutDomainOptions{
Rules: []cos.BucketDomainRule{
{
Status: "ENABLED",
Name: "www.example.com",
Type: "REST",
ForcedReplacement: "CNAME",
},
},
}
_, err := client.Bucket.PutDomain(context.Background(), opt)
if err != nil {
fmt.Println(err)
}
}
```

参数说明

```
type BucketDomainRule struct {
Status string
Name string
Type string
ForcedReplacement string
}
type BucketPutDomainOptions struct {
XMLName xml.Name
```

```
Rules []BucketDomainRule
}
```

参数名称	描述	类型
BucketPutDomainOptions	自定义域名配置	Struct
Rules	域名配置规则	Array
Status	域名上线/下线状态，有效值 ENABLED/DISABLED	String
Name	用户的自定义域名，有效值：字母、数字、点	String
Type	绑定的源站类型，有效值 REST/WEBSITE	String
ForcedReplacement	替换已存在的配置，有效值 CNAME/TXT。填写则强制校验域名所有权后，再下发配置	String

返回错误码说明

该请求可能会发生的一些常见的特殊错误如下：

状态码	说明
HTTP 409 Conflict	该域名记录已存在，且请求中没有设置强制覆盖。或者该域名记录不存在，且请求中设置了强制覆盖
HTTP 451 Unavailable For Legal Reasons	该域名是中国境内域名，并且没有备案

查询自定义域名

功能说明

GET Bucket domain 用于查询存储桶的自定义域名信息。

方法原型

```
func (s *BucketService) GetDomain(ctx context.Context) (*BucketGetDomainResult, *Response, error)
```

请求示例

```
package main
import (
```

```
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
v, _, err := client.Bucket.GetDomain(context.Background())
if err != nil {
fmt.Println(err)
}
fmt.Println(v)
}
```

返回结果说明

```
type BucketGetDomainResult BucketPutDomainOptions
```

参数名称	描述	类型
BucketGetDomainResult	自定义域名配置	Struct

参数名称	描述	类型
Rules	域名配置规则	Array
Status	域名上线/下线状态，有效值 ENABLED/DISABLED	String
Name	用户的自定义域名，有效值：字母、数字、点	String
Type	绑定的源站类型，有效值 REST/WEBSITE	String
ForcedReplacement	替换已存在的配置，有效值 CNAME/TXT。填写则强制校验域名所有权后，再下发配置	String

删除自定义域名

DELETE Bucket domain 用于删除存储桶所有的自定义域名信息。

方法原型

```
func (s *BucketService) DeleteDomain(ctx context.Context) (*Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
```

```
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, err := client.Bucket.DeleteDomain(context.Background())
if err != nil {
fmt.Println(err)
}
}
```

访问控制

最近更新时间：2023-02-08 14:36:23

简介

本文档提供关于存储桶、对象的访问控制列表（ACL）的 API 概览以及 SDK 示例代码。

存储桶 ACL

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶的访问权限控制列表（ACL）
GET Bucket acl	查询存储桶 ACL	查询指定存储桶的访问权限控制列表（ACL）

对象 ACL

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置 Bucket 中某个 Object （文件/对象）的 ACL
GET Object acl	查询对象 ACL	查询 Object（文件/对象）的 ACL

存储桶 ACL

设置存储桶 ACL

功能说明

设置指定存储桶访问权限控制列表（PUT Bucket acl）。

方法原型

```
func (s *BucketService) PutACL(ctx context.Context, opt *BucketPutACLOptions) (*Response, error)
```

请求示例

```
package main
import (
"context"
```

```
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
// 1. 通过请求头设置 Bucket ACL
opt := &cos.BucketPutACLOptions{
Header: &cos.ACLHeaderOptions{
//private, public-read, public-read-write
XCosACL: "private",
},
}
_, err := client.Bucket.PutACL(context.Background(), opt)
if err != nil {
panic(err)
}
// 2. 通过请求体设置 Bucket ACL
opt = &cos.BucketPutACLOptions{
Body: &cos.ACLXml{
Owner: &cos.Owner{
ID: "qcs::cam::uin/100000000001:uin/100000000001",
},
AccessControlList: []cos.ACLGrant{
```

```
{
Grantee: &cos.ACLGrantee{
// Type 备选项 "Group"、"CanonicalUser"
Type: "RootAccount",
ID: "qcs::cam::uin/100000760461:uin/100000760461",
},
// Permission 备选项 "WRITE"、"FULL_CONTROL"
Permission: "FULL_CONTROL",
},
},
},
}
_, err = client.Bucket.PutACL(context.Background(), opt)
if err != nil {
panic(err)
}
}
```

参数说明

```
type ACLHeaderOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
}
```

参数名称	参数描述
XCosACL	设置 Bucket 的 ACL，如 private，public-read，public-read-write
XCosGrantFullControl	赋予指定账户对 Bucket 的读写权限。格式为 `id=" ",id=" "` 。当需要给子账户授权 `id="qcs::cam::uin/{OwnerUin}:uin/{SubUin}"` ，当需要给主账户授权时，为 `id="qcs::cam::uin/{OwnerUin}:uin/{OwnerUin}"` 。例如 `id="qcs::cam::uin/100000000001:uin/100000000011",id="qcs::cam`
XCosGrantRead	赋予指定账户对 Bucket 的读权限。格式为 `id=" ",id=" "` 。当需要给子账户授权时 `id="qcs::cam::uin/{OwnerUin}:uin/{SubUin}"` ，当需要给主账户授权时，为 `id="qcs::cam::uin/{OwnerUin}:uin/{OwnerUin}"` 。例如 `id="qcs::cam::uin/100000000001:uin/100000000011",id="qcs::cam`
XCosGrantWrite	赋予指定账户对 Bucket 的写权限。格式为 `id=" ",id=" "` 。当需要给子账户授权时 `id="qcs::cam::uin/{OwnerUin}:uin/{SubUin}"` ，当需要给主账户授权时，为 `id="qcs::cam::uin/{OwnerUin}:uin/{OwnerUin}"` 。例如 `id="qcs::cam::uin/100000000001:uin/100000000011",id="qcs::cam`

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述
ACLXML	赋予指定账户对 Bucket 的访问权限，具体格式见 GET Bucket acl 返回结果说明

查询存储桶 ACL

功能说明

查询指定存储桶的访问权限控制列表（GET Bucket acl）。

方法原型

```
func (s *BucketService) GetACL(ctx context.Context) (*BucketGetACLResult, *Respon
se, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
t/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
m")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.int
l.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权
限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/59
8/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.i
ntl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最
```

```
小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
_, _, err := client.Bucket.GetACL(context.Background())
if err != nil {
panic(err)
}
}
```

返回结果说明

通过 GetBucketACLResult 返回请求结果。

```
type ACLXml struct {
Owner *Owner
AccessControlList []ACLGrant
}
type Owner struct {
ID string
DisplayName string
}
type ACLGrant struct {
Grantee *ACLGrantee
Permission string
}
type ACLGrantee struct {
Type string
ID string
DisplayName string
URI string
}
```

参数名称	参数描述	类型
Owner	Bucket 拥有者的信息，包括 DisplayName 和 ID	struct
AccessControlList	Bucket 权限授予者的信息，包括 Grantee和 Permission	struct
Grantee	权限授予者的信息，包括 DisplayName，Type，ID 和 URI	struct
Type	权限授予者的类型，类型为 CanonicalUser 或者 Group	string

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
ID	Type 为 CanonicalUser 时，对应权限授予者的 ID，格式为 qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]，如 qcs::cam::uin/100000000001:uin/100000000001，当 Type 指定为 CanonicalUser 时 ID 必选	string
DisplayName	权限被授予者的名称，可以不填，或者和ID的值保持一致	string
URI	Type 为 Group 时，填入预设用户组的 URI，例如 http://cam.qcloud.com/groups/global/AllUsers， 详情请参见 ACL 概述	string
Permission	授予者所拥有的 Bucket 的权限，可选值有 FULL_CONTROL，WRITE，READ，分别对应读写权限、写权限、读权限	string

对象 ACL

设置对象 ACL

功能说明

设置对象访问权限控制列表（PUT Object acl）。

方法原型

```
func (s *ObjectService) PutACL(ctx context.Context, key string, opt *ObjectPutACLOptions) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
```

```
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
// 1.通过请求头设置
opt := &cos.ObjectPutACLOptions{
Header: &cos.ACLHeaderOptions{
XCosACL: "private",
},
}
key := "exampleobject"
_, err := client.Object.PutACL(context.Background(), key, opt)
if err != nil {
panic(err)
}
// 2.通过请求体设置
opt = &cos.ObjectPutACLOptions{
Body: &cos.ACLXml{
Owner: &cos.Owner{
ID: "qcs::cam::uin/100000000001:uin/100000000001",
},
AccessControlList: []cos.ACLGrant{
{
Grantee: &cos.ACLGrantee{
Type: "RootAccount",
ID: "qcs::cam::uin/100000760461:uin/100000760461",
},
Permission: "FULL_CONTROL",
},
},
},
}
_, err = client.Object.PutACL(context.Background(), key, opt)
```

```
if err != nil {
panic(err)
}
}
```

参数说明

```
type ACLHeaderOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
}
```

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
XCosACL	设置 Object 的 ACL，如 private，public-read	string	否
XCosGrantFullControl	赋予被授权者所有的权限。格式：id="[OwnerUin]"	string	否
XCosGrantRead	赋予被授权者读的权限。格式：id="[OwnerUin]"	string	否
ACLXML	赋予指定账户对 Bucket 的访问权限，具体格式见 get object acl 返回结果说明	struct	否

查询对象 ACL

功能说明

查询 Object（文件/对象）的 ACL（GET Object acl）。

方法原型

```
func (s *ObjectService) GetACL(ctx context.Context, key string) (*ObjectGetACLRes
ult, *Response, error)
```

请求示例

```
package main
import (
```

```
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
key := "exampleobject"
_, _, err := client.Object.GetACL(context.Background(), key)
if err != nil {
panic(err)
}
}
```

参数说明

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是

返回结果说明

```
type ACLXml struct {
Owner *Owner
AccessControlList []ACLGrant
}
type Owner struct {
ID string
DisplayName string
}
type ACLGrant struct {
Grantee *ACLGrantee
Permission string
}
type ACLGrantee struct {
Type string
ID string
DisplayName string
URI string
}
```

参数名称	参数描述	类型
Owner	Bucket 拥有者的信息，包括 DisplayName 和 ID	struct
AccessControlList	Bucket 权限授予者的信息，包括 Grantee和 Permission	struct
Grantee	权限授予者的信息，包括 DisplayName，Type，ID 和 URI	struct
Type	权限授予者的类型，类型为 CanonicalUser 或者 Group	string
ID	Type 为 CanonicalUser 时，对应权限授予者的 ID，格式为 qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]，如 qcs::cam::uin/100000000001:uin/100000000001, 当 Type 指定为 CanonicalUser 时 ID 必选	string
DisplayName	权限被授予者的名称，可以不填，或者和ID的值保持一致	string
URI	Type 为 Group 时，填入预设用户组的 URI，例如 http://cam.qcloud.com/groups/global/AllUsers， 详情请参见 ACL 概述	string
Permission	授予者所拥有的 Bucket 的权限，可选值有 FULL_CONTROL，WRITE，READ，分别对应读写权限、写权限、读权限	string

全球加速

最近更新时间：2023-02-08 14:36:23

简介

本文档提供关于全球加速的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket Accelerate	设置全球加速	启用或暂停存储桶的全球加速功能
GET Bucket Accelerate	查询全球加速	查询存储桶的全球加速功能配置信息

设置全球加速

功能说明

启用或者暂停存储桶的全球加速功能。

方法原型

```
func (s *BucketService) PutAccelerate(ctx context.Context, opt *BucketPutAccelera
teOptions) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/produc
t/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
```

版权所有：腾讯云计算（北京）有限责任公司

```
m")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
opt := &cos.BucketPutAccelerateOptions{
Status: "Enabled",
Type: "COS",
}
_, err := client.Bucket.PutAccelerate(context.Background(), opt)
if err != nil {
// ERROR
}
}
```

参数说明

```
type BucketPutAccelerateOptions struct {
Status string
}
```

参数名称	参数描述	类型	是否必填
Status	说明全球加速功能是否开启，枚举值：Suspended、Enabled	string	是

查询全球加速

功能说明

查询存储桶的全球加速功能配置。

方法原型

```
func (s *BucketService) GetAccelerate(ctx context.Context) (*BucketGetAccelerateResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
},
})
res, _, err := client.Bucket.GetAccelerate(context.Background())
if err != nil {
// ERROR
}
fmt.Println(res)
}
```

返回结果说明

```
type BucketGetAccelerateResult struct {
Status string
}
```

参数名称	参数描述	类型	是否必填
Status	说明全球加速功能是否开启，枚举值：Suspended、Enabled	string	是

防盗链

最近更新时间：2023-02-08 14:36:23

简介

本文档提供关于存储桶 Referer 白名单或者黑名单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket referer	设置存储桶 Referer	设置存储桶 Referer 白名单或者黑名单
GET Bucket referer	查询存储桶 Referer	查询存储桶 Referer 白名单或者黑名单

设置存储桶 Referer

功能说明

设置指定存储桶的 Referer 白名单或者黑名单（PUT Bucket referer）。

方法原型

```
func (s *BucketService) PutReferer(ctx context.Context, opt *BucketPutRefererOpti
ons) (*Response, error)
```

请求示例

```
package main
import (
"context"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.
intl.cloud.tencent.com/ ，关于地域的详情见 https://intl.cloud.tencent.com/document/p
roduct/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
```

版权所有：腾讯云计算（北京）有限责任公司

```
m")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
},
})
opt := &cos.BucketPutRefererOptions{
Status: "Enabled",
RefererType: "White-List",
DomainList: []string{
"*.qq.com",
"*.qcloud.com",
},
EmptyReferConfiguration: "Allow",
}
_, err := client.Bucket.PutReferer(context.Background(), opt)
}
```

参数说明

```
type BucketPutRefererOptions struct {
Status string
RefererType string
DomainList []string
EmptyReferConfiguration string
}
```

参数名	参数描述	类型
Status	是否开启防盗链，枚举值：Enabled、Disabled	String
RefererType	防盗链类型，枚举值：Black-List、White-List	String
DomainList	生效域名，支持带端口和 IP、支持通配符*, 支持多条	Array

参数名	参数描述	类型
EmptyReferConfiguration	是否允许空 Refer 访问，枚举值：Allow、Deny	String

查询存储桶 Referer

功能说明

查询指定存储桶 Referer 白名单或者黑名单（GET Bucket referer）。

方法原型

```
func (s *BucketService) GetReferer(ctx context.Context) (*BucketGetRefererResult, *Response, error)
```

请求示例

```
package main
import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"os"
)
func main() {
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
// 替换为用户的 region，存储桶 region 可以在 COS 控制台“存储桶概览”查看 https://console.intl.cloud.tencent.com/ ，关于地域的详情见 https://intl.cloud.tencent.com/document/product/436/6224 。
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
// 通过环境变量获取密钥
// 环境变量 SECRETID 表示用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
SecretID: os.Getenv("SECRETID"), // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
// 环境变量 SECRETKEY 表示用户的 SecretKey，登录访问管理控制台查看密钥，https://console.i
```

```
ntl.cloud.tencent.com/cam/capi
SecretKey: os.Getenv("SECRETKEY"), // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
},
})
res, _, err := client.Bucket.GetReferer(context.Background())
if err != nil {
fmt.Println(err)
}
fmt.Println(res)
}
```

返回结果说明

```
type BucketGetRefererResult struct {
Status string
RefererType string
DomainList []string
EmptyReferConfiguration string
}
```

参数名	参数描述	类型
Status	是否开启防盗链，枚举值：Enabled、Disabled	String
RefererType	防盗链类型，枚举值：Black-List、White-List	String
DomainList	生效域名，支持带端口和 IP、支持通配符*, 支持多条	Array
EmptyReferConfiguration	是否允许空 Refer 访问，枚举值： Allow、Deny	String

数据校验
CRC64 校验

最近更新时间：2023-11-14 10:17:54

简介

数据在客户端和服务器间传输时可能会出现错误，COS 除了可以通过 MD5 和自定义属性 验证数据完整性外，还可以通过 CRC64 检验码来进行数据校验。

COS 会对新上传的对象进行 CRC64 计算，并将结果作为对象的属性进行存储，随后在返回的响应头部中携带 x-cos-hash-crc64ecma，该头部表示上传对象的 CRC64 值，根据 ECMA-182标准 计算得到。对于 CRC64 特性上线前就已经存在于 COS 的对象，COS 不会对其计算 CRC64 值，所以获取此类对象时不会返回其 CRC64 值。

操作说明

目前支持 CRC64 的 API 如下：

- 简单上传接口
  - PUT Object 和 POST Object ：用户可在返回的响应头中获得文件 CRC64 校验值。
- 分块上传接口
  - Upload Part：用户可以根据 COS 返回的 CRC64 值与本地计算的数值进行比较验证。
  - Complete Multipart Upload：如果每个分块都有 CRC64 属性，则会返回整个对象的 CRC64 值，如果某些分块不具备 CRC64 值，则不返回。
- 执行 Upload Part - Copy 时，会返回对应的 CRC64 值。
- 执行 PUT Object - Copy 时，如果源对象存在 CRC64 值，则返回 CRC64，否则不返回。
- 执行 HEAD Object 和 GET Object 时，如果对象存在 CRC64，则返回。用户可以根据 COS 返回的 CRC64 值和本地计算的 CRC64 进行比较验证。

SDK 说明

SDK对应接口可通过响应头部获取 CRC64 值，SDK 上传文件时默认会进行 CRC64 校验。

> 注意：
> COS Go SDK 版本需要大于等于 v0.7.23。

版权所有：腾讯云计算（北京）有限责任公司

请求示例

```
package main

import (
"context"
"fmt"
"github.com/tencentyun/cos-go-sdk-v5"
"net/http"
"net/url"
"strings"
)

func main() {
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
u, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: "SECRETID", // 替换为用户的 SecretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
SecretKey: "SECRETKEY", // 替换为用户的 SecretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
},
})
// 关闭 CRC64 校验，CRC64 默认开启，强烈不建议用户关闭
// client.Conf.EnableCRC = false

name := "exampleobject"
// 通过字符串上传对象
f := strings.NewReader("test")
// SDK 会自动进行 CRC64 校验。
resp, err := client.Object.Put(context.Background(), name, f, nil)
if err != nil {
// ERROR
}
// 根据响应头部获取 CRC64
fmt.Printf("CRC64: %v\n", resp.Header.Get("x-cos-hash-crc64ecma"))
}
```

返回结果说明

参数名称	参数描述	类型
Response	HTTP 响应	Struct

参数名称	参数描述	类型
Response.Header	HTTP 响应头部	Struct
Response.Body	HTTP 响应数据	Struct

版权所有：腾讯云计算（北京）有限责任公司

文件处理
文件压缩

最近更新时间：2023-03-28 14:53:03

简介

本文档提供关于数据万象文件压缩的 API 概览以及 SDK 示例代码。

API	操作描述
提交多文件打包压缩任务	以提交任务的方式进行多文件打包压缩任务
查询多文件打包压缩结果	主动查询指定的多文件打包压缩任务结果

提交多文件打包压缩任务

功能说明

多文件打包压缩功能可以将您的多个文件，打包为 zip 等压缩包格式，以提交任务的方式进行多文件打包压缩，异步返回打包后的文件。

方法原型

```
func (s *CIService) CreateFileProcessJob(ctx context.Context, opt *FileProcessJob
Options) (*FileProcessJobResult, *Response, error)
```

请求示例

```
createJobOpt := &cos.FileProcessJobOptions{
Tag: "FileCompress",
Operation: &cos.FileProcessJobOperation{
FileCompressConfig: &cos.FileCompressConfig{
Format: "zip",
Flatten: "0",
Key: []string{"1.mp3", "2.jpg"},
},
Output: &cos.FileProcessOutput{
Region: "ap-shanghai",
Bucket: "test-1250000000",
```

版权所有：腾讯云计算（北京）有限责任公司

```
Object: "vvvvxxxzz.zip",
},
},
QueueId: "p111a8dd208104ce3b11c78398f658ca8",
}
res, _, err := c.CI.CreateFileProcessJob(context.Background(), createJobOpt)
```

参数说明

```
type FileProcessJobOptions struct {
XMLName xml.Name
Tag string
Input *FileProcessInput
Operation *FileProcessJobOperation
QueueId string
CallBackFormat string
CallBackType string
CallBack string
CallBackMqConfig *NotifyConfigCallBackMqConfig
}
type FileProcessInput struct {
Region string
Bucket string
Object string
}
type FileProcessJobOperation struct {
FileHashCodeConfig *FileHashCodeConfig
FileHashCodeResult *FileHashCodeResult
FileUncompressConfig *FileUncompressConfig
FileUncompressResult *FileUncompressResult
FileCompressConfig *FileCompressConfig
FileCompressResult *FileCompressResult
Output *FileProcessOutput
UserData string
}
type FileHashCodeConfig struct {
Type string
AddToHeader bool
}
type FileHashCodeResult struct {
MD5 string
SHA1 string
SHA256 string
FileSize int
LastModified string
Etag string
```

```
}
type FileUncompressConfig struct {
Prefix string
PrefixReplaced string
}
type FileUncompressResult struct {
Region string
Bucket string
FileCount string
}
type FileCompressConfig struct {
Flatten string
Format string
UrlList string
Prefix string
Key []string
}
type FileCompressResult struct {
Region string
Bucket string
Object string
}
type FileProcessOutput struct {
Region string
Bucket string
Object string
}
type NotifyConfigCallBackMqConfig struct {
MqMode string
MqRegion string
MqName string
}
```

参数名称	描述	类型	是否必选
Tag	创建任务的 Tag，目前仅支持：FileCompress	String	是
Input	待操作的文件对象。详情见 Input	Container	是
Operation	操作规则	Container	是
Operation.FileCompressConfig	指定文件打包压缩的处理规则。详情见 FileCompressConfig	Container	是
Operation.FileCompressResult	多文件打包压缩的结果，任务未完成时不返回。详情见 FileCompressResult	Container	否

参数名称	描述	类型	是否必选
Operation.Output	指定文件处理后的文件保存的地址信息。详情见 Output	Container	是
Operation.UserData	透传用户信息, 可打印的 ASCII 码, 长度不超过 1024	String	否
QueueId	任务所在的队列 ID	String	是
CallBackFormat	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式。	String	否
CallBackType	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型。	String	否
CallBack	任务回调的地址，优先级高于队列的回调地址。	String	否
CallBackMqConfig	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

返回结果说明

```
type FileProcessJobResult struct {
JobsDetail FileProcessJobResult
}
type FileProcessJobResult struct {
Code string
Message string
JobId string
Tag string
State string
CreationTime string
StartTime string
EndTime string
QueueId string
Input *FileProcessInput
Operation *FileProcessJobOperation
}
```

参数名称	描述	类型
JobsDetail	任务的详细信息	Container
Code	错误码，只有 State 为 Failed 时有意义	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
Message	错误描述，只有 State 为 Failed 时有意义	String
JobId	新创建任务的 ID	String
Tag	新创建任务的 Tag：FileCompress	String
State	任务的状态，为 Submitted、Running、Success、Failed、Pause、Cancel 其中一个	String
CreationTime	任务的创建时间	String
StartTime	任务的开始时间	String
EndTime	任务的结束时间	String
QueueId	任务所属的队列 ID	String
Input	该任务的输入文件路径，详情请参见 FileProcessJobOptions.Input	Container
Operation	该任务的规则，详情请参见 FileProcessJobOptions.Operation	Container

查询多文件打包压缩结果

功能说明

查询一个文件处理任务,根据任务 ID 查询任务详情。

方法原型

```
func (s *CIService) DescribeFileProcessJob(ctx context.Context, jobid string) (*FileProcessJobResult, *Response, error)
```

请求示例

```
jobid := "<jobid>"
res, _, err := c.CI.DescribeFileProcessJob(context.Background(), jobid)
```

参数说明

参数名称	描述	类型	是否必选
jobid	文件处理任务 ID	String	是

返回结果说明

```
type FileProcessJobResult struct {
JobsDetail *FileProcessJobsDetail
}
```

参数名称	描述	类型
JobsDetail	任务的详细信息， 同 CreateFileProcessJob 接口的 Response.JobsDetail 节点	Container

文件哈希值计算

最近更新时间：2023-03-28 14:38:56

简介

本文档提供关于数据万象文件哈希值计算的 API 概览以及 SDK 示例代码。

API	操作描述
哈希值计算同步请求	以同步请求的方式进行文件哈希值计算，实时返回计算得到的哈希值
提交哈希值计算任务	以提交任务的方式进行文件哈希值计算任务
查询哈希值计算结果	主动查询指定的文件哈希值计算任务结果

哈希值计算同步请求

功能说明

以同步请求的方式进行文件哈希值计算，返回得到的哈希值。

方法原型

```
func (s *CIService) GetFileHash(ctx context.Context, name string, opt *GetFileHashOptions) (*GetFileHashResult, *Response, error)
```

请求示例

```
opt := &cos.GetFileHashOptions{
CIProcess: "filehash",
Type: "md5",
}
res, _, err := c.CI.GetFileHash(context.Background(), "test.jpg", opt)
```

参数说明

```
type GetFileHashOptions struct {
CIProcess string
Type string
```

```
AddToHeader bool
}
```

参数名称	描述	类型	是否必选
name	执行操作的文件全路径	String	是
CIProcess	操作类型，哈希值计算固定为：filehash	String	是
Type	支持的哈希算法类型，有效值：md5、sha1、sha256	String	是
AddToHeader	是否将计算得到的哈希值，自动添加至文件的自定义header，格式为：x-cos-meta-md5/sha1/sha256; 有效值： true、false，不填则默认为false。	Bool	否

返回结果说明

```
type GetFileHashResult struct {
FileHashCodeResult *FileHashCodeResult
Input *FileProcessInput
}
```

参数名称	描述	类型
FileHashCodeResult	文件哈希值的结果。详情请参见 FileProcessJobOptions.Input	Container
Input	输入文件的基本信息。详情请参见 FileProcessJobOptions.Operation.FileHashCodeResult	Container

提交哈希值计算任务

功能说明

以提交任务的方式进行文件哈希值计算，异步返回计算得到的哈希值。

方法原型

```
func (s *CIService) CreateFileProcessJob(ctx context.Context, opt *FileProcessJob
Options) (*FileProcessJobResult, *Response, error)
```

请求示例

```
createJobOpt := &cos.FileProcessJobOptions{
Tag: "FileHashCode",
```

```
Input: &cos.FileProcessInput{
Object: "294028.zip",
},
Operation: &cos.FileProcessJobOperation{
FileHashCodeConfig: &cos.FileHashCodeConfig{
Type: "sha1",
AddToHeader: true,
},
},
QueueId: "p111a8dd208104ce3b11c78398f658ca8",
}
res, _, err := c.CI.CreateFileProcessJob(context.Background(), createJobOpt)
```

参数说明

```
type FileProcessJobOptions struct {
XMLName xml.Name
Tag string
Input *FileProcessInput
Operation *FileProcessJobOperation
QueueId string
CallBackFormat string
CallBackType string
CallBack string
CallBackMqConfig *NotifyConfigCallBackMqConfig
}
type FileProcessInput struct {
Region string
Bucket string
Object string
}
type FileProcessJobOperation struct {
FileHashCodeConfig *FileHashCodeConfig
FileHashCodeResult *FileHashCodeResult
FileUncompressConfig *FileUncompressConfig
FileUncompressResult *FileUncompressResult
FileCompressConfig *FileCompressConfig
FileCompressResult *FileCompressResult
Output *FileProcessOutput
UserData string
}
type FileHashCodeConfig struct {
Type string
AddToHeader bool
}
type FileHashCodeResult struct {
```

版权所有：腾讯云计算（北京）有限责任公司

```
MD5 string
SHA1 string
SHA256 string
FileSize int
LastModified string
Etag string
}
type FileUncompressConfig struct {
Prefix string
PrefixReplaced string
}
type FileUncompressResult struct {
Region string
Bucket string
FileCount string
}
type FileCompressConfig struct {
Flatten string
Format string
UrlList string
Prefix string
Key string
}
type FileCompressResult struct {
Region string
Bucket string
Object string
}
type FileProcessOutput struct {
Region string
Bucket string
Object string
}
type NotifyConfigCallBackMqConfig struct {
MqMode string
MqRegion string
MqName string
}
```

参数名称	描述	类型	是否必选
Tag	创建任务的 Tag，目前仅支持：FileHashCode	String	是
Input	待操作的文件对象。详情见 Input	Container	是
Operation	操作规则	Container	是

参数名称	描述	类型	是否必选
Operation.FileHashCodeConfig	指定哈希值计算的处理规则。详情见 FileHashCodeConfig	Container	是
Operation.FileHashCodeResult	计算得到的文件 hash 值信息，任务未完成时不返回。详情见 FileHashCodeResult	Container	否
Operation.UserData	透传用户信息, 可打印的 ASCII 码, 长度不超过 1024	String	否
QueueId	任务所在的队列 ID	String	是
CallBackFormat	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式。	String	否
CallBackType	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型。	String	否
CallBack	任务回调的地址，优先级高于队列的回调地址。	String	否
CallBackMqConfig	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

返回结果说明

```
type FileProcessJobResult struct {
JobsDetail FileProcessJobResult
}
type FileProcessJobResult struct {
Code string
Message string
JobId string
Tag string
State string
CreationTime string
StartTime string
EndTime string
QueueId string
Input *FileProcessInput
Operation *FileProcessJobOperation
}
```

参数名称	描述	类型
JobsDetail	任务的详细信息	Container

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
Code	错误码，只有 State 为 Failed 时有意义	String
Message	错误描述，只有 State 为 Failed 时有意义	String
JobId	新创建任务的 ID	String
Tag	新创建任务的 Tag：FileHashCode	String
State	任务的状态，为 Submitted、Running、Success、Failed、Pause、Cancel 其中一个	String
CreationTime	任务的创建时间	String
StartTime	任务的开始时间	String
EndTime	任务的结束时间	String
QueueId	任务所属的队列 ID	String
Input	该任务的输入文件路径，详情请参见 FileProcessJobOptions.Input	Container
Operation	该任务的规则，详情请参见 FileProcessJobOptions.Operation	Container

查询哈希值计算结果

功能说明

查询一个文件处理任务,根据任务 ID 查询任务详情。

方法原型

```
func (s *CIService) DescribeFileProcessJob(ctx context.Context, jobid string) (*F
ileProcessJobResult, *Response, error)
```

请求示例

```
jobid := "<jobid>"
res, _, err := c.CI.DescribeFileProcessJob(context.Background(), jobid)
```

参数说明

参数名称	描述	类型

参数名称	描述	类型
jobid	文件处理任务 ID	String

返回结果说明

```
type FileProcessJobResult struct {
JobsDetail *FileProcessJobsDetail
}
```

参数名称	描述	类型	是否必选
JobsDetail	任务的详细信息， 同 CreateFileProcessJob 接口的 Response.JobsDetail 节点	Container	是

文件解压

最近更新时间：2023-03-28 14:53:46

简介

本文档提供关于数据万象文件解压的 API 概览以及 SDK 示例代码。

API	操作描述
提交文件解压任务	以提交任务的方式进行文件解压任务
查询文件解压结果	主动查询指定的文件解压任务结果

提交文件解压任务

功能说明

以提交任务的方式进行文件解压缩，异步返回压缩包内被解压出来的文件。

方法原型

```
func (s *CIService) CreateFileProcessJob(ctx context.Context, opt *FileProcessJob
Options) (*FileProcessJobResult, *Response, error)
```

请求示例

```
createJobOpt := &cos.FileProcessJobOptions{
Tag: "FileUncompress",
Input: &cos.FileProcessInput{
Object: "294028.zip",
},
Operation: &cos.FileProcessJobOperation{
FileUncompressConfig: &cos.FileUncompressConfig{
Prefix: "uncomp",
PrefixReplaced: "1",
},
Output: &cos.FileProcessOutput{
Region: "ap-beijing",
Bucket: "test-1250000000",
}
},
```

```
QueueId: "p111a8dd208104ce3b11c78398f658ca8",
}
res, _, err := c.CI.CreateFileProcessJob(context.Background(), createJobOpt)
```

参数说明

```
type FileProcessJobOptions struct {
XMLName xml.Name
Tag string
Input *FileProcessInput
Operation *FileProcessJobOperation
QueueId string
CallBackFormat string
CallBackType string
CallBack string
CallBackMqConfig *NotifyConfigCallBackMqConfig
}
type FileProcessInput struct {
Region string
Bucket string
Object string
}
type FileProcessJobOperation struct {
FileHashCodeConfig *FileHashCodeConfig
FileHashCodeResult *FileHashCodeResult
FileUncompressConfig *FileUncompressConfig
FileUncompressResult *FileUncompressResult
FileCompressConfig *FileCompressConfig
FileCompressResult *FileCompressResult
Output *FileProcessOutput
UserData string
}
type FileHashCodeConfig struct {
Type string
AddToHeader bool
}
type FileHashCodeResult struct {
MD5 string
SHA1 string
SHA256 string
FileSize int
LastModified string
Etag string
}
type FileUncompressConfig struct {
Prefix string
```

```
PrefixReplaced string
}
type FileUncompressResult struct {
Region string
Bucket string
FileCount string
}
type FileCompressConfig struct {
Flatten string
Format string
UrlList string
Prefix string
Key string
}
type FileCompressResult struct {
Region string
Bucket string
Object string
}
type FileProcessOutput struct {
Region string
Bucket string
Object string
}
type NotifyConfigCallBackMqConfig struct {
MqMode string
MqRegion string
MqName string
}
```

参数名称	描述	类型	是否必选
Tag	创建任务的 Tag，目前仅支持：FileUncompress	String	是
Input	待操作的文件对象。详情见 Input	Container	是
Operation	操作规则	Container	是
Operation.FileUncompressConfig	指定文件解压的处理规则。详情见 FileUncompressConfig	Container	是
Operation.FileUncompressResult	文件解压的结果，任务未完成时不返回。详情见 FileUncompressResult	Container	否
Operation.Output	指定文件处理后的文件保存的地址信息。详情见 Output	Container	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型	是否必选
Operation.UserData	透传用户信息, 可打印的 ASCII 码, 长度不超过 1024	String	否
QueueId	任务所在的队列 ID	String	是
CallBackFormat	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式。	String	否
CallBackType	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型。	String	否
CallBack	任务回调的地址，优先级高于队列的回调地址。	String	否
CallBackMqConfig	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

返回结果说明

```
type FileProcessJobResult struct {
JobsDetail FileProcessJobResult
}
type FileProcessJobResult struct {
Code string
Message string
JobId string
Tag string
State string
CreationTime string
StartTime string
EndTime string
QueueId string
Input *FileProcessInput
Operation *FileProcessJobOperation
}
```

参数名称	描述	类型
JobsDetail	任务的详细信息	Container
Code	错误码，只有 State 为 Failed 时有意义	String
Message	错误描述，只有 State 为 Failed 时有意义	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
JobId	新创建任务的 ID	String
Tag	新创建任务的 Tag：FileUncompress	String
State	任务的状态，为 Submitted、Running、Success、Failed、Pause、Cancel 其中一个	String
CreationTime	任务的创建时间	String
StartTime	任务的开始时间	String
EndTime	任务的结束时间	String
QueueId	任务所属的队列 ID	String
Input	该任务的输入文件路径，详情请参见 FileProcessJobOptions.Input	Container
Operation	该任务的规则，详情请参见 FileProcessJobOptions.Operation	Container

查询文件解压结果

功能说明

查询一个文件处理任务,根据任务 ID 查询任务详情。

方法原型

```
func (s *CIService) DescribeFileProcessJob(ctx context.Context, jobid string) (*FileProcessJobResult, *Response, error)
```

请求示例

```
jobid := "<jobid>"
res, _, err := c.CI.DescribeFileProcessJob(context.Background(), jobid)
```

参数说明

参数名称	描述	类型	是否必选
jobid	文件处理任务 ID	String	是

返回结果说明

版权所有：腾讯云计算（北京）有限责任公司

```
type FileProcessJobResult struct {
JobsDetail *FileProcessJobsDetail
}
```

参数名称	描述	类型
JobsDetail	任务的详细信息， 同 CreateFileProcessJob 接口的 Response.JobsDetail 节点	Container

版权所有：腾讯云计算（北京）有限责任公司

异常处理

最近更新时间：2022-01-04 12:17:06

简介

API 返回的 Response 为 Golang HTTP 标准库 Response 类型。用户可通过 err.Error() 获取错误提示，服务端返回的具体信息，请参见 COS 错误码。

服务端异常

API 返回的 Response 结构中包含调用结构，如下所示：

成员	描述	类型
X-Cos-Request-Id	Response 中响应头，请求 ID，用于表示一个请求，对于排查问题十分重要	string
StatusCode	Response 的 status 状态码，4xx是指请求因客户端而失败，5xx是服务端异常导致的失败，详情请参见 COS 错误码	string

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。
   COS 自助诊断工具
   点击自助诊断
   输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**。
4. 稍侯片刻，便能看到相应的智能诊断结果。

版权所有：腾讯云计算（北京）有限责任公司

设置访问域名（CDN/全球加速）

最近更新时间：2022-01-06 16:17:43

简介

本文档提供关于如何使用非默认域名请求对象存储（Cloud Object Storage，COS）服务。

设置默认 COS 域名

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION修改为真实的信息
u, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
// 用于Get Service 查询，默认全地域 service.cos.myqcloud.com
su, _ := url.Parse("https://cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: u, ServiceURL: su}
// 1.永久密钥
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: "SECRETID",
SecretKey: "SECRETKEY",
},
})
```

默认 CDN 加速域名

功能说明

在 COS 控制台为存储桶开启默认 CDN 加速域名后，可在 SDK 代码中设置对应的域名，以便实现默认加速功能。关于如何开启默认加速域名请参见 开启默认 CDN 加速域名。

请求示例

```
// 将 examplebucket-1250000000 修改为真实的信息
u, _ := url.Parse("https://examplebucket-1250000000.file.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
// 使用 CDN 加速域名，无需再发送 COS 的认证信息
client := cos.NewClient(b, &http.Client{})
```

自定义 CDN 加速域名

功能说明

在 COS 控制台为存储桶设置自定义 CDN 加速域名后，可在 SDK 代码中设置对应的域名，以便实现自定义域名的加速功能。关于如何开启自定义 CDN 加速域名，请参见 开启自定义 CDN 加速域名。

请求示例

```
// 设置自定义cdn加速域名
u, _ := url.Parse("https://xxx.xxx.com")
b := &cos.BaseURL{BucketURL: u}
// 使用 CDN 加速域名，无需再发送 COS 的认证信息
client := cos.NewClient(b, &http.Client{})
```

自定义源站域名

请求示例

```
// 自定义源站域名
u, _ := url.Parse("https://xxx.xxx.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: "SECRETID",
SecretKey: "SECRETKEY",
},
})
```

全球加速域名

请求示例

```
// 全球加速域名
// 将 examplebucket-1250000000 修改为真实的信息
u, _ := url.Parse("https://examplebucket-1250000000.cos.accelerate.myqcloud.com")
b := &cos.BaseURL{BucketURL: u}
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: "SECRETID",
SecretKey: "SECRETKEY",
},
})
```

版权所有：腾讯云计算（北京）有限责任公司

图片处理
图片持久化处理

最近更新时间：2022-10-09 16:01:23

简介

本文档提供关于图片持久化处理的 API 概览以及 SDK 示例代码。

API	说明
图片持久化处理	用于图片持久化处理

上传时处理

COS 提供的上传时处理功能可以帮助使用者在上传时实现图片处理。您只需要在请求包头部中加入 Pic-Operations 项并设置好相应参数，即可在图片上传时实现相应的图片处理，并可将原图和处理结果存入到 COS。目前支持20M以内文件处理。

请求示例

```
opt := &cos.ObjectPutOptions{
nil,
&cos.ObjectPutHeaderOptions{
XOptionHeader: &http.Header{},
},
}
pic := &cos.PicOperations{
IsPicInfo: 1,
Rules: []cos.PicOperationsRules{
{
FileId: "format.jpg",
Rule: "imageView2/format/png",
},
},
}
opt.XOptionHeader.Add("Pic-Operations", cos.EncodePicOperations(pic))
name := "test.jpg"
local_filename := "./test.jpg"
res, _, err := c.CI.PutFromFile(context.Background(), name, local_filename, opt)
```

版权所有：腾讯云计算（北京）有限责任公司

参数说明

```
type PicOperations struct {
IsPicInfo int
Rules []PicOperationsRules
}
type PicOperationsRules struct {
Bucket string
FileId string
Rule string
}
```

参数名称	参数描述	类型	是否必填
IsPicInfo	是否返回原图信息，0不返回原图信息，1返回原图信息，默认为0	int	否
Rules	处理规则，一条规则对应一个处理结果（目前支持五条规则），不填则不进行图片处理	Array	否
Bucket	存储结果的目标存储桶名称，格式为 BucketName-APPID，如果不指定的话默认保存到当前存储桶	string	否
FileId	处理结果的文件路径名称，如以 `/` 开头，则存入指定文件夹中，否则，存入原图文件存储的同目录	string	是
Rule	处理参数，参见图片持久化处理 API。若按指定样式处理，则以 `style/` 开头，后加样式名，如样式名为 `test`，则 rule 字段为 `style/test`	string	否

云上数据处理

该接口能够对已存储在 COS 的图片进行相应处理操作，并将结果存入到 COS。

请求示例

```
opt := &cos.ImageProcessOptions{
IsPicInfo: 1,
Rules: []cos.PicOperationsRules{
{
FileId: "format.jpg",
Rule: "imageView2/format/png",
},
},
}
```

```
name := "test.jpg"
res, _, err := c.CI.ImageProcess(context.Background(), name, opt)
```

版权所有：腾讯云计算（北京）有限责任公司

基础图片处理

最近更新时间：2022-10-09 16:01:23

简介

本文档提供关于基础图片处理的 API 概览以及 SDK 示例代码。

API	操作描述
缩放	对图片进行缩小或放大
裁剪	对图片进行裁剪，包括普通裁剪、缩放裁剪、内切圆裁剪、圆角裁剪和人脸智能裁剪
旋转	对图片进行旋转，包括普通旋转和自适应旋转
格式转换	对图片进行格式转换、gif 格式优化、渐进显示
质量变换	对图片质量进行调节
高斯模糊	对图片进行模糊处理
锐化	对图片进行锐化
图片水印	对图片进行水印处理
文字水印	对图片进行实时文字水印处理
获取图片基本信息	查询图片基本信息，包括格式、长、宽等
获取图片 EXIF	查询 EXIF 信息
获取图片主色调	查询图片主色调信息
去除元信息	去除图片元信息，包括 exif 信息
快速缩略模板	通过图片处理模板，生成相应的缩略图
管道操作符	实现对图片按顺序进行多种处理

缩放

功能说明

腾讯云数据万象通过 **imageMogr2** 接口提供图片缩放功能。

方法原型

```
func (s *CIService) Get(ctx context.Context, key string, operation string, opt *ObjectGetOptions, id ...string) (*Response, error)
func (s *CIService) GetToFile(ctx context.Context, key, localpath, operation string, opt *ObjectGetOptions, id ...string) (*Response, error)
```

请求示例

```
name := "test.jpg"
// Case 1 从响应体中获取对象
resp, err := c.CI.Get(context.Background(), name, "imageMogr2/thumbnail/!50px", nil)
if err != nil {
//ERROR
}
defer resp.Body.Close()
ioutil.ReadAll(resp.Body)
// Case 2 下载对象到文件
filepath := "test.jpg"
_, err = c.CI.GetToFile(context.Background(), name, filepath, "imageMogr2/thumbnail/!50px", nil)
```

参数说明

参数名称	参数描述	类型	是否必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
operation	基础图片处理，可以通过该参数实现缩放、裁剪、旋转、格式转换、质量变换等基础图片处理功能。	string	是
ObjectGetOptions	对象下载参数，详见 下载对象	string	否
id	对象 VersionId	string	否

裁剪

功能说明

腾讯云数据万象通过 **imageMogr2** 接口提供裁剪功能，包括普通裁剪、缩放裁剪、内切圆裁剪、圆角裁剪和人脸智能裁剪。

请求示例

```
name := "test.jpg"
filepath := "test.jpg"
_, err = c.CI.GetToFile(context.Background(), name, filepath, "imageMogr2/cut/600x600x100x10", nil)
```

其他基础图片处理

功能说明

基础图片处理统一使用上述方法完成，其他基础图片处理只需要修改 operation 参数的值即可，例如将图片顺时针旋转90度的样例如下。

```
name := "test.jpg"
filepath := "test.jpg"
_, err = c.CI.GetToFile(context.Background(), name, filepath, "imageMogr2/rotate/90", nil)
```

版权所有：腾讯云计算（北京）有限责任公司

图片高级压缩

最近更新时间：2022-10-09 16:01:23

简介

本文档提供关于图片高级压缩的 API 概览以及 SDK 示例代码。

API	操作描述
图片高级压缩	图片高级压缩可以更加高效地将图片转换为 TPG 或 HEIF 等高压缩比格式，有效降低图片传输链路及加载耗时，降低带宽及流量成本

请求示例

```
// 将原图转换为 TPG 格式
name := "test.png"
filepath := "test.tpg"
_, err := c.CI.GetToFile(context.Background(), name, filepath, "imageMogr2/format/tpg", nil)
if err != nil {
// ERROR
}
// 将原图转换为 HEIF 格式
filepath = "test.heif"
_, err = c.CI.GetToFile(context.Background(), name, filepath, "imageMogr2/format/heif", nil)
if err != nil {
// ERROR
}
```

版权所有：腾讯云计算（北京）有限责任公司

Guetzli 压缩

最近更新时间：2022-10-13 12:10:39

简介

本文档提供关于 Guetzli 压缩的相关的 API 概览以及 SDK 示例代码。

API	操作描述
开通 Guetzli 压缩	对 Bucket 开通 Guetzli 压缩功能
查询 Guetzli 状态	用于查询 Guetzli 压缩功能是否开启
关闭 Guetzli 压缩	用于关闭 Guetzli 压缩功能

开通 Guetzli 压缩

功能说明

对 Bucket 开通 Guetzli 压缩功能。

方法原型

```
func (s *CIService) PutGuetzli(ctx context.Context) (*Response, error)
```

请求示例

```
// 需要填写CIURL
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
cu, _ := url.Parse("http://examplebucket-1250000000.pic.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
},
})
_, err := c.CI.PutGuetzli(context.Background())
log_status(err)
```

版权所有：腾讯云计算（北京）有限责任公司

查询 Guetzli 状态

功能说明

用于查询 Guetzli 压缩功能是否开启。

方法原型

```
func (s *CIService) GetGuetzli(ctx context.Context) (*GetGuetzliResult, *Respons
e, error)
```

请求示例

```
// 需要填写 CIURL
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.co
m")
cu, _ := url.Parse("http://examplebucket-1250000000.pic.ap-guangzhou.myqcloud.co
m")
b := &cos.BaseURL{BucketURL: u, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
},
})
res, _, err := c.CI.GetGuetzli(context.Background())
if err == nil {
fmt.Printf("%v\n", res.GuetzliStatus)
}
```

结果说明

```
type GetGuetzliResult struct {
GuetzliStatus string
}
```

节点名称	描述	类型
GuetzliStatus	是否开启 Guetzli 压缩功能，拥有 `on` 与 `off` 两种状态。	String

版权所有：腾讯云计算（北京）有限责任公司

关闭 Guetzli 压缩

用于关闭 Guetzli 压缩功能。

方法原型

```
func (s *CIService) DeleteGuetzli(ctx context.Context) (*Response, error)
```

请求示例

```
// 需要填写CIURL
u, _ := url.Parse("https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com")
cu, _ := url.Parse("http://examplebucket-1250000000.pic.ap-guangzhou.myqcloud.com")
b := &cos.BaseURL{BucketURL: u, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
},
})
_, err := c.CI.DeleteGuetzli(context.Background())
```

版权所有：腾讯云计算（北京）有限责任公司

盲水印

最近更新时间：2022-10-09 16:01:24

简介

本文档提供关于盲水印的相关的 API 概览以及 SDK 示例代码。

API	操作描述
盲水印	对本地图片添加或提取盲水印并上传至存储桶

添加盲水印

功能说明

盲水印支持在上传时添加以及下载时添加。

请求示例1: 上传时添加盲水印

```
opt := &cos.ObjectPutOptions{
nil,
&cos.ObjectPutHeaderOptions{
XOptionHeader: &http.Header{},
},
}
pic := &cos.PicOperations{
IsPicInfo: 1,
Rules: []cos.PicOperationsRules{
{
FileId: ".jpg",
Rule: "watermark/3/type/3/text/" + base64.StdEncoding.EncodeToString([]byte("test
watermark")),
},
},
}
opt.XOptionHeader.Add("Pic-Operations", cos.EncodePicOperations(pic))
name := "test.jpg"
filepath := "./test.jpg"
res, _, err := c.CI.PutFromFile(context.Background(), name, filepath, opt)
```

请求示例2：下载时添加盲水印

```
name = "test.jpg"
filepath := "watermark.jpg"
_, err = c.CI.GetToFile(context.Background(), name, filepath, "watermark/3/type/
3/text/"+base64.StdEncoding.EncodeToString([]byte("testwatermark")), nil)
```

提取盲水印

提取盲水印的请求包与 COS 简单上传文件接口一致，只需在请求包头部增加图片处理参数 Pic-Operations 并使用提取盲水印参数（watermark/4）即可。

请求示例

```
opt := &cos.ObjectPutOptions{
nil,
&cos.ObjectPutHeaderOptions{
XOptionHeader: &http.Header{},
},
}
pic := &cos.PicOperations{
IsPicInfo: 1,
Rules: []cos.PicOperationsRules{
{
FileId: "format2.jpg",
Rule: "watermark/4/type/3/text/" + base64.StdEncoding.EncodeToString([]byte("test
watermark")),
},
},
}
opt.XOptionHeader.Add("Pic-Operations", cos.EncodePicOperations(pic))
name := "test2.jpg"
_, err := c.Object.PutFromFile(context.Background(), name, filepath, opt)
```

版权所有：腾讯云计算（北京）有限责任公司

内容审核
图片审核

最近更新时间：2022-12-20 14:15:16

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于图片审核的相关的 API 概览以及 SDK 示例代码。

API	操作描述
图片单次审核	借助数据万象的内容审核接口，实现对 COS 存量数据的涉黄、违法违规以及广告引导类图片的扫描
图片批量审核	借助数据万象的内容审核接口，实现对 COS 存量数据的涉黄、违法违规以及广告引导类图片的扫描
查询图片审核任务结果	用于查询图片审核任务详情

图片单次审核

功能说明

图片审核的存量扫描功能通过借助数据万象的持久化处理接口，实现对 COS 存量数据的涉黄、违法违规以及广告引导类图片的扫描。图片审核请求包属于 GET 请求。

方法原型

```
// 基本操作
func (s *CIService) ImageRecognition(ctx context.Context, name string, DetectType string) (*ImageRecognitionResult, *Response, error)
// 支持指定更多参数
func (s *CIService) ImageAuditing(ctx context.Context, name string, opt *ImageRecognitionOptions) (*ImageRecognitionResult, *Response, error)
```

请求示例

```
DetectType := "porn,ads"
name := "test.jpg"
res, _, err := c.CI.ImageRecognition(context.Background(), name, DetectType)
opt := &cos.ImageRecognitionOptions{
CIProcess: "sensitive-content-recognition",
DetectType: "porn,ads",
DetectUrl: "http://www.example.com/test.jpg",
BizType: "ce25f391a72e11eb99f********",
}
res, _, err := c.CI.ImageAuditing(context.Background(), "", opt)
```

参数说明

```
type ImageRecognitionOptions struct {
CIProcess string
DetectType string
DetectUrl string
Interval int
MaxFrames int
BizType string
LargeImageDetect int
DataId string
Async int
Callback string
}
```

参数名称	描述	类型	是否必选
name	COS 存储桶中的图片文件名称，COS 存储桶由Host指定，例如在北京的 examplebucket-1250000000 存储桶中的目录 test 下的文件 img.jpg，则 Host 填写 `examplebucket-1250000000.cos.ap-beijing.myqcloud.com`，ObjectKey填写 `test/img.jpg`。如使用 ImageAuditing 接口审核 url 图片链接，传空字符串。	String	是

参数名称	描述	类型	是否必选
CIProcess	标识数据处理功能的字段，内容审核的值为：`sensitive-content-recognition`。	String	是
BizType	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置审核策略。	String	否
DetectType	审核类型，拥有 porn（涉黄识别）、ads（广告识别）。用户可选择多种识别类型，例如 detect-type=porn,ads 表示对图片进行涉黄及广告审核。如您有更多场景的审核需要，请使用 BizType 参数。	String	否
DetectUrl	您可以通过填写 detect-url 审核任意公网可访问的图片链接。不填写 detect-url 时，后台会默认审核 ObjectKey。填写了 detect-url 时，后台会审核 detect-url 链接，无需再填写 ObjectKey。detect-url 示例：`http://www.example.com/abc.jpg`。	String	否
Interval	审核 GIF 动图时，可使用该参数进行截帧配置，代表截帧的间隔。例如值设为5，则表示从第1帧开始截取，每隔5帧截取一帧，默认值5。	Integer	否
MaxFrames	针对 GIF 动图审核的最大截帧数量，需大于0。例如值设为5，则表示最大截取5帧，默认值为5。	Integer	否
LargeImageDetect	对于超过大小限制的图片是否进行压缩后再审核，取值为：0（不压缩），1（压缩）。默认为0。注：压缩最大支持32MB的图片，且会收取图片压缩费用。对于 GIF 等动态图过大时，压缩时间较长，可能会导致审核超时失败。	Integer	否
DataId	图片标识，该字段在结果中返回原始内容，长度限制为512字节。	String	否
Async	是否异步进行审核，取值 0：同步返回结果，1：异步进行审核，默认为0。	String	否
Callback	审核结果（Detail版本）以回调形式发送至您的回调地址，异步审核时生效，支持以 `http://` 或者 `https://` 开头的地址，例如：`http://www.callback.com`。	String	否

结果说明

调用 ImageRecognition 或 ImageAuditing 函数，会解析 api 返回的 xml 内容到 ImageRecognitionResult 结构，具体返回参数可查看 图片单次审核 文档。

图片批量审核

功能说明

图片批量审核接口为同步请求方式，您可以通过本接口对多个图片文件进行内容审核。该接口属于 POST 请求。

方法原型

```
// 图片批量审核
func (s *CIService) BatchImageAuditing(ctx context.Context, opt *BatchImageAuditingOptions) (*BatchImageAuditingJobResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
// CI 任务需要提供 CIURL
bu, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
cu, _ := url.Parse("https://examplebucket-1250000000.ci.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
},
})
opt := &cos.BatchImageAuditingOptions{
Input: []cos.ImageAuditingInputOptions{
cos.ImageAuditingInputOptions{
DataId: "1",
Object: "test.jpg",
},
cos.ImageAuditingInputOptions{
DataId: "2",
Url: "http://www.example.com/test.jpg",
},
},
Conf: &cos.ImageAuditingJobConf{
DetectType: "Porn,Ads",
},
}
res, _, err := c.CI.BatchImageAuditing(context.Background(), opt)
```

参数说明

```
type ImageAuditingInputOptions struct {
DataId string
```

```
Object string
Url string
Interval int
MaxFrames int
LargeImageDetect int
UserInfo *UserExtraInfo
}
type UserExtraInfo struct {
TokenId string
Nickname string
DeviceId string
AppId string
Room string
IP string
Type string
}
type ImageAuditingJobConf struct {
DetectType string
BizType string
Async int
Callback string
}
type BatchImageAuditingOptions struct {
Input []ImageAuditingInputOptions
Conf *ImageAuditingJobConf
}
```

参数名称	描述	类型	是否必选
DataId	图片标识，该字段在结果中返回原始内容，长度限制为512字节。	String	否
Object	存储在 COS 存储桶中的图片文件名称，例如在目录 test 中的文件 image.jpg，则文件名称为 test/image.jpg。Object 和 Url 只能选择其中一种。	String	否
Url	图片文件的链接地址，例如 http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/image.jpg。Object 和 Url 只能选择其中一种。	String	否
Interval	截帧频率，GIF 图检测专用，默认值为5，表示从第一帧（包含）开始每隔5帧截取一帧。	Integer	否
MaxFrames	最大截帧数量，GIF 图检测专用，默认值为5，表示只截取 GIF 的5帧图片进行审核，必须大于0。	Integer	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型	是否必选
BizType	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置公共审核策略。	String	否
DetectType	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以,分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。	String	否
Async	是否异步进行审核，取值 0：同步返回结果，1：异步进行审核，默认为0。	String	否
Callback	审核结果（Detail版本）以回调形式发送至您的回调地址，异步审核时生效，支持以 `http://` 或者 `https://` 开头的地址，例如：`http://www.callback.com`。	String	否
LargeImageDetect	对于超过大小限制的图片是否进行压缩后再审核，取值为：0（不压缩），1（压缩）。默认为0。注：压缩最大支持32MB的图片，且会收取图片压缩费用。对于 GIF 等动态图过大时，压缩时间较长，可能会导致审核超时失败。	Integer	否
UserInfo	用户业务字段。	Object	否

结果说明

调用 BatchImageAuditing 函数，会解析 api 返回的 xml 内容到 BatchImageAuditingJobResult 结构，具体返回参数可查看 图片批量审核 文档。

查询图片审核任务结果

功能说明

用于查询一个图片审核任务的结果详情。该接口属于 GET 请求。

方法原型

```
// 图片审核-查询任务
func (s *CIService) GetImageAuditingJob(ctx context.Context, jobid string) (*GetImageAuditingJobResult, *Response, error)
```

请求示例

```
jobId := "iace25f391a72e11eb99f********"
res, _, err := c.CI.GetImageAuditingJob(context.Background(), jobId)
```

参数说明

参数名称	参数描述	类型
jobId	任务 ID	String

结果说明

调用 GetImageAuditingJob 函数，会解析 api 返回的 xml 内容到 GetImageAuditingJobResult 结构，具体返回参数可查看 查询图片审核任务结果 文档。

视频审核

最近更新时间：2022-12-20 14:15:16

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于视频审核的相关的 API 概览以及 SDK 示例代码。

API	操作描述
创建任务	用于提交一个视频审核任务
查询任务	用于查询指定的视频审核任务

创建任务

PutVideoAuditingJob 接口用于提交一个视频审核任务。您可以通过主动设置回调地址接收审核信息，也可以通过 JobId 进行查询。

方法原型

```
func (s *CIService) PutVideoAuditingJob(ctx context.Context, opt *PutVideoAuditingJobOptions) (*PutVideoAuditingJobResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION修改为真实的信息
// CI 任务需要提供CIURL
```

```
bu, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com"
)
cu, _ := url.Parse("https://examplebucket-1250000000.ci.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
}
})
opt := &cos.PutVideoAuditingJobOptions{
InputObject: "demo.mp4",
Conf: &cos.VideoAuditingJobConf{
DetectType: "Porn,Ads",
Snapshot: &cos.PutVideoAuditingJobSnapshot{
Mode: "Interval",
TimeInterval: 50.5,
Count: 100,
},
},
}
res, _, err := c.CI.PutVideoAuditingJob(context.Background(), opt)
```

参数说明

```
type PutVideoAuditingJobOptions struct {
InputObject string
InputUrl string
InputDataId string
InputUserInfo *UserExtraInfo
Conf *VideoAuditingJobConf
Type string
}
type UserExtraInfo struct {
TokenId string
Nickname string
DeviceId string
AppId string
Room string
IP string
Type string
}
type VideoAuditingJobConf struct {
DetectType string
Snapshot *PutVideoAuditingJobSnapshot
Callback string
```

版权所有：腾讯云计算（北京）有限责任公司

```
CallbackVersion string
CallbackType int
BizType string
DetectContent int
}
type PutVideoAuditingJobSnapshot struct {
Mode string
Count int
TimeInterval float32
}
```

参数名称	参数描述	类型
InputObject	当前 COS 存储桶中的视频文件名称，例如在目录 test 中的文件 video.mp4，则文件名称为 `test/video.mp4` 。Object 和 Url 只能选择其中一种。	String
InputUrl	视频文件的链接地址，例如 `http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/test.mp4` 。Object 和 Url 只能选择其中一种。	String
InputDataId	视频标识，该字段在结果中返回原始内容，长度限制为512字节。	String
InputUserInfo	用户业务字段。	Object
Type	审核的任务类型，直播流审核固定为 live_video。	String
Conf	审核规则配置。	Struct
BizType	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置审核策略。	String
DetectType	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。	String
Snapshot	视频画面的审核通过视频截帧能力截取出一定量的截图，通过对截图逐一审核而实现的，该参数用于指定视频截帧的配置。	Struct
Callback	审核结果以回调形式发送至您的回调地址，支持以 `http://` 或者 `https://` 开头的地址，例如：`http://www.callback.com` 。	String
CallbackVersion	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。	String
CallbackType	回调片段类型，有效值：1（回调全部截帧和音频片段）、2（仅回调违规截帧和音频片段）。默认为 1。	Integer

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
DetectContent	用于指定是否审核视频声音，当值为0时：表示只审核视频画面截图；值为1时：表示同时审核视频画面截图和视频声音。默认值为0。	Integer
Mode	截帧模式，默认值为Interval。Interval 表示间隔模式；Average 表示平均模式；Fps 表示固定帧率模式。Interval 模式：TimeInterval，Count 参数生效。当设置 Count，未设置 TimeInterval 时，表示截取所有帧，共 Count 张图片。Average 模式：Count 参数生效。表示整个视频，按平均间隔截取共 Count 张图片。Fps 模式：TimeInterval 表示每秒截取多少帧，未设置 TimeInterval 时，表示截取所有帧，Count 表示共截取多少帧。	String
Count	视频截帧数量，范围为(0, 10000]。	Integer
TimeInterval	视频截帧频率，范围为(0, 60]，单位为秒，支持 float 格式，执行精度精确到毫秒。	Float

结果说明

调用 PutVideoAuditingJob 函数，会解析 api 返回的 xml 内容到 PutVideoAuditingJobResult 结构，具体返回参数可查看 提交视频审核任务 文档。

查询任务

GetVideoAuditingJob 接口用来查询指定的视频审核任务。您可以根据视频审核任务的 JobId 来查询视频审核结果。

方法原型

```
func (s *CIService) GetVideoAuditingJob(ctx context.Context, jobid string) (*GetVideoAuditingJobResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION修改为真实的信息
// CI 任务需要提供CIURL
bu, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
cu, _ := url.Parse("https://examplebucket-1250000000.ci.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
```

```
}
})
jobId := "avce25f391a72e11eb99f********"
res, _, err := c.CI.GetVideoAuditingJob(context.Background(), jobId)
```

参数说明

参数名称	参数描述	类型
jobId	任务 ID	String

结果说明

调用 GetVideoAuditingJob 函数，会解析 api 返回的 xml 内容到 GetVideoAuditingJobResult 结构，具体返回参数可查看 查询视频审核任务结果 文档。

音频审核

最近更新时间：2022-12-20 14:15:16

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于音频审核的相关的 API 概览以及 SDK 示例代码。

API	操作描述
创建任务	用于提交一个音频审核任务
查询任务	用于查询指定的音频审核任务

创建任务

PutAudioAuditingJob 接口用来提交一个音频审核任务。您可以通过主动设置回调地址接收审核信息，也可以通过 JobId 进行查询。

方法原型

```
func (s *CIService) PutAudioAuditingJob(ctx context.Context, opt *PutAudioAuditingJobOptions) (*PutAudioAuditingJobResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION修改为真实的信息
// CI 任务需要提供CIURL
```

```
bu, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com"
)
cu, _ := url.Parse("https://examplebucket-1250000000.ci.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
}
})
opt := &cos.PutAudioAuditingJobOptions{
InputObject: "test.mp3",
Conf: &cos.AudioAuditingJobConf{
DetectType: "Porn,Ads",
},
}
res, _, err := c.CI.PutAudioAuditingJob(context.Background(), opt)
```

参数说明

```
type PutAudioAuditingJobOptions struct {
InputObject string
InputUrl string
InputDataId string
InputUserInfo *UserExtraInfo
Conf *AudioAuditingJobConf
}
type UserExtraInfo struct {
TokenId string
Nickname string
DeviceId string
AppId string
Room string
IP string
Type string
}
type AudioAuditingJobConf struct {
DetectType string
Callback string
CallbackVersion string
BizType string
}
```

参数名称	参数描述	类型

参数名称	参数描述	类型
InputObject	存储在 COS 存储桶中的音频文件名称，例如在目录 test 中的文件 audio.mp3，则文件名称为 `test/audio.mp3` 。Object 和 Url 只能选择其中一种。	String
InputUrl	音频文件的链接地址，例如 `http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/audio.mp3` 。Object 和 Url 只能选择其中一种。	String
InputDataId	音频标识，该字段在结果中返回原始内容，长度限制为512字节。	String
InputUserInfo	用户业务字段。	Object
Conf	审核规则配置。	Struct
BizType	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置审核策略。	String
DetectType	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。	String
Callback	审核结果以回调形式发送至您的回调地址，支持以 `http://` 或者 `https://` 开头的地址，例如： `http://www.callback.com` 。	String
CallbackVersion	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。	String

结果说明

调用 PutAudioAuditingJob 函数，会解析 api 返回的 xml 内容到 PutAudioAuditingJobResult 结构，具体返回参数可查看 提交音频审核任务 文档。

查询任务

GetAudioAuditingJob 接口用来查询指定的音频审核任务。您可以根据音频审核任务的 JobId 来查询音频审核结果。

方法原型

```
func (s *CIService) GetAudioAuditingJob(ctx context.Context, jobid string) (*GetAudioAuditingJobResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION修改为真实的信息
// CI 任务需要提供CIURL
bu, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com"
)
cu, _ := url.Parse("https://examplebucket-1250000000.ci.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
}
})
jobId := "sace25f391a72e11eb99f********"
res, _, err := c.CI.GetAudioAuditingJob(context.Background(), jobId)
```

参数说明

参数名称	参数描述	类型
jobId	任务 ID。	String

结果说明

调用 GetAudioAuditingJob 函数，会解析 api 返回的 xml 内容到 GetAudioAuditingJobResult 结构，具体返回参数可查看 查询音频审核任务结果 文档。

文本审核

最近更新时间：2022-12-20 14:15:16

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于文本审核的相关的 API 概览以及 SDK 示例代码。

API	操作描述
创建任务	用于提交一个文本审核任务
查询任务	用于查询指定的文本审核任务

创建任务

PutTextAuditingJob 接口用来提交一个文本审核任务。当审核文本内容时，响应体直接返回审核结果，当审核文本对象时，返回任务摘要，您可以通过主动设置回调地址接收审核信息，也可以通过 JobId 进行查询。

方法原型

```
func (s *CIService) PutTextAuditingJob(ctx context.Context, opt *PutTextAuditingJobOptions) (*PutTextAuditingJobResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION修改为真实的信息
// CI 任务需要提供CIURL
```

```
bu, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com"
)
cu, _ := url.Parse("https://examplebucket-1250000000.ci.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
}
})
opt := &cos.PutTextAuditingJobOptions{
InputObject: "test.txt",
Conf: &cos.TextAuditingJobConf{
DetectType: "Porn,Ads",
},
}
res, _, err := c.CI.PutTextAuditingJob(context.Background(), opt)
```

参数说明

```
type PutTextAuditingJobOptions struct {
InputObject string
InputUrl string
InputContent string
InputDataId string
InputUserInfo *UserExtraInfo
Conf *TextAuditingJobConf
}
type UserExtraInfo struct {
TokenId string
Nickname string
DeviceId string
AppId string
Room string
IP string
Type string
}
type TextAuditingJobConf struct {
DetectType string
Callback string
CallbackVersion string
BizType string
}
```

参数名称	参数描述	类型

参数名称	参数描述	类型
InputObject	当前 COS 存储桶中的文本文件名称，例如在目录 test 中的文件 test.txt，则文件名称为 `test/test.txt`，文本文件仅支持UTF8编码和 GBK 编码的内容，且文件大小不得超过1MB。Object， Url 和 Content 只能选择其中一种。	String
InputContent	当传入的内容为纯文本信息，需要先经过 base64 编码，文本编码前的原文长度不能超过10000个 utf8 编码字符。若超出长度限制，接口将会报错。Object，Url 和 Content 只能选择其中一种。	String
InputUrl	文本文件的链接地址，例如 `http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/test.txt`。Object，Url 和 Content 只能选择其中一种。	String
InputDataId	文本标识，该字段在结果中返回原始内容，长度限制为512字节。	String
InputUserInfo	用户业务字段。	Object
Conf	审核规则配置。	Struct
BizType	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置审核策略。	String
DetectType	审核的场景类型，有效值：Porn（涉黄）、Ads（广告）、Illegal（违法）、Abuse（谩骂），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。	String
Callback	审核结果可以回调形式发送至您的回调地址，支持以 `http://` 或者 `https://` 开头的地址，例如：`http://www.callback.com`。当Input使用Content时，该参数不生效，结果会直接返回。	String
CallbackVersion	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为Simple。	String

结果说明

调用 PutTextAuditingJob 函数，会解析 api 返回的 xml 内容到 PutTextAuditingJobResult 结构，具体返回参数可查看 提交文本审核任务 文档。

查询任务

GetTextAuditingJob 接口用来查询指定的文本审核任务。您可以根据文本审核任务的 JobId 来查询文本审核结果。

方法原型

```
func (s *CIService) GetTextAuditingJob(ctx context.Context, jobid string) (*GetTe
xtAuditingJobResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION修改为真实的信息
// CI 任务需要提供CIURL
bu, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com"
)
cu, _ := url.Parse("https://examplebucket-1250000000.ci.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
}
})
jobId := "stce25f391a72e11eb99f********"
res, _, err := c.CI.GetTextAuditingJob(context.Background(), jobId)
```

参数说明

参数名称	参数描述	类型
jobId	任务 ID。	String

结果说明

调用 GetTextAuditingJob 函数，会解析 api 返回的 xml 内容到 GetTextAuditingJobResult 结构，具体返回参数可查看 查询文本审核任务结果 文档。

文档审核

最近更新时间：2022-12-20 14:15:16

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于文档审核的相关的 API 概览以及 SDK 示例代码。

API	操作描述
创建任务	用于提交一个文档审核任务
查询任务	用于查询指定的文档审核任务

创建任务

PutDocumentAuditingJob 接口用来提交一个文档审核任务。您可以通过主动设置回调地址接收审核信息，也可以通过 JobId 进行查询。

方法原型

```
func (s *CIService) PutDocumentAuditingJob(ctx context.Context, opt *PutDocumentAuditingJobOptions) (*PutDocumentAuditingJobResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION修改为真实的信息
// CI 任务需要提供CIURL
```

```
bu, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com"
)
cu, _ := url.Parse("https://examplebucket-1250000000.ci.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
}
})
opt := &cos.PutDocumentAuditingJobOptions{
InputUrl: "http://www.example.com/doctest.docx",
InputType: "docx",
Conf: &cos.DocumentAuditingJobConf{
DetectType: "Porn,Ads",
},
}
res, _, err := c.CI.PutDocumentAuditingJob(context.Background(), opt)
```

参数说明

```
type PutDocumentAuditingJobOptions struct {
InputObject string
InputUrl string
InputType string
InputDataId string
InputUserInfo *UserExtraInfo
Conf *DocumentAuditingJobConf
}
type UserExtraInfo struct {
TokenId string
Nickname string
DeviceId string
AppId string
Room string
IP string
Type string
}
type DocumentAuditingJobConf struct {
DetectType string
Callback string
BizType string
}
```

参数名称	参数描述	类型

参数名称	参数描述	类型
InputObject	存储在 COS 存储桶中的文件名称，例如在目录 test 中的文件test.doc，则文件名称为 `test/test.doc` 。Object 和 Url 只能选择其中一种。	String
InputUrl	文档文件的链接地址，例如 `http://www.example.com/doctest.doc` ，Object 和 Url 只能选择其中一种。	String
InputType	指定文档文件的类型，如未指定则默认以文件的后缀为类型。如果文件没有后缀，该字段必须指定，否则会审核失败。例如：doc、docx、ppt、pptx 等。	String
InputDataId	文档标识，该字段在结果中返回原始内容，长度限制为512字节。	String
InputUserInfo	用户业务字段。	Object
Conf	审核规则配置。	Struct
BizType	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置审核策略。	String
DetectType	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。	String
Callback	审核结果可以回调形式发送至您的回调地址，支持以 `http://` 或者 `https://` 开头的地址，例如： `http://www.callback.com` 。	String

结果说明

调用 PutDocumentAuditingJob 函数，会解析 api 返回的 xml 内容到 PutDocumentAuditingJobResult 结构，具体返回参数可查看 提交文档审核任务 文档。

查询任务

GetDocumentAuditingJob 接口用来查询指定的文档审核任务。您可以根据文档审核任务的 JobId 来查询文档审核结果。

方法原型

```
func (s *CIService) GetDocumentAuditingJob(ctx context.Context, jobid string) (*GetDocumentAuditingJobResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION修改为真实的信息
// CI 任务需要提供CIURL
bu, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com"
)
cu, _ := url.Parse("https://examplebucket-1250000000.ci.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
}
})
jobId := "sdce25f391a72e11eb99f********"
res, _, err := c.CI.GetDocumentAuditingJob(context.Background(), jobId)
```

参数说明

参数名称	参数描述	类型
jobId	任务 ID。	String

结果说明

调用 GetDocumentAuditingJob 函数，会解析 api 返回的 xml 内容到 GetDocumentAuditingJobResult 结构，具体返回参数可查看 查询文档审核任务结果 文档。

网页审核

最近更新时间：2023-03-16 15:16:52

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于网页审核的 API 概览和 SDK 示例代码。

API	操作描述
创建任务	用于提交一个网页审核任务
查询任务	用于查询指定的网页审核任务

创建任务

PutWebpageAuditingJob 接口用于提交一个网页审核任务。您可以通过主动设置回调地址接收审核信息，也可以通过 JobId 进行查询。

方法原型

```
func (s *CIService) PutWebpageAuditingJob(ctx context.Context, opt *PutWebpageAuditingJobOptions) (*PutWebpageAuditingJobResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
// CI 任务需要提供CIURL
```

```
bu, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com"
)
cu, _ := url.Parse("https://examplebucket-1250000000.ci.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
SecretKey: os.Getenv("SECRETKEY"),
}
})
opt := &cos.PutWebpageAuditingJobOptions{
InputUrl: "http://www.test.com",
Conf: &cos.WebpageAuditingJobConf{
DetectType: "Porn,Ads",
},
}
res, _, err := c.CI.PutWebpageAuditingJob(context.Background(), opt)
```

参数说明

```
type PutWebpageAuditingJobOptions struct {
InputUrl string
InputDataId string
InputUserInfo *UserExtraInfo
Conf *WebpageAuditingJobConf
}
type UserExtraInfo struct {
TokenId string
Nickname string
DeviceId string
AppId string
Room string
IP string
Type string
}
type WebpageAuditingJobConf struct {
DetectType string
Callback string
ReturnHighlightHtml bool
}
```

参数名称	参数描述	类型
InputUrl	网页文件的链接地址，例如 `http://www.test.com` 。	String

参数名称	参数描述	类型
InputDataId	网页标识，该字段在结果中返回原始内容，长度限制为512字节。	String
InputUserInfo	用户业务字段。	Object
Conf	审核规则配置。	Struct
DetectType	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。	String
Callback	回调地址，以 `http://` 或者 `https://` 开头的地址。	String
ReturnHighlightHtml	指定是否需要高亮展示网页内的违规文本，并返回高亮展示的 html 链接。取值为 true 和 false，默认为 false。	Bool

结果说明

调用 PutWebpageAuditingJob 函数，会解析 API 返回的 xml 内容到 PutWebpageAuditingJobResult 结构，具体返回参数可查看 提交网页审核任务 文档。

查询任务

GetWebpageAuditingJob 接口用于查询指定的网页审核任务。您可以根据网页审核任务的 JobId 来查询网页审核结果。

方法原型

```
func (s *CIService) GetWebpageAuditingJob(ctx context.Context, jobid string) (*GetWebpageAuditingJobResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
// CI 任务需要提供 CIURL
bu, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
cu, _ := url.Parse("https://examplebucket-1250000000.ci.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu, CIURL: cu}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: os.Getenv("SECRETID"),
```

```
SecretKey: os.Getenv("SECRETKEY"),
}
})
jobId := "shce25f391a72e11eb99f********"
res, _, err := c.CI.GetWebpageAuditingJob(context.Background(), jobId)
```

参数说明

参数名称	参数描述	类型
jobId	任务 ID。	String

结果说明

调用 GetWebpageAuditingJob 函数，会解析 API 返回的 xml 内容到 GetWebpageAuditingJobResult 结构，具体返回参数可查看 查询网页审核任务结果 文档。

iOS SDK
快速入门

最近更新时间：2023-01-30 11:50:32

相关资源

- SDK 源码地址请参考： XML iOS SDK。
- 示例 Demo 可参考：XML iOS SDK Demo。
- SDK 接口与参数文档请参见 SDK API 参考。
- SDK 文档中的所有示例代码请参见 SDK 代码示例。
- SDK 更新日志请参见：ChangeLog。
- SDK 常见问题请参见：iOS SDK 常见问题。

> 说明：
> 如果您在使用 XML 版本 SDK 时遇到函数或方法不存在等错误，请先将 XML 版本 SDK 升级到最新版再重试。

准备工作

1. 您需要一个 iOS 应用，这个应用可以是您现有的工程，也可以是您新建的一个空的工程。
2. 请确保应用基于 iOS 8.0 及以上版本的 SDK 构建。
3. 您需要一个可以获取腾讯云临时密钥的远程地址，关于临时密钥的有关说明请参考 移动应用直传实践。

第一步：安装 SDK

方式一：使用 Cocoapods 集成（推荐）

标准版 SDK

在您工程的 `Podfile` 文件中使用：

```
pod 'QCloudCOSXML'
```

精简版 SDK

如果您仅仅使用到上传和下载功能，并且对 SDK 体积要求较高，可以使用我们的精简版 SDK。

精简版 SDK 是通过 Cocoapods 的 Subspec 功能实现的，因此目前只支持通过自动集成的方式。在您工程的 `Podfile` 文件中使用：

```
pod 'QCloudCOSXML/Transfer'
```

方式二：手动集成

您可以通过 快速下载地址 下载最新的正式包，也可以在 SDK Releases 里面找到我们所有历史版本的正式包。

1. 导入二进制库

将 **QCloudCOSXML.framework、QCloudCore.framework 和 BeaconAPI_Base.framework 以及 QimeiSDK.framework** 拖入到工程中。



并添加以下依赖库：

- CoreTelephony
- Foundation
- SystemConfiguration
- libc++.tbd

2. 工程配置

在 Build Settings 中设置 Other Linker Flags，加入以下参数：

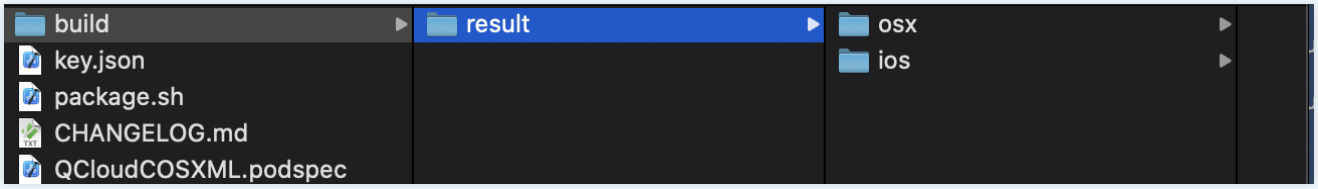
```
-ObjC
-all_load
```



说明：
SDK 提供了打包脚本，支持根据业务需求自行打包（该打包脚本依赖于 cocoapods，请先确保您的开发环境安装了 Cocoapods），打包步骤如下：

1. 下载源码：`git clone https://github.com/tencentyun/qcloud-sdk-ios` 。
2. 运行打包脚本：`source package.sh` 。
3. 将打包产物拖到工程中，然后按照上面手动集成的方式操作即可。

- iOS：将 ios 文件夹下的产物拖入项目。
- macOS：将 osx 文件夹下的产物拖入项目。



第二步：开始使用

1. 导入头文件

Objective-c

```
#import <QCloudCOSXML/QCloudCOSXML.h>
```

swift

```
import QCloudCOSXML
```

对于精简版 SDK，请导入：

Objective-c

```
#import <QCloudCOSXML/QCloudCOSXMLTransfer.h>
```

swift

```
import QCloudCOSXMLTransfer
```

2. 初始化 COS 服务并实现签名协议

注意：

- 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

方式一：获取临时密钥对请求授权（推荐）

SDK 在发出请求时，需要获取临时密钥计算签名，因此需要您实现 `QCloudSignatureProvider` 协议，在该协议中获取密钥后将密钥通过参数 `continueBlock` 回调给 SDK。

建议把初始化过程放在 `AppDelegate` 或者**程序单例**中。

具体步骤请参考下面的完整示例代码。

Objective-c

```
//AppDelegate.m
//AppDelegate 需遵循 QCloudSignatureProvider
@interface AppDelegate()<QCloudSignatureProvider>
@end
@implementation AppDelegate
- (BOOL)application:(UIApplication * )application
didFinishLaunchingWithOptions:(NSDictionary * )launchOptions {
QCloudServiceConfiguration* configuration = [QCloudServiceConfiguration new];
QCloudCOSXMLEndPoint* endpoint = [[QCloudCOSXMLEndPoint alloc] init];

// 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.cloud.tencent.com/cos5/bucket
// COS支持的所有region列表参见https://www.qcloud.com/document/product/436/6224
endpoint.regionName = @"COS_REGION";
// 使用 HTTPS
endpoint.useHTTPS = true;
configuration.endpoint = endpoint;
// 密钥提供者为自己
```

```
configuration.signatureProvider = self;
// 初始化 COS 服务示例
[QCloudCOSXMLService registerDefaultCOSXMLWithConfiguration:configuration];
[QCloudCOSTransferMangerService registerDefaultCOSTransferMangerWithConfiguratio
n:
configuration];
return YES;
}
// 获取签名的方法入口，这里演示了获取临时密钥并计算签名的过程
// 您也可以自定义计算签名的过程
- (void) signatureWithFields:(QCloudSignatureFields*)fileds
request:(QCloudBizHTTPRequest*)request
urlRequest:(NSMutableURLRequest*)urlRequst
compelete:(QCloudHTTPAuthentationContinueBlock)continueBlock
{
//这里同步从后台服务器获取临时密钥，强烈建议将获取临时密钥的逻辑放在这里，最大程度上保证密钥的可
用性
//...
QCloudCredential* credential = [QCloudCredential new];
// 临时密钥 SecretId
// sercret_id替换为用户的 SecretId，登录访问管理控制台查看密钥，https://console.cloud.te
ncent.com/cam/capi
credential.secretID = @"SECRETID";
// 临时密钥 SecretKey
// sercret_key替换为用户的 SecretKey，登录访问管理控制台查看密钥，https://console.cloud.
tencent.com/cam/capi
credential.secretKey = @"SECRETKEY";
// 临时密钥 Token
// 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https:
//intl.cloud.tencent.com/document/product/436/14048
credential.token = @"TOKEN";
/** 强烈建议返回服务器时间作为签名的开始时间，用来避免由于用户手机本地时间偏差过大导致的签名不正
确(参数startTime和expiredTime单位为秒)
*/
credential.startDate = [NSDate dateWithTimeIntervalSince1970:startTime]; // 单位是
秒
credential.expirationDate = [NSDate dateWithTimeIntervalSince1970:expiredTime]];
// 单位是秒
QCloudAuthentationV5Creator* creator = [[QCloudAuthentationV5Creator alloc]
initWithCredential:credential];
// 注意 这里不要对urlRequst 进行copy以及mutableCopy操作
QCloudSignature *signature = [creator signatureForData:urlRequst];
continueBlock(signature, nil);
}
@end
```

Swift

```
//AppDelegate.swift
//AppDelegate 需遵循 QCloudSignatureProvider
class AppDelegate: UIResponder, UIApplicationDelegate,
QCloudSignatureProvider {
func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
let config = QCloudServiceConfiguration.init();
let endpoint = QCloudCOSXMLEndPoint.init();
// 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.clou
d.tencent.com/cos5/bucket
// COS支持的所有region列表参见https://www.qcloud.com/document/product/436/6224
endpoint.regionName = "COS_REGION";
// 使用 HTTPS
endpoint.useHTTPS = true;
config.endpoint = endpoint;
// 密钥提供者为自己
config.signatureProvider = self;
// 初始化 COS 服务示例
QCloudCOSXMLService.registerDefaultCOSXML(with: config);
QCloudCOSTransferMangerService.registerDefaultCOSTransferManger(
with: config);
return true
}
// 获取签名的方法入口，这里演示了获取临时密钥并计算签名的过程
// 您也可以自定义计算签名的过程
func signature(with fileds: QCloudSignatureFields!,
request: QCloudBizHTTPRequest!,
urlRequest urlRequst: NSMutableURLRequest!,
compelete continueBlock: QCloudHTTPAuthentationContinueBlock!) {

//这里同步从后台服务器获取临时密钥
//...
let credential = QCloudCredential.init();
// 临时密钥 SecretId
// sercret_id替换为用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.clo
ud.tencent.com/cam/capi
credential.secretID = "SECRETID";
// 临时密钥 SecretKey
// sercret_key替换为用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.c
loud.tencent.com/cam/capi
credential.secretKey = "SECRETKEY";
// 临时密钥 Token
// 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见http
s://intl.cloud.tencent.com/document/product/436/14048
credential.token = "TOKEN";
```

```
/** 强烈建议返回服务器时间作为签名的开始时间，用来避免由于用户手机本地时间偏差过大导致的签名不正确(参数startTime和expiredTime单位为秒)
*/
credential.startDate = Date.init(timeIntervalSince1970: TimeInterval(startTime)!)
DateFormatter().date(from: "startTime");
credential.expirationDate = Date.init(timeIntervalSince1970: TimeInterval(expired
Time)!)
let creator = QCloudAuthentationV5Creator.init(credential: credential);
// 注意 这里不要对urlRequst 进行copy以及mutableCopy操作
let signature = creator?.signature(forData: urlRequst);
continueBlock(signature,nil);

}
}
```

注意：

- APPID 是您在成功申请腾讯云账户后所得到的账号，由系统自动分配，具有固定性和唯一性，可在 账号信息 中查看。腾讯云账号的 APPID，是与账号 ID 有唯一对应关系的应用 ID。
- SecretId 和 SecretKey 合称为云 API 密钥，是用户访问腾讯云 API 进行身份验证时需要用到的安全凭证，可在 API 密钥管理 中获取。SecretKey 是用于加密签名字符串和服务器端验证签名字符串的密钥。SecretId 用于标识 API 调用者身份。一个 APPID 可以创建多个云 API 密钥。

SDK 提供了一个 `QCloudCredentailFenceQueue` 的脚手架，实现对临时密钥的缓存与复用。脚手架在密钥过期之后会重新调用该协议的方法来重新获取新的密钥，直到该密钥过期时间大于设备的当前时间。

注意：
脚手架只是根据密钥的过期时间是否大于设备的当前时间来判断密钥是否能复用，如果您申请密钥时设置了复杂的策略，则不建议使用脚手架工具。

建议把初始化过程放在 `AppDelegate` 或者**程序单例**中。使用脚手架您需要实现以下两个协议：

- QCloudSignatureProvider
- QCloudCredentailFenceQueueDelegate

完整示例代码请参考如下：

Objective-c

```
//AppDelegate.m
//AppDelegate 需遵循 QCloudSignatureProvider 与
//QCloudCredentailFenceQueueDelegate 协议
@interface AppDelegate()<QCloudSignatureProvider, QCloudCredentailFenceQueueDeleg
ate>
// 一个脚手架实例
@property (nonatomic) QCloudCredentailFenceQueue* credentialFenceQueue;
@end
@implementation AppDelegate
- (BOOL)application:(UIApplication * )application
didFinishLaunchingWithOptions:(NSDictionary * )launchOptions {
QCloudServiceConfiguration* configuration = [QCloudServiceConfiguration new];
QCloudCOSXMLEndPoint* endpoint = [[QCloudCOSXMLEndPoint alloc] init];
// 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.cloud.tenc
ent.com/cos5/bucket
// COS支持的所有region列表参见https://www.qcloud.com/document/product/436/6224
endpoint.regionName = @"COS_REGION";
// 使用 HTTPS
endpoint.useHTTPS = true;
configuration.endpoint = endpoint;
// 密钥提供者为自己
configuration.signatureProvider = self;
// 初始化 COS 服务示例
[QCloudCOSXMLService registerDefaultCOSXMLWithConfiguration:configuration];
[QCloudCOSTransferMangerService registerDefaultCOSTransferMangerWithConfiguratio
n:
configuration];
// 初始化临时密钥脚手架
self.credentialFenceQueue = [QCloudCredentailFenceQueue new];
self.credentialFenceQueue.delegate = self;
return YES;
}
- (void) fenceQueue:(QCloudCredentailFenceQueue * )queue requestCreatorWithContin
ue:(QCloudCredentailFenceQueueContinue)continueBlock
{
//这里同步从◊后台服务器获取临时密钥，强烈建议将获取临时密钥的逻辑放在这里，最大程度上保证密钥的
可用性
//...
QCloudCredential* credential = [QCloudCredential new];
// 临时密钥 SecretId
// sercret_id替换为用户的 SecretId，登录访问管理控制台查看密钥，https://console.cloud.te
ncent.com/cam/capi
credential.secretID = @"SECRETID";
// 临时密钥 SecretKey
// sercret_key替换为用户的 SecretKey，登录访问管理控制台查看密钥，https://console.cloud.
tencent.com/cam/capi
```

```
credential.secretKey = @"SECRETKEY";
// 临时密钥 Token
// 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
credential.token = @"TOKEN";
/** 强烈建议返回服务器时间作为签名的开始时间，用来避免由于用户手机本地时间偏差过大导致的签名不正确(参数startTime和expiredTime单位为秒)
*/
credential.startDate = [NSDate dateWithTimeIntervalSince1970:startTime]; // 单位是秒
credential.expirationDate = [NSDate dateWithTimeIntervalSince1970:expiredTime]];
// 单位是秒
QCloudAuthentationV5Creator* creator = [[QCloudAuthentationV5Creator alloc]
initWithCredential:credential];
continueBlock(creator, nil);
}
// 获取签名的方法入口，这里演示了获取临时密钥并计算签名的过程
// 您也可以自定义计算签名的过程
- (void) signatureWithFields:(QCloudSignatureFields*)fileds
request:(QCloudBizHTTPRequest*)request
urlRequest:(NSMutableURLRequest*)urlRequst
compelete:(QCloudHTTPAuthentationContinueBlock)continueBlock
{
[self.credentialFenceQueue performAction:^(QCloudAuthentationCreator *creator,
NSError *error) {
if (error) {
continueBlock(nil, error);
} else {
// 注意 这里不要对urlRequst 进行copy以及mutableCopy操作
QCloudSignature* signature = [creator signatureForData:urlRequst];
continueBlock(signature, nil);
}
}];
}
@end
```

Swift

```
//AppDelegate.swift
//AppDelegate 需遵循 QCloudSignatureProvider 与
//QCloudCredentailFenceQueueDelegate 协议
class AppDelegate: UIResponder, UIApplicationDelegate,
QCloudSignatureProvider, QCloudCredentailFenceQueueDelegate {
var credentialFenceQueue:QCloudCredentailFenceQueue?;
func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
```

```
let config = QCloudServiceConfiguration.init();
let endpoint = QCloudCOSXMLEndPoint.init();
// 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.clou
d.tencent.com/cos5/bucket
// COS支持的所有region列表参见https://www.qcloud.com/document/product/436/6224
endpoint.regionName = "COS_REGION";
// 使用 HTTPS
endpoint.useHTTPS = true;
config.endpoint = endpoint;
// 密钥提供者为自己
config.signatureProvider = self;
// 初始化 COS 服务示例
QCloudCOSXMLService.registerDefaultCOSXML(with: config);
QCloudCOSTransferMangerService.registerDefaultCOSTransferManger(
with: config);
// 初始化临时密钥脚手架
self.credentialFenceQueue = QCloudCredentailFenceQueue.init();
self.credentialFenceQueue?.delegate = self;
return true
}
func fenceQueue(_ queue: QCloudCredentailFenceQueue!,
requestCreatorWithContinue continueBlock:
QCloudCredentailFenceQueueContinue!) {
//这里同步从后台服务器获取临时密钥
//...
let credential = QCloudCredential.init();
// 临时密钥 SecretId
// sercret_id替换为用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.clo
ud.tencent.com/cam/capi
credential.secretID = "SECRETID";
// 临时密钥 SecretKey
// sercret_key替换为用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.c
loud.tencent.com/cam/capi
credential.secretKey = "SECRETKEY";
// 临时密钥 Token
// 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见http
s://intl.cloud.tencent.com/document/product/436/14048
credential.token = "TOKEN";
/** 强烈建议返回服务器时间作为签名的开始时间，用来避免由于用户手机本地时间偏差过大导致的签名不正
确(参数startTime和expiredTime单位为秒)
*/
credential.startDate = Date.init(timeIntervalSince1970: TimeInterval(startTime)!)
DateFormatter().date(from: "startTime");
credential.expirationDate = Date.init(timeIntervalSince1970: TimeInterval(expired
Time)!)
let auth = QCloudAuthentationV5Creator.init(credential: credential);
continueBlock(auth,nil);
```

```
}
// 获取签名的方法入口，这里演示了获取临时密钥并计算签名的过程
// 您也可以自定义计算签名的过程
func signature(with fileds: QCloudSignatureFields!,
request: QCloudBizHTTPRequest!,
urlRequest urlRequst: NSMutableURLRequest!,
compelete continueBlock: QCloudHTTPAuthentationContinueBlock!) {
self.credentialFenceQueue?.performAction({ (creator, error) in
if error != nil {
continueBlock(nil,error!);
}else{
// 注意 这里不要对urlRequst 进行copy以及mutableCopy操作
let signature = creator?.signature(forData: urlRequst);
continueBlock(signature,nil);
}
})
}
}
```

注意：

- 关于存储桶不同地域的简称请参考 地域和访问域名。
- 建议您使用 HTTPS 来请求数据，但如果您希望使用 HTTP 协议，为了确保在 iOS 9.0 以上的系统上可以运行，您需要为应用开启允许通过 HTTP 传输。详细的指引请参考 Apple 官方的说明文档 Preventing Insecure Network Connections。

如果您的 `QCloudServiceConfiguration` 发生改变，可以通过以下方法注册一个新的实例：

```
+ (QCloudCOSTransferMangerService*) registerCOSTransferMangerWithConfiguration:(QCloudServiceConfig
```

方式二：使用永久密钥进行本地调试

您可以使用腾讯云的永久密钥来进行开发阶段的本地调试。**由于该方式存在泄漏密钥的风险，请务必在上线前替换为临时密钥的方式。**

在使用永久密钥时，您可以不实现 `QCloudCredentailFenceQueueDelegate` 协议。

Objective-C

```
- (void) signatureWithFields:(QCloudSignatureFields*)fileds
request:(QCloudBizHTTPRequest*)request
urlRequest:(NSMutableURLRequest*)urlRequst
compelete:(QCloudHTTPAuthentationContinueBlock)continueBlock
{

QCloudCredential* credential = [QCloudCredential new];

// 永久密钥 secretID
// sercret_id替换为用户的 SecretId，登录访问管理控制台查看密钥，https://console.cloud.tencent.com/cam/capi
credential.secretID = @"SECRETID";
// 永久密钥 SecretKey
// sercret_key替换为用户的 SecretKey，登录访问管理控制台查看密钥，https://console.cloud.tencent.com/cam/capi
credential.secretKey = @"SECRETKEY";
// 使用永久密钥计算签名
QCloudAuthentationV5Creator* creator = [[QCloudAuthentationV5Creator alloc]
initWithCredential:credential];
// 注意 这里不要对urlRequst 进行copy以及mutableCopy操作
QCloudSignature* signature = [creator signatureForData:urlRequst];
continueBlock(signature, nil);
}
```

Swift

```
func signature(with fileds: QCloudSignatureFields!,
request: QCloudBizHTTPRequest!,
urlRequest urlRequst: NSMutableURLRequest!,
compelete continueBlock: QCloudHTTPAuthentationContinueBlock!) {
let credential = QCloudCredential.init();

// 永久密钥 secretID
// sercret_id替换为用户的 SecretId，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
credential.secretID = "SECRETID";
// 永久密钥 SecretKey
// sercret_key替换为用户的 SecretKey，登录访问管理控制台查看密钥，https://console.intl.cloud.tencent.com/cam/capi
credential.secretKey = "SECRETKEY";
// 使用永久密钥计算签名
let auth = QCloudAuthentationV5Creator.init(credential: credential);
// 注意 这里不要对urlRequst 进行copy以及mutableCopy操作
let signature = auth?.signature(forData: urlRequst)
continueBlock(signature,nil);
}
```

方式三：使用后台计算的签名对请求授权

在签名过程放在后台时，您可以不实现 `QCloudCredentailFenceQueueDelegate` 协议。

Objective-C

```
- (void) signatureWithFields:(QCloudSignatureFields*)fileds
request:(QCloudBizHTTPRequest*)request
urlRequest:(NSMutableURLRequest*)urlRequst
compelete:(QCloudHTTPAuthentationContinueBlock)continueBlock
{
// 签名过期时间
NSDate *expiration = [[[NSDateFormatter alloc] init]
dateFromString:@"expiredTime"];
QCloudSignature *sign = [[QCloudSignature alloc] initWithSignature:
@"后台计算好的签名" expiration:expiration];
continueBlock(signature, nil);
}
```

Swift

```
func signature(with fileds: QCloudSignatureFields!,
request: QCloudBizHTTPRequest!,
urlRequest urlRequst: NSMutableURLRequest!,
compelete continueBlock: QCloudHTTPAuthentationContinueBlock!) {
// 签名过期时间
let expiration = DateFormatter().date(from: "expiredTime");
let sign = QCloudSignature.init(signature: "后台计算好的签名",
expiration: expiration);
continueBlock(signature,nil);
}
```

第三步：访问 COS 服务

上传对象

SDK 支持上传本地文件与二进制数据 NSData。下面以上传本地文件为例。

Objective-C

```
QCloudCOSXMLUploadObjectRequest* put = [QCloudCOSXMLUploadObjectRequest new];
// 本地文件路径
NSURL* url = [NSURL fileURLWithPath:@"文件的URL"];
```

```
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
put.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = @"exampleobject";
//需要上传的对象内容。可以传入NSData*或者NSURL*类型的变量
put.body = url;
//监听上传进度
[put setSendProcessBlock:^(int64_t bytesSent,
int64_t totalBytesSent,
int64_t totalBytesExpectedToSend) {
// bytesSent 本次要发送的字节数（一个大文件可能要分多次发送）
// totalBytesSent 已发送的字节数
// totalBytesExpectedToSend 本次上传要发送的总字节数（即一个文件大小）
}];
//监听上传结果
[put setFinishBlock:^(id outputObject, NSError *error) {
//可以从 outputObject 中获取 response 中 etag 或者自定义头部等信息
NSDictionary * result = (NSDictionary *)outputObject;
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:put];
```

说明：

- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见**生成预签名链接**文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

Swift

```
let put:QCloudCOSXMLUploadObjectRequest = QCloudCOSXMLUploadObjectRequest<AnyObject>();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
put.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = "exampleobject";
//需要上传的对象内容。可以传入NSData*或者NSURL*类型的变量
put.body = NSURL.fileURL(withPath: "Local File Path") as AnyObject;
//监听上传结果
put.setFinish { (result, error) in
// 获取上传结果
if error != nil{
```

```
print(error!);
}else{
print(result!);
}
}
//监听上传进度
put.sendProcessBlock = { (bytesSent, totalBytesSent,
totalBytesExpectedToSend) in
// bytesSent 本次要发送的字节数（一个大文件可能要分多次发送）
// totalBytesSent 已发送的字节数
// totalBytesExpectedToSend 本次上传要发送的总字节数（即一个文件大小）
};
//设置上传参数
put.initMultipleUploadFinishBlock = {(multipleUploadInitResult, resumeData) in
//在初始化分块上传完成以后会回调该 block，在这里可以获取 resumeData
//并且可以通过 resumeData 生成一个分块上传的请求
let resumeUploadRequest = QCloudCOSXMLUploadObjectRequest<AnyObject>
.init(request: resumeData as Data?);
}
QCloudCOSTransferMangerService.defaultCOSTransferManager().uploadObject(put);
```

说明：

- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见**生成预签名链接**文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

下载对象

Objective-C

```
QCloudCOSXMLDownloadObjectRequest * request = [QCloudCOSXMLDownloadObjectRequest
new];

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
//设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = [NSURL fileURLWithPath:@"Local File Path"];
//监听下载结果
[request setFinishBlock:^(id outputObject, NSError *error) {
//outputObject 包含所有的响应 http 头部
```

```
NSDictionary* info = (NSDictionary *) outputObject;
}];
//监听下载进度
[request setDownProcessBlock:^(int64_t bytesDownload,
int64_t totalBytesDownload,
int64_t totalBytesExpectedToDownload) {
// bytesDownload 本次要下载的字节数（一个大文件可能要分多次发送）
// totalBytesDownload 已下载的字节数
// totalBytesExpectedToDownload 本次要下载的总字节数（即一个文件大小）
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] DownloadObject:reques
t];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let request : QCloudCOSXMLDownloadObjectRequest = QCloudCOSXMLDownloadObjectReque
st();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
request.bucket = "examplebucket-1250000000";
// 对象键
request.object = "exampleobject";
//设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = NSURL.fileURL(withPath: "Local File Path") as URL?;
//监听下载进度
request.sendProcessBlock = { (bytesDownload, totalBytesDownload,
totalBytesExpectedToDownload) in
// bytesDownload 本次要下载的字节数（一个大文件可能要分多次发送）
// totalBytesDownload 已下载的字节数
// totalBytesExpectedToDownload 本次要下载的总字节数（即一个文件大小）
}
//监听下载结果
request.finishBlock = { (copyResult, error) in
if error != nil{
print(error!);
}else{
print(copyResult!);
}
}
```

```
QCloudCOSTransferMangerService.defaultCOSTransferManager().downloadObject(reques
t);
```

说明：
更多完整示例，请前往 GitHub 查看。

iOS SDK 常见问题

最近更新时间：2023-02-27 16:36:42

手动集成 SDK 后，设置 QCloudCOSXMLEndPoint 实例的 regionName，抛出异常“[__NSCFConstantString matchesRegularExpression:]: unrecognized selector sent to instance xxx”，该如何处理？

原因：matchesRegularExpression 是 NSString 的分类提供的方法。Objective-C 链接器不会为每个方法建立符号表，只是为类建立了符号表。SDK 是静态库，如果静态库中定义了一份已经存在的类的分类，系统就会认为这个类已经存在，则不会将类和类的分类的代码整合起来，在最后的可执行代码中，就会缺少分类中的方法。

解决办法：

1. 在 target > Build Settings > All > Other Linker Flags 中添加 `-Objc` 和 `-all_load` 两个参数。
2. `-ObjC` 会将静态库中的类和分类都加载到最后的可执行文件中，但是如果库中只有分类没有类的话，这个参数就会失效，就需要 `-all_load`，该参数会把所有找到的目标文件都加到可执行文件中。
3. 若上述方法依然没有解决该错误，请尝试关闭 bitcode。

集成 SDK 发送请求之后抛出异常“您没有配置默认的 OCR 服务配置，请配置之后再调用该方法或者您没有配置 Key 为 'xxx' 的 OCR 服务配置，请配置之后再调用该方法”，该如何处理？

原因：SKD 所有的请求都依赖于一个 QCloudCOSXMLServeice 和 QCloudCOSTransferMangerService（上传的高级接口依赖于该实例），如果在发送请求之前没有注册对应的 service 会抛出该异常。

解决办法：通过以下代码注册请求所需要的 service 实例，并确保该实例在请求发出前已存在。

```
QCloudServiceConfiguration *configuration = [QCloudServiceConfiguration new];
configuration.appID = "AppId";
configuration.signatureProvider = self;
QCloudCOSXMLEndPoint *endpoint = [[QCloudCOSXMLEndPoint alloc] init];
endpoint.regionName = @"region";
endpoint.useHTTPS = YES;
configuration.endpoint = endpoint;

[QCloudCOSXMLService registerDefaultCOSXMLWithConfiguration:configuration];
[QCloudCOSTransferMangerService registerDefaultCOSTransferMangerWithConfiguratio
n:configuration];
```

集成 SDK 后抛出异常“默认的 COSXMLService 已存在，如有新的配置，请通过 registerCOSXMLWithConfiguration:withKey:重新注册”，该如何处理？

原因：在 SDK 中，不同的 COSXMLService 实例对应不同的配置（配置信息可以参考相关属性），例如 regionName 为 ap-guangzhou 和 ap-beijing 是两个不同的配置，需要注册两个不同的 service，如果在用 ap-

版权所有：腾讯云计算（北京）有限责任公司

guangzhou 注册了一个 service 之后，将其 region 又改成 ap-guangzhou 重新注册就会触发该异常。

解决办法：

1. 默认的 service 即通过 `registerDefaultCOSXMLWithConfiguration：` 不需要指定 key。
2. 通过以下代码注册一个新的 service：

```
//判断要注册的 key 是否存在,
if(![QCloudCOSXMLService hasServiceForKey:@"要注册的 key"]){
//不存在注册一个新的 servide
[QCloudCOSXMLService registerCOSXMLWithConfiguration:configuration withKey:@"要注册的 key"]
}
```

集成 SDK 之后运行，代理“- (void)signatureWithFields:(QCloudSignatureFields *)fileds request:urlRequest:compelete:”没有被调用，该如何处理？

原因：该代理方法是获取密钥/签名的方法，为了最大程度的保证密钥的有效性（调用签名过早容易过期），SDK 会在请求发出之前即 `task resume` 之前才会调用。

解决办法：

1. 请确保该代理方法所在的类在请求发出之前不会被销毁，建议将该代理方法在一个单例类中实现。
2. 是否在创建 `QCloudServiceConfiguration 实例之后` ，设置其 `signatureProvider` ，例如 `configuration.signatureProvider = self(self 为实现代理方法所在的类)` 。
3. 如果以上都没问题，请检查是否发送请求。

SDK 能否缓存和复用密钥，在密钥过期之后重新请求新的密钥，该如何处理？

SDK 提供了 QCloudCredentailFenceQueue 来实现密钥的缓存和复用，具体使用方法请参考 快速入门 文档。

通过 SDK 的高级接口“QCloudCOSXMLUploadObjectRequest”抛出异常“不支持设置该类型的 body，支持的类型为 NSData、QCloudFileOffsetBody、NSURL”，该如何处理？

SDK 目前只支持设置以下三种类型的 body：

1. NSURL：文件的本地路径，通过 `[NSURL fileURLWithPath:@"文件在本地的路径"]` 初始化一个 URL。
2. NSData：二进制数据。
3. QCloudFileOffsetBody：分块的 body，开发者一般不需要关心该类型，属于 SDK 高级接口内部使用的 body 类型。

通过 SDK 的高级接口“QCloudCOSXMLUploadObjectRequest”上传系统图库中的视频或者文件，断点续传失败，报错“The specified Content-Length is zero”，该如何处理？

SDK 只支持续传沙盒中的文件，如需使用断点续传的功能，请先将文件移到沙盒中。

版权所有：腾讯云计算（北京）有限责任公司

集成 SDK 后，调用上传接口上传的文件大小和本地的文件不一致，该如何处理？

请确保您设置好 body 之后，本地文件不会发生改变，例如文件在压缩过程中或者还没有完成写入之前调用了上传接口触发了上传，SDK 就会以当时的文件大小为准进行分块进行上传，从而导致上传到 cos 上的文件和本地文件大小不一致。

集成 SDK 后，调用上传接口，报错 "您输入的 body 的URL不是本地 URL，请检查后使用！"该如何处理？

解决办法：

请确保 URL 是以 file:// 开头，可以通过以下两种方式初始化 ：

1. `[NSURL URLWithString:@"file:////var/mobile/Containers/Data/Application/DBPF7490-D5U8-4ABF-A0AF-CC49D6A60AEB/Documents/exampleobject"]`
2. `[NSURL fileURLWithPath:@"/var/mobile/Containers/Data/Application/DBPF7490-D5U8-4ABF-A0AF-CC49D6A60AEB/Documents/exampleobject"]`

集成 SDK 之后，调用上传接口，上传成功之后的文件大小为0，该如何处理？

解决办法：

1. 如果上传路径是文件在系统图库的路径，请检查是否有读取该文件的权限。例如：`file:///var/mobile/Media/DCIM/101APPLE/` 这个路径无法直接访问，需要通过 Photos 框架里的 request 方法获取照片。
2. 如果 App 要兼容 iOS11，在上传图库视频时再调用 `[[PHImageManager defaultManager] requestPlayerItemForVideo:asset options:option resultHandler:^(AVPlayerItem *playerItem, NSDictionary *info) { //及时将文件移到沙盒中或者保存 playerItem }]` ；方法获取 playerItem 之后将其保存起来，或者在该回调中将要上传的文件移到 App 的沙盒中。因为 iOS11中 playerItem 被销毁之后，文件的读权限就会失效，从而导致上传的文件大小为0。
3. 如果上传的是 App 的沙盒中的文件，请检查上传的文件是否在沙盒的 tmp 文件夹下，例如 `/var/mobile/Containers/Data/Application/0BFBB3FE-0FD0-46CB-ADDE-DDE08F6D62C3/tmp/` ，该目录下的文件会被系统随时清理，请将要上传的文件移到安全的目录，保证文件在上传的过程中不会被清理，更多关于沙盒的描述请参考 File System Basics。

集成 SDK 后使用高级接口上传报错“上传过程中 MD5校验与本地不一致，请检查本地文件在上传过程中是否发生了变化：分块上传过程中，每上传完一个分块就会校验这个分块的 MD5 和本地片的 MD5 是否一致，不一致就报错”，该如何处理？

原因：该错误发生在上传大于1MB的文件的情况下，超过1MB的文件使用 SDK 上传时会根据将文件分成若干个1MB的文件进行分块上传，在上传完每一个分块之后会用后台返回的 etag 和本地文件的分块进行对比，如果发现不一致会抛出该错误。

解决办法：请检查文件在上传的过程中是否发生改变。

SDK 能否使用 CDN 加速域名进行访问？

支持，请根据您所使用的编程语言，并参见对应的SDK 文档进行操作。

SDK 如何设置请求的超时时间?

解决：SDK 5.7.0 之后支持自定义请求的超时时间，可以通过以下方式设置：

1. 初始化 `QCloudServiceConfiguration *config = [QCloudServiceConfiguration new]`
2. 设置 config 的 `timeoutInterval` 属性即可，例如设置 `config 的.timeoutInterval = 30;`

COS iOS SDK 出现 crash，该如何处理？

请检查当前版本是否为最新版本，若是旧版本建议您升级到最新的 SDK 版本。最新版本 iOS SDK 请参见 qcloud-sdk-ios。

“Undefined symbols for architecture arm64: "_ne10 init dsp", referenced from:”类错误，该如何处理？

可以参见 编译报错：Undefined symbols for architecture arm64。

快速体验

最近更新时间：2022-04-11 12:43:07

简介

本文档将介绍如何配置客户端和运行示例 Demo。

相关资源

本文涉及的所有工具和 Demo 都存放在 Github 仓库，用户可前往获取。

搭建用户客户端

配置客户端

修改 QCloudCOSXMLDemo/QCloudCOSXMLDemo/key.json 文件，填入 APPID，secretID，secretKey 等参数值，然后执行以下命令：

```
pod install
```

> 说明：
> APPID，secretID，secretKey 可前往 API 密钥管理 页面获取。

执行命令完成后，打开 QCloudCOSXMLDemo.xcworkspace 即可进入 Demo 体验。

运行示例 Demo

查询存储桶列表

启动示例 App 后，将展示当前用户已创建的存储桶。

创建存储桶

点击右上角**新建桶**，在配置页面输入桶名称并选择存储桶的所属 地域。

查询对象列表

版权所有：腾讯云计算（北京）有限责任公司

选择某个存储桶，进入存储桶详情，将看到该存储桶内所有文件以及文件夹。

上传文件

在文件列表页面点击右上角的**上传**，然后选择照片进行上传，支持设置文件的访问权限和传输状态控制。

下载或删除文件

选择并点击文件，然后点击文件的下载或删除按钮，对文件进行下载或删除操作。

版权所有：腾讯云计算（北京）有限责任公司

存储桶操作

最近更新时间：2022-12-15 16:07:49

简介

本文档提供关于存储桶基本操作的相关 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Service（List Buckets）	查询存储桶列表	查询指定账号下所有的存储桶列表
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

查询存储桶列表

功能说明

用于查询指定账号下所有存储桶列表。

示例代码

Objective-C

```
// 获取所属账户的所有存储空间列表的方法
QCloudGetServiceRequest* request = [[QCloudGetServiceRequest alloc] init];
[request setFinishBlock:^(QCloudListAllMyBucketsResult* result,
NSError* error) {

// 从 result 中获取返回信息 存储桶列表
NSArray<QCloudBucket*> *buckets = result.buckets;

// bucket owner的信息
```

```
QCloudOwner *owner = result.owner;
}];
[[QCloudCOSXMLService defaultCOSXML] GetService:request];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
// 获取所属账户的所有存储空间列表的方法.
let getServiceReq = QCloudGetServiceRequest.init();
getServiceReq.setFinish{(result,error) in
if let result = result {
let buckets = result.buckets
let owner = result.owner
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getService(getServiceReq);
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

创建存储桶

功能说明

创建一个存储桶（PUT Bucket）。

示例代码

Objective-C

```
// 创建存储桶
QCloudPutBucketRequest* request = [QCloudPutBucketRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
```

```
[request setFinishBlock:^(id outputObject, NSError* error) {
// 可以从 outputObject 中获取服务器返回的 header 信息
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucket:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
// 创建存储桶
let putBucketReq = QCloudPutBucketRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
putBucketReq.bucket = "examplebucket-1250000000";
putBucketReq.finishBlock = {(result,error) in
// 可以从 result 中获取服务器返回的 header 信息
if error != nil {
print(error!);
} else {
print(result!);
}
}
QCloudCOSXMLService.defaultCOSXML().putBucket(putBucketReq);
```

说明：
更多完整示例，请前往 GitHub 查看。

检索存储桶及其权限

功能说明

HEAD Bucket 请求可以确认该存储桶是否存在，是否有权限访问。有以下几种情况：

- 存储桶存在且有读取权限，返回 HTTP 状态码为200。
- 无存储桶读取权限，返回 HTTP 状态码为403。
- 存储桶不存在，返回 HTTP 状态码为404。

示例代码

Objective-C

```
QCloudHeadBucketRequest* request = [QCloudHeadBucketRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
[request setFinishBlock:^(id outputObject, NSError* error) {
// 可以从 outputObject 中获取服务器返回的 header 信息
NSDictionary * result = (NSDictionary *)outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] HeadBucket:request];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let headBucketReq = QCloudHeadBucketRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
headBucketReq.bucket = "examplebucket-1250000000";
headBucketReq.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().headBucket(headBucketReq);
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

判断存储桶是否存在

功能说明

您可以通过 SDK 提供的快捷接口来判断 Bucket 是否存在

示例代码

Objective-C

```
// 存储桶名称，格式为 BucketName-APPID
[[QCloudCOSXMLService defaultCOSXML] doesBucketExist: @"examplebucket-1250000000"
];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
// 存储桶名称，格式为 BucketName-APPID
QCloudCOSXMLService.defaultCOSXML().doesBucketExist("examplebucket-1250000000");
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

删除存储桶

功能说明

删除指定的存储桶（DELETE Bucket）。

> 注意：
> 删除存储桶前，请确保存储桶内的数据和未完成上传的分块数据已全部清空，否则会无法删除存储桶。

示例代码

Objective-C

```
QCloudDeleteBucketRequest* request = [[QCloudDeleteBucketRequest alloc ] init];
// 存储桶名称，命名格式：BucketName-APPID
```

版权所有：腾讯云计算（北京）有限责任公司

```
request.bucket = @"examplebucket-1250000000";
[request setFinishBlock:^(id outputObject,NSError*error) {
// 可以从 outputObject 中获取服务器返回的 header 信息
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteBucket:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let deleteBucketReq = QCloudDeleteBucketRequest.init();
// 存储桶名称，命名格式：BucketName-APPID
deleteBucketReq.bucket = "examplebucket-1250000000";
deleteBucketReq.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().deleteBucket(deleteBucketReq);
```

说明：
更多完整示例，请前往 GitHub 查看。

对象操作
上传对象

最近更新时间：2022-05-25 15:25:01

简介

本文档提供关于上传对象的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object	简单上传对象	上传一个对象至存储桶

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传操作
Upload Part	上传分块	分块上传对象
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

高级接口（推荐）

上传对象

高级接口封装了简单上传、分块上传接口，根据文件大小智能的选择上传方式，同时支持续传功能。

版权所有：腾讯云计算（北京）有限责任公司

示例代码一: 上传本地文件

Objective-C

```
QCloudCOSXMLUploadObjectRequest* put = [QCloudCOSXMLUploadObjectRequest new];
/** 本地文件路径，请确保 URL 是以 file:// 开头，格式如下 ：
1. [NSURL URLWithString:@"file:///var/mobile/Containers/Data/Application/DBPF7490-D5U8-4ABF-A0AF-CC49D6A60AEB/Documents/exampleobject"]
2. [NSURL fileURLWithPath:@"/var/mobile/Containers/Data/Application/DBPF7490-D5U8-4ABF-A0AF-CC49D6A60AEB/Documents/exampleobject"]
*/
NSURL* url = [NSURL fileURLWithPath:@"文件的URL"];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
put.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = @"exampleobject";
// 需要上传的对象内容。可以传入NSData*或者NSURL*类型的变量
put.body = url;
// 监听上传进度
[put setSendProcessBlock:^(int64_t bytesSent,
int64_t totalBytesSent,
int64_t totalBytesExpectedToSend) {
// bytesSent 本次要发送的字节数（一个大文件可能要分多次发送）
// totalBytesSent 已发送的字节数
// totalBytesExpectedToSend 本次上传要发送的总字节数（即一个文件大小）
}];
// 监听上传结果
[put setFinishBlock:^(QCloudUploadObjectResult *result, NSError *error) {
// 在上传结果 result.location 中获取上传文件的下载链接
NSString * fileUrl = result.location;
// 获取文件crc64
NSString * crc64 = [[outputObject __originHTTPURLResponse__].allHeaderFields valueForKey:@"x-cos-hash-crc64ecma"];
}];
[put setInitMultipleUploadFinishBlock:^(QCloudInitiateMultipartUploadResult *multipleUploadInitResult,
QCloudCOSXMLUploadObjectResumeData resumeData) {
// 在初始化分块上传完成以后会回调该 block，在这里可以获取 resumeData，uploadid
NSString* uploadId = multipleUploadInitResult.uploadId;
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:put];
```

说明：

- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 **生成预签名链接** 文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

Swift

```
let put:QCloudCOSXMLUploadObjectRequest = QCloudCOSXMLUploadObjectRequest<AnyObject>();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
put.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = "exampleobject";
// 需要上传的对象内容。可以传入NSData*或者NSURL*类型的变量
put.body = NSURL.fileURL(withPath: "Local File Path") as AnyObject;
// 监听上传结果
put.setFinish { (result, error) in
// 获取上传结果
if let result = result {
// 文件的 etag
let eTag = result.eTag
// 文件下载链接
let location = result.location;
// 获取文件crc64
let crc64 = result?.__originHTTPURLResponse__.allHeaderFields["x-cos-hash-crc64ecma"];
} else {
print(error!);
}
}
// 监听上传进度
put.sendProcessBlock = { (bytesSent, totalBytesSent,
totalBytesExpectedToSend) in
// bytesSent 本次要发送的字节数（一个大文件可能要分多次发送）
// totalBytesSent 已发送的字节数
// totalBytesExpectedToSend 本次上传要发送的总字节数（即一个文件大小）
};
// 设置上传参数
put.initMultipleUploadFinishBlock = {(multipleUploadInitResult, resumeData) in
// 在初始化分块上传完成以后会回调该 block，在这里可以获取 resumeData,以及 uploadId
if let multipleUploadInitResult = multipleUploadInitResult {
let uploadId = multipleUploadInitResult.uploadId
}
}
QCloudCOSTransferMangerService.defaultCOSTransferManager().uploadObject(put);
```

说明：

- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 **生成预签名链接** 文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

示例代码二: 上传二进制数据

Objective-C

```
QCloudCOSXMLUploadObjectRequest* put = [QCloudCOSXMLUploadObjectRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
put.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = @"exampleobject";
// 需要上传的对象内容。可以传入NSData*或者NSURL*类型的变量
put.body = [@"My Example Content" dataUsingEncoding:NSUTF8StringEncoding];
// 监听上传进度
[put setSendProcessBlock:^(int64_t bytesSent,
int64_t totalBytesSent,
int64_t totalBytesExpectedToSend) {
// bytesSent 新增字节数
// totalBytesSent 本次上传的总字节数
// totalBytesExpectedToSend 本地上传的目标字节数
}];
// 监听上传结果
[put setFinishBlock:^(QCloudUploadObjectResult *result, NSError *error) {
// 在上传结果 result.location 中获取上传文件的下载链接
NSString * fileUrl = result.location;
// 获取文件crc64
NSString * crc64 = [[outputObject __originHTTPURLResponse__].allHeaderFields valueForKey:@"x-cos-hash-crc64ecma"];
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:put];
```

说明：

- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 **生成预签名链接** 文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

Swift

```
let put:QCloudCOSXMLUploadObjectRequest = QCloudCOSXMLUploadObjectRequest<AnyObject>();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
put.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = "exampleobject";
// 需要上传的对象内容
let dataBody:NSData = "wrwrwrwrwrw".data(using: .utf8)! as NSData;
put.body = dataBody;
// 监听上传结果
put.setFinish { (result, error) in
// 获取上传结果
if let result = result {
// 文件的 etag
let eTag = result.eTag
// 文件下载链接
let location = result.location;
// 获取文件crc64
let crc64 = result?.__originHTTPURLResponse__.allHeaderFields["x-cos-hash-crc64ecma"];
} else {
print(error!);
}
}
// 监听上传进度
put.sendProcessBlock = { (bytesSent, totalBytesSent,
totalBytesExpectedToSend) in

// bytesSent 新增字节数
// totalBytesSent 本次上传的总字节数
// totalBytesExpectedToSend 本地上传的目标字节数
};
QCloudCOSTransferMangerService.defaultCOSTransferManager().uploadObject(put);
```

说明：

- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 **生成预签名链接** 文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

示例代码三: 上传暂停、继续与取消

Objective-C

对于上传任务，可以通过以下方式暂停：

```
NSError *error;
NSData *resmeData = [put cancelByProductingResumeData:&error];
```

暂停之后，可以通过以下方式续传：

```
QCloudCOSXMLUploadObjectRequest *resumeRequest = [QCloudCOSXMLUploadObjectRequest requestWithRequestData:resmeData];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:resumeRequest];
```

也通过以下方式取消上传：

```
//丢弃该上传
[put abort:^(id outputObject, NSError *error) {

}];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

对于上传任务，可以通过以下方式暂停：

```
var error : NSError?;
var uploadResumeData:Data = put.cancel(byProductingResumeData:&error) as Data;
```

暂停之后，可以通过以下方式续传：

```
var resumeRequest = QCloudCOSXMLUploadObjectRequest<AnyObject>.init(request: uploadResumeData);
QCloudCOSTransferMangerService.defaultCOSTransferManager().uploadObject(resumeRequest);
```

也通过以下方式取消上传：

```
//丢弃该上传
put.abort { (outputObject, error) in

}
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码四: 批量上传

Objective-C

```
for (int i = 0; i<20; i++) {
QCloudCOSXMLUploadObjectRequest* put = [QCloudCOSXMLUploadObjectRequest new];

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
put.bucket = @"examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = [NSString stringWithFormat:@"exampleobject-%d",i];

// 需要上传的对象内容。可以传入NSData*或者NSURL*类型的变量
put.body = [@"My Example Content" dataUsingEncoding:NSUTF8StringEncoding];

// 监听上传进度
[put setSendProcessBlock:^(int64_t bytesSent,
int64_t totalBytesSent,
int64_t totalBytesExpectedToSend) {
// bytesSent 本次要发送的字节数（一个大文件可能要分多次发送）
// totalBytesSent 已发送的字节数
// totalBytesExpectedToSend 本次上传要发送的总字节数（即一个文件大小）
}];

// 监听上传结果
[put setFinishBlock:^(QCloudUploadObjectResult *result, NSError *error) {
// 在上传结果 result.location 中获取上传文件的下载链接
NSString * fileUrl = result.location;
// 获取文件crc64
NSString * crc64 = [[outputObject __originHTTPURLResponse__].allHeaderFields valueForKey:@"x-cos-hash-crc64ecma"];
}];
```

```
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:put];
}
```

Swift

```
for i in 1...10 {
let put:QCloudCOSXMLUploadObjectRequest = QCloudCOSXMLUploadObjectRequest<AnyObje
ct>();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
put.bucket = "examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = "exampleobject-".appendingFormat("%d", i);

// 需要上传的对象内容
let dataBody:NSData = "wrwrwrwrwrw".data(using: .utf8)! as NSData;
put.body = dataBody;

// 监听上传结果
put.setFinish { (result, error) in
// 获取上传结果
if let result = result {
// 文件的 etag
let eTag = result.eTag
// 文件的下载链接
let location = result.location;
// 获取文件crc64
let crc64 = result?.__originHTTPURLResponse__.allHeaderFields["x-cos-hash-crc64ec
ma"];
} else {
print(error!);
}
}
// 监听上传进度
put.sendProcessBlock = { (bytesSent, totalBytesSent,
totalBytesExpectedToSend) in

// bytesSent 本次要发送的字节数（一个大文件可能要分多次发送）
// totalBytesSent 已发送的字节数
// totalBytesExpectedToSend 本次上传要发送的总字节数（即一个文件大小）
};

QCloudCOSTransferMangerService.defaultCOSTransferManager().uploadObject(put);
}
```

示例代码五: 自定义简单上传与分块上传的阈值

Objective-C

```
for (int i = 0; i<20; i++) {
QCloudCOSXMLUploadObjectRequest* put = [QCloudCOSXMLUploadObjectRequest new];

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
put.bucket = @"examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = [NSString stringWithFormat:@"exampleobject-%d",i];

// 需要上传的对象内容。可以传入NSData*或者NSURL*类型的变量
put.body = [@"My Example Content" dataUsingEncoding:NSUTF8StringEncoding];
// 自定义简单上传与分块上传的阈值：默认当文件大于1MB时启用分块上传
put.mutilThreshold = 10 *1024*1024;
// 监听上传进度
[put setSendProcessBlock:^(int64_t bytesSent,
int64_t totalBytesSent,
int64_t totalBytesExpectedToSend) {
// bytesSent 本次要发送的字节数（一个大文件可能要分多次发送）
// totalBytesSent 已发送的字节数
// totalBytesExpectedToSend 本次上传要发送的总字节数（即一个文件大小）
}];

// 监听上传结果
[put setFinishBlock:^(QCloudUploadObjectResult *result, NSError *error) {
// 在上传结果 result.location 中获取上传文件的下载链接
NSString * location = result.location;
// 获取文件crc64
NSString * crc64 = [[outputObject __originHTTPURLResponse__].allHeaderFields valueForKey:@"x-cos-hash-crc64ecma"];
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:put];
}
```

Swift

```
for i in 1...10 {
let put:QCloudCOSXMLUploadObjectRequest = QCloudCOSXMLUploadObjectRequest<AnyObject>();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
put.bucket = "examplebucket-1250000000";
```

```
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = "exampleobject-".appendingFormat("%d", i);

// 需要上传的对象内容
let dataBody:NSData = "wrwrwrwrwrw".data(using: .utf8)! as NSData;
put.body = dataBody;
// 自定义简单上传与分块上传的阈值：默认当文件大于1MB时启用分块上传
put.mutilThreshold = 10 *1024*1024;
// 监听上传结果
put.setFinish { (result, error) in
// 获取上传结果
if let result = result {
// 文件的 etag
let eTag = result.eTag
// 文件下载链接
let location = result.location;
// 获取文件crc64
let crc64 = result?.__originHTTPURLResponse__.allHeaderFields["x-cos-hash-crc64ecma"];
} else {
print(error!);
}
}
// 监听上传进度
put.sendProcessBlock = { (bytesSent, totalBytesSent,
totalBytesExpectedToSend) in

// bytesSent 本次要发送的字节数（一个大文件可能要分多次发送）
// totalBytesSent 已发送的字节数
// totalBytesExpectedToSend 本次上传要发送的总字节数（即一个文件大小）
};

QCloudCOSTransferMangerService.defaultCOSTransferManager().uploadObject(put);
}
```

示例代码六: 自定义分块大小

Objective-C

```
for (int i = 0; i<20; i++) {
QCloudCOSXMLUploadObjectRequest* put = [QCloudCOSXMLUploadObjectRequest new];

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
put.bucket = @"examplebucket-1250000000";
```

```
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = [NSString stringWithFormat:@"exampleobject-%d",i];

// 需要上传的对象内容。可以传入NSData*或者NSURL*类型的变量
put.body = [@"My Example Content" dataUsingEncoding:NSUTF8StringEncoding];
// 自定义分片大小，默认为1MB
put.sliceSize = 10 *1024*1024;
// 监听上传进度
[put setSendProcessBlock:^(int64_t bytesSent,
int64_t totalBytesSent,
int64_t totalBytesExpectedToSend) {
// bytesSent 本次要发送的字节数（一个大文件可能要分多次发送）
// totalBytesSent 已发送的字节数
// totalBytesExpectedToSend 本次上传要发送的总字节数（即一个文件大小）
}];

// 监听上传结果
[put setFinishBlock:^(QCloudUploadObjectResult *result, NSError *error) {
// 在上传结果 result.location 中获取上传文件的下载链接
NSString * location = result.location;
// 获取文件crc64
NSString * crc64 = [[outputObject __originHTTPURLResponse__].allHeaderFields valu
eForKey:@"x-cos-hash-crc64ecma"];
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:put];
}
```

Swift

```
for i in 1...10 {
let put:QCloudCOSXMLUploadObjectRequest = QCloudCOSXMLUploadObjectRequest<AnyObje
ct>();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
put.bucket = "examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = "exampleobject-".appendingFormat("%d", i);

// 需要上传的对象内容
let dataBody:NSData = "wrwrwrwrwrw".data(using: .utf8)! as NSData;
put.body = dataBody;
// 自定义分片大小，默认为1MB
put.sliceSize = 10 *1024*1024
// 监听上传结果
```

```
put.setFinish { (result, error) in
// 获取上传结果
if let result = result {
// 文件的 etag
let eTag = result.eTag
// 文件下载链接
let location = result.location;
// 获取文件crc64
let crc64 = result?.__originHTTPURLResponse__.allHeaderFields["x-cos-hash-crc64ecma"];
} else {
print(error!);
}
}
// 监听上传进度
put.sendProcessBlock = { (bytesSent, totalBytesSent,
totalBytesExpectedToSend) in
// bytesSent 本次要发送的字节数（一个大文件可能要分多次发送）
// totalBytesSent 已发送的字节数
// totalBytesExpectedToSend 本次上传要发送的总字节数（即一个文件大小）
};

QCloudCOSTransferMangerService.defaultCOSTransferManager().uploadObject(put);
}
```

示例代码七: 上传时限速

> 注意：
> 需要 COS iOS SDK v5.8.0 及以上版本。

Objective-C

```
QCloudCOSXMLUploadObjectRequest* put = [QCloudCOSXMLUploadObjectRequest new];

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
put.bucket = @"examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = [NSString stringWithFormat:@"exampleobject-%d",i];

// 需要上传的对象内容。可以传入NSData*或者NSURL*类型的变量
put.body = [@"My Example Content" dataUsingEncoding:NSUTF8StringEncoding];
```

```
// 使用trafficLimit参数设置上传限速，单位为bit/s，限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s。
put.trafficLimit = 819200;
// 监听上传进度
[put setSendProcessBlock:^(int64_t bytesSent,
int64_t totalBytesSent,
int64_t totalBytesExpectedToSend) {
// bytesSent 本次要发送的字节数（一个大文件可能要分多次发送）
// totalBytesSent 已发送的字节数
// totalBytesExpectedToSend 本次上传要发送的总字节数（即一个文件大小）
}];

// 监听上传结果
[put setFinishBlock:^(QCloudUploadObjectResult *result, NSError *error) {
// 在上传结果 result.location 中获取上传文件的下载链接
NSString * location = result.location;
// 获取文件crc64
NSString * crc64 = [[outputObject __originHTTPURLResponse__].allHeaderFields valueForKey:@"x-cos-hash-crc64ecma"];
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:put];
```

Swift

```
let put:QCloudCOSXMLUploadObjectRequest = QCloudCOSXMLUploadObjectRequest<AnyObject>();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
put.bucket = "examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = "exampleobject-".appendingFormat("%d", i);

// 需要上传的对象内容
let dataBody:NSData = "wrwrwrwrwrw".data(using: .utf8)! as NSData;
put.body = dataBody;
// 使用trafficLimit参数设置上传限速，单位为bit/s，限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s。
put.trafficLimit = 819200;
// 监听上传结果
put.setFinish { (result, error) in
// 获取上传结果
if let result = result {
// 文件的 etag
let eTag = result.eTag
```

```
// 文件下载链接
let location = result.location;
// 获取文件crc64
let crc64 = result?.__originHTTPURLResponse__.allHeaderFields["x-cos-hash-crc64ec
ma"];
} else {
print(error!);
}
}
// 监听上传进度
put.sendProcessBlock = { (bytesSent, totalBytesSent,
totalBytesExpectedToSend) in
// bytesSent 本次要发送的字节数（一个大文件可能要分多次发送）
// totalBytesSent 已发送的字节数
// totalBytesExpectedToSend 本次上传要发送的总字节数（即一个文件大小）
};

QCloudCOSTransferMangerService.defaultCOSTransferManager().uploadObject(put);
```

示例代码八: 创建目录

Objective-C

```
QCloudCOSXMLUploadObjectRequest* put = [QCloudCOSXMLUploadObjectRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
put.bucket = @"examplebucket-1250000000";
// 要创建的目录的名称
put.object = @"dir1";

// 需要上传的对象内容，如需创建目录使用空串生成NSData即可
put.body = [@"" dataUsingEncoding:NSUTF8StringEncoding];

// 监听上传结果
[put setFinishBlock:^(QCloudUploadObjectResult *result, NSError *error) {
// 在上传结果 result.location 中获取上传文件的下载链接
NSString * location = result.location;
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:put];
```

说明：

- 更多完整示例，请前往 GitHub 查看。

- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 **生成预签名链接** 文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

Swift

```
let put:QCloudCOSXMLUploadObjectRequest = QCloudCOSXMLUploadObjectRequest<AnyObject>();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
put.bucket = "examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = "dir/";

// 需要上传的对象内容
let dataBody:NSData = "".data(using: .utf8)! as NSData;
put.body = dataBody;

// 监听上传结果
put.setFinish { (result, error) in
// 获取上传结果
if let result = result {
// 文件下载链接
let location = result.location;
} else {
print(error!);
}
}
QCloudCOSTransferMangerService.defaultCOSTransferManager().uploadObject(put);
```

说明：

- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 **生成预签名链接** 文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

简单操作

简单上传对象

功能说明

PUT Object 接口可以上传一个对象至指定存储桶中，该操作需要请求者对存储桶有 WRITE 权限。最大支持上传不超过5GB的对象，5GB以上对象请使用 分块上传 或 高级接口 上传。

> 注意：
> Key（文件名）不能以 `/` 结尾，否则会被识别为文件夹。

示例代码

Objective-C

```
QCloudPutObjectRequest* put = [QCloudPutObjectRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
put.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = @"exampleobject";
// 文件内容，可以传入NSData*或者NSURL*类型的变量
put.body = [@"testFileContent" dataUsingEncoding:NSUTF8StringEncoding];
[put setFinishBlock:^(id result, NSError *error) {
// result 包含响应的 header 信息
// 获取文件crc64
NSString * crc64 = [[outputObject __originHTTPURLResponse__].allHeaderFields valueForKey:@"x-cos-hash-crc64ecma"];
}];
[[QCloudCOSXMLService defaultCOSXML] PutObject:put];
```

> 说明：
>
> - 更多完整示例，请前往 GitHub 查看。
> - 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 **生成预签名链接** 文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

Swift

```
let putObject = QCloudPutObjectRequest<AnyObject>.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
putObject.bucket = "examplebucket-1250000000";
// 需要上传的对象内容。可以传入NSData*或者NSURL*类型的变量
```

```
let dataBody:NSData? = "wrwrwrwrwrw".data(using: .utf8) as NSData?;
putObject.body = dataBody!;
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
putObject.object = "exampleobject";
putObject.finishBlock = {(result,error) in
if let result = result {
// 获取文件crc64
let crc64 = result?.__originHTTPURLResponse__.allHeaderFields["x-cos-hash-crc64ecma"];
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putObject(putObject);
```

说明：

- 更多完整示例，请前往 GitHub 查看。
- 上传之后，您可以用同样的 Key 生成文件下载链接，具体使用方法见 **生成预签名链接** 文档。但注意如果您的文件是私有读权限，那么下载链接只有一定的有效期。

分块操作

这里说明下分块上传的流程。

分块上传的流程

1. 初始化分块上传（Initiate Multipart Upload），得到 UploadId
2. 使用 UploadId 上传分块（Upload Part）
3. 完成分块上传（Complete Multipart Upload）

分块继续上传的流程

1. 如果没有记录 UploadId，查询分块上传任务（List Multipart Uploads），得到对应文件的 UploadId
2. 使用 UploadId 列出已上传的分块（List Parts）
3. 使用 UploadId 上传剩余的分块（Upload Part）
4. 完成分块上传（Complete Multipart Upload）

终止分块上传的流程

1. 如果没有记录 UploadId，查询分块上传任务（List Multipart Uploads），得到对应文件的 UploadId

版权所有：腾讯云计算（北京）有限责任公司

2. 终止分块上传并删除已上传分块（Abort Multipart Upload）

查询分块上传

功能说明

查询指定存储桶中正在进行的分块上传（List Multipart Uploads）。

示例代码

Objective-C

```
QCloudListBucketMultipartUploadsRequest* uploads = [QCloudListBucketMultipartUploadsRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
uploads.bucket = @"examplebucket-1250000000";
// 设置最大返回的 multipart 数量，合法取值从 1 到 1000
uploads.maxUploads = 100;
[uploads setFinishBlock:^(QCloudListMultipartUploadsResult* result,
NSError *error) {
// 可以从 result 中返回分块信息
// 进行中的分块上传对象
NSArray<QCloudListMultipartUploadContent*> *uploads = result.uploads;
}];
[[QCloudCOSXMLService defaultCOSXML] ListBucketMultipartUploads:uploads];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let listParts = QCloudListBucketMultipartUploadsRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
listParts.bucket = "examplebucket-1250000000";
// 设置最大返回的 multipart 数量，合法取值从 1 到 1000
listParts.maxUploads = 100;
listParts.setFinish { (result, error) in
if let result = result {
// 未完成的所有分块上传任务
let uploads = result.uploads;
} else {
```

```
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().listBucketMultipartUploads(listParts);
```

说明：
更多完整示例，请前往 GitHub 查看。

初始化分块上传

功能说明

初始化 Multipart Upload 上传操作，获取对应的 uploadId（Initiate Multipart Upload）。

示例代码

Objective-C

```
QCloudInitiateMultipartUploadRequest* initRequest = [QCloudInitiateMultipartUploadRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
initRequest.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
initRequest.object = @"exampleobject";
// 将作为对象的元数据返回
initRequest.cacheControl = @"cacheControl";
initRequest.contentDisposition = @"contentDisposition";
// 定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private
initRequest.accessControlList = @"public";
// 赋予被授权者读的权限。
initRequest.grantRead = @"grantRead";
// 赋予被授权者全部权限
initRequest.grantFullControl = @"grantFullControl";
[initRequest setFinishBlock:^(QCloudInitiateMultipartUploadResult* outputObject,
NSError *error) {
// 获取分块上传的 uploadId，后续的上传都需要这个 ID，请保存以备后续使用
self->uploadId = outputObject.uploadId;

}];
[[QCloudCOSXMLService defaultCOSXML] InitiateMultipartUpload:initRequest];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let initRequest = QCloudInitiateMultipartUploadRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
initRequest.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
initRequest.object = "exampleobject";
initRequest.setFinish { (result, error) in
if let result = result {
// 获取分块上传的 uploadId，后续的上传都需要这个 ID，请保存以备后续使用
self.uploadId = result.uploadId;
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().initiateMultipartUpload(initRequest);
```

说明：
更多完整示例，请前往 GitHub 查看。

上传分块

分块上传对象（Upload Part）。

示例代码

Objective-C

```
QCloudUploadPartRequest* request = [QCloudUploadPartRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 块编号
request.partNumber = 1;
// 标识本次分块上传的 ID；使用 Initiate Multipart Upload 接口初始化分块上传时会得到一个 uploadId
```

```
request.uploadId = uploadId;
// 上传的数据：支持 NSData*, NSURL(本地 URL) 和 QCloudFileOffsetBody * 三种类型
request.body = [@"testFileContent" dataUsingEncoding:NSUTF8StringEncoding];
[request setSendProcessBlock:^(int64_t bytesSent,
int64_t totalBytesSent,
int64_t totalBytesExpectedToSend) {
// 上传进度信息
// bytesSent 新增字节数
// totalBytesSent 本次上传的总字节数
// totalBytesExpectedToSend 本地上传的目标字节数
}];
[request setFinishBlock:^(QCloudUploadPartResult* outputObject, NSError *error) {
QCloudMultipartInfo *part = [QCloudMultipartInfo new];
// 获取所上传分块的 etag
part.eTag = outputObject.eTag;
part.partNumber = @"1";
// 保存起来用于最好完成上传时使用
self.parts = @[part];
// 获取文件crc64
NSString * crc64 = [[outputObject __originHTTPURLResponse__].allHeaderFields valu
eForKey:@"x-cos-hash-crc64ecma"];

}];
[[QCloudCOSXMLService defaultCOSXML] UploadPart:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let uploadPart = QCloudUploadPartRequest<AnyObject>.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
uploadPart.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
uploadPart.object = "exampleobject";
uploadPart.partNumber = 1;
// 标识本次分块上传的 ID
if let uploadId = self.uploadId {
uploadPart.uploadId = uploadId;
}
// 示例文件内容
let dataBody:NSData? = "wrwrwrwrwrwwrwrwrwrwrwwwrwrw"
```

```
.data(using: .utf8) as NSData?;
uploadPart.body = dataBody!;
uploadPart.setFinish { (result, error) in
if let result = result {
let mutipartInfo = QCloudMultipartInfo.init();
// 获取分块的 etag
mutipartInfo.eTag = result.eTag;
mutipartInfo.partNumber = "1";
// 保存起来用于最后完成上传时使用
self.parts = [mutipartInfo];
// 获取文件crc64
let crc64 = result?.__originHTTPURLResponse__.allHeaderFields["x-cos-hash-crc64ec
ma"];
} else {
print(error!);
}
}
uploadPart.sendProcessBlock = {(bytesSent,totalBytesSent,
totalBytesExpectedToSend) in
// 上传进度信息
// bytesSent 新增字节数
// totalBytesSent 本次上传的总字节数
// totalBytesExpectedToSend 本地上传的目标字节数

}
QCloudCOSXMLService.defaultCOSXML().uploadPart(uploadPart);
```

说明：
更多完整示例，请前往 GitHub 查看。

查询已上传的分块

功能说明

查询特定分块上传操作中的已上传的块（List Parts）。

示例代码

Objective-C

```
QCloudListMultipartRequest* request = [QCloudListMultipartRequest new];
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
```

```
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 在初始化分块上传的响应中，会返回一个唯一的描述符（upload ID）
request.uploadId = uploadId;
[request setFinishBlock:^(QCloudListPartsResult * _Nonnull result,
NSError * _Nonnull error) {

// 从 result 中获取已上传分块信息
// 用来表示每一个块的信息
NSArray<QCloudMultipartUploadPart*> *parts = result.parts;
}];
[[QCloudCOSXMLService defaultCOSXML] ListMultipart:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudListMultipartRequest.init();
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
req.object = "exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
// 在初始化分块上传的响应中，会返回一个唯一的描述符（upload ID）
if let uploadId = self.uploadId {
req.uploadId = uploadId;
}
req.setFinish { (result, error) in
if let result = result {
// 所有已完成的分片
let parts = result.parts
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().listMultipart(req);
```

说明：
更多完整示例，请前往 GitHub 查看。

完成分块上传

功能说明

完成整个文件的分块上传（Complete Multipart Upload）。

示例代码

Objective-C

```
QCloudCompleteMultipartUploadRequest *completeRequst = [QCloudCompleteMultipartUploadRequest new];
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
completeRequst.object = @"exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
completeRequst.bucket = @"examplebucket-1250000000";
// 本次要查询的分块上传的 uploadId，可从初始化分块上传的请求结果 QCloudInitiateMultipartUploadResult 中得到
completeRequst.uploadId = uploadId;
// 已上传分块的信息
QCloudCompleteMultipartUploadInfo *partInfo = [QCloudCompleteMultipartUploadInfo new];
NSMutableArray * parts = [self.parts mutableCopy];
// 对已上传的块进行排序
[parts sortUsingComparator:^NSComparisonResult(QCloudMultipartInfo* _Nonnull obj1,
QCloudMultipartInfo* _Nonnull obj2) {
int a = obj1.partNumber.intValue;
int b = obj2.partNumber.intValue;

if (a < b) {
return NSOrderedAscending;
} else {
return NSOrderedDescending;
}
}];
partInfo.parts = [parts copy];
completeRequst.parts = partInfo;
[completeRequst setFinishBlock:^(QCloudUploadObjectResult * _Nonnull result,
NSError * _Nonnull error) {
// 从 result 中获取上传结果
// 获取文件crc64
NSString * crc64 = [[outputObject __originHTTPURLResponse__].allHeaderFields valueForKey:@"x-cos-hash-crc64ecma"];
}];
[[QCloudCOSXMLService defaultCOSXML] CompleteMultipartUpload:completeRequst];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let complete = QCloudCompleteMultipartUploadRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
complete.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
complete.object = "exampleobject";
// 本次要查询的分块上传的 uploadId，可从初始化分块上传的请求结果
// QCloudInitiateMultipartUploadResult 中得到
complete.uploadId = "exampleUploadId";
if let uploadId = self.uploadId {
complete.uploadId = uploadId;
}
// 已上传分块的信息
let completeInfo = QCloudCompleteMultipartUploadInfo.init();
if self.parts == nil {
print("没有要完成的分块");
return;
}
if self.parts != nil {
completeInfo.parts = self.parts ?? [];
}
complete.parts = completeInfo;
complete.setFinish { (result, error) in
if let result = result {
// 文件的 eTag
let eTag = result.eTag
// 不带签名的文件链接
let location = result.location

// 获取文件crc64
let crc64 = result?.__originHTTPURLResponse__.allHeaderFields["x-cos-hash-crc64ecma"];
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().completeMultipartUpload(complete);
```

说明：
更多完整示例，请前往 GitHub 查看。

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块（Abort Multipart Upload）。

示例代码

Objective-C

```
QCloudAbortMultipfartUploadRequest *abortRequest = [QCloudAbortMultipfartUploadRequest new];
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
abortRequest.object = @"exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
abortRequest.bucket = @"examplebucket-1250000000";
// 本次要终止的分块上传的 uploadId
// 可从初始化分块上传的请求结果 QCloudInitiateMultipartUploadResult 中得到
abortRequest.uploadId = @"exampleUploadId";
[abortRequest setFinishBlock:^(id outputObject, NSError *error) {
// 可以从 outputObject 中获取 response 中 etag 或者自定义头部等信息
NSDictionary * result = (NSDictionary *)outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML]AbortMultipfartUpload:abortRequest];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let abort = QCloudAbortMultipfartUploadRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
abort.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
abort.object = "exampleobject";
// 本次要查询的分块上传的 uploadId，可从初始化分块上传的请求结果
// QCloudInitiateMultipartUploadResult 中得到
```

```
abort.uploadId = self.uploadId!;
abort.finishBlock = {(result,error)in
if let result = result {
// 可以从 result 中获取服务器返回的 header 信息
} else {
print(error!)
}
}
QCloudCOSXMLService.defaultCOSXML().abortMultipfartUpload(abort);
```

说明：
更多完整示例，请前往 GitHub 查看。

下载对象

最近更新时间：2022-01-10 12:55:12

简介

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

高级接口（推荐）

下载对象

高级接口支持暂停、恢复以及取消下载请求，同时支持断点下载功能。

示例代码一：下载单个对象

Objective-C

```
QCloudCOSXMLDownloadObjectRequest * request = [QCloudCOSXMLDownloadObjectRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = [NSURL fileURLWithPath:@"Local File Path"];
// 本地已下载的文件大小，如果是从头开始下载，请不要设置
request.localCacheDownloadOffset = 100;
// 监听下载结果
[request setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
```

```
}];
// 监听下载进度
[request setDownProcessBlock:^(int64_t bytesDownload,
int64_t totalBytesDownload,
int64_t totalBytesExpectedToDownload) {

// bytesDownload 新增字节数
// totalBytesDownload 本次下载接收的总字节数
// totalBytesExpectedToDownload 本次下载的目标字节数
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] DownloadObject:reques
t];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let request : QCloudCOSXMLDownloadObjectRequest = QCloudCOSXMLDownloadObjectReque
st();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
request.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = "exampleobject";
// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = NSURL.fileURL(withPath: "Local File Path") as URL?;
// 本地已下载的文件大小，如果是从头开始下载，请不要设置
request.localCacheDownloadOffset = 100;
// 监听下载进度
request.sendProcessBlock = { (bytesDownload, totalBytesDownload,
totalBytesExpectedToDownload) in

// bytesDownload 新增字节数
// totalBytesDownload 本次下载接收的总字节数
// totalBytesExpectedToDownload 本次下载的目标字节数
}
// 监听下载结果
request.finishBlock = { (result, error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
```

```
}
}
QCloudCOSTransferMangerService.defaultCOSTransferManager().downloadObject(reques
t);
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

示例代码二：下载暂停、继续与取消

Objective-C

对于下载任务，可以通过以下方式暂停：

```
[request cancel];
```

暂停之后，可以通过以下方式续传：

```
// 本地已下载的文件大小
int64_t localCacheDownloadOffset = 0;
request.localCacheDownloadOffset = localCacheDownloadOffset;
```

也通过以下方式取消下载：

```
[request cancel];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

对于下载任务，可以通过以下方式暂停：

```
request.cancel();
```

暂停之后，可以通过以下方式续传：

```
// 本地已下载的文件大小
let localCacheDownloadOffset = 100;
```

```
request.localCacheDownloadOffset = Int64(localCacheDownloadOffset);
```

也通过以下方式取消下载：

```
request.cancel();
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

示例代码三：断点下载

Objective-C

```
QCloudCOSXMLDownloadObjectRequest *getObjectRequest = [[QCloudCOSXMLDownloadObjectRequest alloc] init];
//支持断点下载，默认不支持
getObjectRequest.resumableDownload = true;
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
getObjectRequest.bucket = transferTestBucket.name;
// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
getObjectRequest.downloadingURL = [NSURL URLWithString:QCloudTempFilePathWithExtension(@"downding")];
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
getObjectRequest.object = put.object;
// 监听下载结果
[getObjectRequest setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];

// 监听下载进度
[getObjectRequest setDownProcessBlock:^(int64_t bytesDownload,
int64_t totalBytesDownload,
int64_t totalBytesExpectedToDownload) {

// bytesDownload 新增字节数
// totalBytesDownload 本次下载接收的总字节数
// totalBytesExpectedToDownload 本次下载的目标字节数
}];

[[QCloudCOSTransferMangerService costransfermangerServiceForKey:kHTTPServiceKey] DownloadObject:getObjectRequest];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let request : QCloudCOSXMLDownloadObjectRequest = QCloudCOSXMLDownloadObjectRequest();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
request.bucket = "examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = "exampleobject";

request.resumableDownload = true;
// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = NSURL.fileURL(withPath: "Local File Path") as URL?;

// 本地已下载的文件大小，如果是从头开始下载，请不要设置
request.localCacheDownloadOffset = 100;
// 监听下载进度
request.sendProcessBlock = { (bytesDownload, totalBytesDownload, totalBytesExpectedToDownload) in

// bytesDownload 新增字节数
// totalBytesDownload 本次下载接收的总字节数
// totalBytesExpectedToDownload 本次下载的目标字节数
}
// 监听下载结果
request.finishBlock = { (result, error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}

QCloudCOSTransferMangerService.defaultCOSTransferManager().downloadObject(request);
}
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码四：批量下载

Objective-C

```
for (int i = 0; i<20; i++) {
QCloudCOSXMLDownloadObjectRequest * request = [QCloudCOSXMLDownloadObjectRequest new];

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";

// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = [NSURL fileURLWithPath:@"Local File Path"];

// 本地已下载的文件大小，如果是从头开始下载，请不要设置
request.localCacheDownloadOffset = 100;

// 监听下载结果
[request setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];

// 监听下载进度
[request setDownProcessBlock:^(int64_t bytesDownload,
int64_t totalBytesDownload,
int64_t totalBytesExpectedToDownload) {

// bytesDownload 新增字节数
// totalBytesDownload 本次下载接收的总字节数
// totalBytesExpectedToDownload 本次下载的目标字节数
}];

[[QCloudCOSTransferMangerService defaultCOSTransferManager] DownloadObject:request];
}
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
for i in 1...10 {
let request : QCloudCOSXMLDownloadObjectRequest = QCloudCOSXMLDownloadObjectReque
st();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
request.bucket = "examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = "exampleobject";

// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = NSURL.fileURL(withPath: "Local File Path") as URL?;

// 本地已下载的文件大小，如果是从头开始下载，请不要设置
request.localCacheDownloadOffset = 100;
// 监听下载进度
request.sendProcessBlock = { (bytesDownload, totalBytesDownload,
totalBytesExpectedToDownload) in

// bytesDownload 新增字节数
// totalBytesDownload 本次下载接收的总字节数
// totalBytesExpectedToDownload 本次下载的目标字节数
}
// 监听下载结果
request.finishBlock = { (result, error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}

QCloudCOSTransferMangerService.defaultCOSTransferManager().downloadObject(reques
t);
}
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码五：下载文件夹及其文件

Objective-C

```
QCloudGetBucketRequest* request = [QCloudGetBucketRequest new];
// 存储桶名称，格式为 BucketName-APPID
request.bucket = @"examplebucket-1250000000";
// 单次返回的最大条目数量，默认1000
request.maxKeys = 100;
//要下载的文件夹在cos上的完整路径
request.prefix = @"cos_path";

[request setFinishBlock:^(QCloudListBucketResult * result, NSError* error) {
if(!error){
for (QCloudBucketContents *content in result.contents) {
QCloudCOSXMLDownloadObjectRequest * request = [QCloudCOSXMLDownloadObjectRequest
new];

// 存储桶名称，格式为 BucketName-APPID
request.bucket = @"examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "dir1/object1"
request.object = content.key;

// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = [NSURL fileURLWithPath:[@"Local File Path" stringByAppen
dingFormat:@"/%@",content.key]];

// 监听下载结果
[request setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];

// 监听下载进度
[request setDownProcessBlock:^(int64_t bytesDownload,
int64_t totalBytesDownload,
int64_t totalBytesExpectedToDownload) {

// bytesDownload 新增字节数
// totalBytesDownload 本次下载接收的总字节数
```

```
// totalBytesExpectedToDownload 本次下载的目标字节数
}];

[[QCloudCOSTransferMangerService defaultCOSTransferManager] DownloadObject:reques
t];
}


}
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucket:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getBucketReq = QCloudGetBucketRequest.init();

// 存储桶名称，格式为 BucketName-APPID
getBucketReq.bucket = "examplebucket-1250000000";

// 单次返回的最大条目数量，默认1000
getBucketReq.maxKeys = 100;

//要下载的文件夹在cos上的完整路径
getBucketReq.prefix = "cos_path";

getBucketReq.setFinish { (result, error) in
if let result = result {
let contents = result.contents;
for content in contents {
let info = QCloudDeleteObjectInfo.init();
let request : QCloudCOSXMLDownloadObjectRequest = QCloudCOSXMLDownloadObjectReque
st();

// 存储桶名称，格式为 BucketName-APPID
request.bucket = "examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "dir1/object1"
request.object = content.key;

// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
```

```
request.downloadingURL = NSURL.fileURL(withPath: "Local File Path" ) as URL?;

// 本地已下载的文件大小，如果是从头开始下载，请不要设置
request.localCacheDownloadOffset = 100;
// 监听下载进度
request.sendProcessBlock = { (bytesDownload, totalBytesDownload,
totalBytesExpectedToDownload) in

// bytesDownload 新增字节数
// totalBytesDownload 本次下载接收的总字节数
// totalBytesExpectedToDownload 本次下载的目标字节数
}
// 监听下载结果
request.finishBlock = { (result, error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}

QCloudCOSTransferMangerService.defaultCOSTransferManager().downloadObject(reques
t);
}
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getBucket(getBucketReq);
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码六：下载时限速

注意：
需要 COS iOS SDK v5.9.5 及以上版本。

Objective-C

```
QCloudCOSXMLDownloadObjectRequest * request = [QCloudCOSXMLDownloadObjectRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = [NSURL fileURLWithPath:@"Local File Path"];
// 本地已下载的文件大小，如果是从头开始下载，请不要设置
request.localCacheDownloadOffset = 100;
// 使用trafficLimit参数设置下载限速，单位为bit/s，限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s。
request.trafficLimit = 819200;
// 监听下载结果
[request setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];
// 监听下载进度
[request setDownProcessBlock:^(int64_t bytesDownload,
int64_t totalBytesDownload,
int64_t totalBytesExpectedToDownload) {

// bytesDownload 新增字节数
// totalBytesDownload 本次下载接收的总字节数
// totalBytesExpectedToDownload 本次下载的目标字节数
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] DownloadObject:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let request : QCloudCOSXMLDownloadObjectRequest = QCloudCOSXMLDownloadObjectRequest();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
request.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
```

```
request.object = "exampleobject";
// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = NSURL.fileURL(withPath: "Local File Path") as URL?;
// 本地已下载的文件大小，如果是从头开始下载，请不要设置
request.localCacheDownloadOffset = 100;
// 使用trafficLimit参数设置下载限速，单位为bit/s，限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s。
request.trafficLimit = 819200;
// 监听下载进度
request.sendProcessBlock = { (bytesDownload, totalBytesDownload,
totalBytesExpectedToDownload) in

// bytesDownload 新增字节数
// totalBytesDownload 本次下载接收的总字节数
// totalBytesExpectedToDownload 本次下载的目标字节数
}
// 监听下载结果
request.finishBlock = { (result, error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSTransferMangerService.defaultCOSTransferManager().downloadObject(reques
t);
```

说明：
更多完整示例，请前往 GitHub 查看。

简单操作

下载对象

功能说明

下载一个 Object（文件/对象）至本地（GET Object）。

示例代码一：下载对象

Objective-C

```
QCloudGetObjectRequest* request = [QCloudGetObjectRequest new];
// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = [NSURL URLWithString:QCloudTempFilePathWithExtension(@"downding")];
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
[request setFinishBlock:^(id outputObject, NSError *error) {
// 可以从 outputObject 中获取 response 中 etag 或者自定义头部等信息
NSDictionary* info = (NSDictionary *) outputObject;
// 服务端文件的crc64
result[@"x-cos-hash-crc64ecma"]
// 下载到本地的文件crc64 若返回的crc64与本地计算一致，则表示下载的文件与远端文件完整一致；
uint64_t localCrc64 = [本地文件data qcloud_crc64];
}];
[request setDownProcessBlock:^(int64_t bytesDownload, int64_t totalBytesDownload, int64_t totalBytesExpectedToDownload) {

// 下载过程中的进度
// bytesDownload 一次下载的字节数,
// totalBytesDownload 总过接受的字节数
// totalBytesExpectedToDownload 文件一共多少字节
}];
[[QCloudCOSXMLService defaultCOSXML] GetObject:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getObject = QCloudGetObjectRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
getObject.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
getObject.object = "exampleobject";
// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
getObject.downloadingURL = URL.init(string: NSTemporaryDirectory())!
.appendingPathComponent(getObject.object);
getObject.finishBlock = {(result,error) in
```

```
if let result = result {
// 服务端文件的crc64
result["x-cos-hash-crc64ecma"]
// 下载到本地的文件crc64 若返回的crc64与本地计算一致，则表示下载的文件与远端文件完整一致；
uint64_t localCrc64 = 本地文件data.qcloud_crc64();
} else {
print(error!);
}
};
getObject.downProcessBlock = {(bytesDownload, totalBytesDownload,
totalBytesExpectedToDownload) in
// bytesDownload 一次下载的字节数,
// totalBytesDownload 总过接受的字节数
// totalBytesExpectedToDownload 文件一共多少字节
}
QCloudCOSXMLService.defaultCOSXML().getObject(getObject);
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码二：下载时限速

注意：
需要 COS iOS SDK v5.9.5 及以上版本。

Objective-C

```
QCloudGetObjectRequest* request = [QCloudGetObjectRequest new];
// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = [NSURL URLWithString:QCloudTempFilePathWithExtension(@"d
ownding")];
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 使用trafficLimit参数设置下载限速，单位为bit/s，限速值设置范围为819200 - 838860800，即1
00KB/s - 100MB/s。
request.trafficLimit = 819200;
[request setFinishBlock:^(id outputObject, NSError *error) {
// 可以从 outputObject 中获取 response 中 etag 或者自定义头部等信息
```

```
NSDictionary* info = (NSDictionary *) outputObject;
// 获取文件crc64
NSString * crc64 = [[outputObject __originHTTPURLResponse__].allHeaderFields valu
eForKey:@"x-cos-hash-crc64ecma"];
}];
[request setDownProcessBlock:^(int64_t bytesDownload, int64_t totalBytesDownload,
int64_t totalBytesExpectedToDownload) {

// 下载过程中的进度
// bytesDownload 一次下载的字节数,
// totalBytesDownload 总过接受的字节数
// totalBytesExpectedToDownload 文件一共多少字节
}];
[[QCloudCOSXMLService defaultCOSXML] GetObject:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getObject = QCloudGetObjectRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
getObject.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
getObject.object = "exampleobject";
// 设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
getObject.downloadingURL = URL.init(string: NSTemporaryDirectory())!
.appendingPathComponent(getObject.object);

// 使用trafficLimit参数设置下载限速，单位为bit/s，限速值设置范围为819200 - 838860800，即1
00KB/s - 100MB/s。
getObject.trafficLimit = 819200;
getObject.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
// 获取文件crc64
let crc64 = result?.__originHTTPURLResponse__.allHeaderFields["x-cos-hash-crc64ec
ma"];
} else {
print(error!);
}
};
```

```
getObject.downProcessBlock = {(bytesDownload, totalBytesDownload,
totalBytesExpectedToDownload) in
// bytesDownload 一次下载的字节数,
// totalBytesDownload 总过接受的字节数
// totalBytesExpectedToDownload 文件一共多少字节
}
QCloudCOSXMLService.defaultCOSXML().getObject(getObject);
```

说明：
更多完整示例，请前往 GitHub 查看。

列出对象

最近更新时间：2022-05-25 15:25:02

简介

本文档提供关于列出对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象
GET Bucket Object Versions	查询对象及其历史版本列表	查询存储桶下的部分或者全部对象及其历史版本信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

示例代码一: 获取第一页数据

Objective-C

```
QCloudGetBucketRequest* request = [QCloudGetBucketRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 单次返回的最大条目数量，默认1000
request.maxKeys = 100;
// 前缀匹配，用来规定返回的文件前缀地址
request.prefix = @"dir1/";
[request setFinishBlock:^(QCloudListBucketResult * result, NSError* error) {
// result 返回具体信息
// QCloudListBucketResult.contents 桶内文件数组
```

```
// QCloudListBucketResult.commonPrefixes 桶内文件夹数组
if (result.isTruncated) {
// 表示数据被截断，需要拉取下一页数据
self->prevPageResult = result;
}
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucket:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getBucketReq = QCloudGetBucketRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
getBucketReq.bucket = "examplebucket-1250000000";
// 单次返回的最大条目数量，默认1000
getBucketReq.maxKeys = 100;
// 前缀匹配
getBucketReq.prefix = "dir/";
getBucketReq.setFinish { (result, error) in
if let result = result {
// 文件列表
let contents = result.contents

if (result.isTruncated) {
// 数据被截断，需要请求下一页数据
self.prevPageResult = result;
}
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getBucket(getBucketReq);
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码二：请求下一页数据

Objective-C

```
QCloudGetBucketRequest* request = [QCloudGetBucketRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// prevPageResult 是上一页的返回结果
// 分页参数 默认以UTF-8二进制顺序列出条目，所有列出条目从marker开始
request.marker = prevPageResult.nextMarker;
// 单次返回的最大条目数量，默认1000
request.maxKeys = 100;
[request setFinishBlock:^(QCloudListBucketResult * result, NSError* error) {
// result 返回具体信息
// QCloudListBucketResult.contents 桶内文件数组
// QCloudListBucketResult.commonPrefixes 桶内文件夹数组
if (result.isTruncated) {
// 表示数据被截断，需要拉取下一页数据
self->prevPageResult = result;
}
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucket:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getBucketReq = QCloudGetBucketRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
getBucketReq.bucket = "examplebucket-1250000000";
// 分页参数 默认以UTF-8二进制顺序列出条目，所有列出条目从marker开始
if let result = self.prevPageResult {
getBucketReq.marker = result.marker
}
// 单次返回的最大条目数量，默认1000
getBucketReq.maxKeys = 100;
// 前缀匹配
getBucketReq.prefix = "dir/";
getBucketReq.setFinish { (result, error) in
if let result = result {
// 文件列表
let contents = result.contents
```

```
if (result.isTruncated) {
// 数据被截断，需要请求下一页数据
self.prevPageResult = result;
}
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getBucket(getBucketReq);
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码三：获取对象列表与子目录

Objective-C

```
QCloudGetBucketRequest* request = [QCloudGetBucketRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 单次返回的最大条目数量，默认1000
request.maxKeys = 100;
// 前缀匹配，用来规定返回的文件前缀地址
request.prefix = @"dir1/";
// 定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，
// 定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始
// delimiter:路径分隔符 固定为 /
request.delimiter = @"/";
// prevPageResult 是上一页的返回结果
// 分页参数 默认以UTF-8二进制顺序列出条目，所有列出条目从marker开始
request.marker = prevPageResult.nextMarker;
[request setFinishBlock:^(QCloudListBucketResult * result, NSError* error) {
// result 返回具体信息
// QCloudListBucketResult.contents 桶内文件数组
// QCloudListBucketResult.commonPrefixes 桶内文件夹数组
if (result.isTruncated) {
// 表示数据被截断，需要拉取下一页数据
self->prevPageResult = result;
}
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucket:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getBucketReq = QCloudGetBucketRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
getBucketReq.bucket = "examplebucket-1250000000";
// 单次返回的最大条目数量，默认1000
getBucketReq.maxKeys = 100;
// 前缀匹配，用来规定返回的文件前缀地址
getBucketReq.prefix = "dir/";
// 定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，
// 定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始
// delimiter:路径分隔符 固定为 /
getBucketReq.delimiter = "/";
// 分页参数 默认以UTF-8二进制顺序列出条目，所有列出条目从marker开始
if let result = self.prevPageResult {
getBucketReq.marker = result.marker
}
getBucketReq.setFinish { (result, error) in
if let result = result {
// 文件列表
let contents = result.contents

if (result.isTruncated) {
// 数据被截断，需要请求下一页数据
self.prevPageResult = result;
}
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getBucket(getBucketReq);
```

说明：
更多完整示例，请前往 GitHub 查看。

查询对象历史版本列表

功能说明

查询开启版本控制的存储桶下的部分或者全部对象。

示例代码：获取对象历史版本列表第一页数据

```
QCloudListObjectVersionsRequest* listObjectVersionsRequest = [[QCloudListObjectVersionsRequest alloc] init];
// 存储桶名称
listObjectVersionsRequest.bucket = @"bucketname";
// 一页请求数据条目数，默认 1000
listObjectVersionsRequest.maxKeys = 100;
//从当前key列出剩余的条目
listObjectVersionsRequest.keyMarker = prevPageResult.nextKeyMarker;
//从当前key的某个版本列出剩余的条目
listObjectVersionsRequest.versionIdMarker = prevPageResult.nextVersionIDMarkder;
[listObjectVersionsRequest setFinishBlock:^(QCloudListVersionsResult * _Nonnull result,
NSError * _Nonnull error) {

// 已删除的文件
NSArray<QCloudDeleteMarker*> *deleteMarker = result.deleteMarker;

// 对象版本条目
NSArray<QCloudVersionContent*> *versionContent = result.versionContent;

if (result.isTruncated) {
// 表示数据被截断，需要拉取下一页数据
self->prevPageResult = result;
}

}];
[[QCloudCOSXMLService defaultCOSXML] ListObjectVersions:listObjectVersionsRequest];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

复制与移动对象

最近更新时间：2022-05-25 15:25:02

简介

本文档提供关于复制、移动对象的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object - Copy	设置对象复制（修改对象属性）	复制文件到目标路径
DELETE Object	删除单个对象	在存储桶中删除指定对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传/复制	查询正在进行中的分块上传/复制信息
Initiate Multipart Upload	初始化分块上传/复制	初始化分块上传/复制操作
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传/复制块	查询特定分块操作中的已上传/复制的块
Complete Multipart Upload	完成分块上传/复制	完成整个文件的分块上传/复制
Abort Multipart Upload	终止分块上传/复制	终止一个分块操作并删除已上传/复制的块

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

高级接口（推荐）

复制对象

高级接口封装了简单复制、分块复制接口的异步请求，并支持暂停、恢复以及取消复制请求。

版权所有：腾讯云计算（北京）有限责任公司

示例代码

Objective-C

```
QCloudCOSXMLCopyObjectRequest* request = [[QCloudCOSXMLCopyObjectRequest alloc] init];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 文件来源存储桶，需要是公有读或者在当前账号有权限
request.sourceBucket = @"sourcebucket-1250000000";
// 源文件名称
request.sourceObject = @"sourceObject";
// 源文件的 APPID
request.sourceAPPID = @"1250000000";
// 来源的地域
request.sourceRegion= @"COS_REGION";
[request setFinishBlock:^(QCloudCopyObjectResult* result, NSError* error) {
// 可以从 outputObject 中获取 response 中 etag 或者自定义头部等信息
}];
// 注意如果是跨地域复制，这里使用的 transferManager 所在的 region 必须为目标桶所在的 region
[[QCloudCOSTransferMangerService defaultCOSTransferManager] CopyObject:request];
// 取消copy
// 若需要取消copy 调用cancel方法
[request cancel];
```

说明：

更多完整示例，请前往 GitHub 查看。

Swift

```
let copyRequest = QCloudCOSXMLCopyObjectRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
copyRequest.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
copyRequest.object = "exampleobject";
// 文件来源存储桶，需要是公有读或者在当前账号有权限
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
```

```
copyRequest.sourceBucket = "sourcebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
copyRequest.sourceObject = "sourceObject";
// 源文件的 APPID
copyRequest.sourceAPPID = "1250000000";
// 来源的地域
copyRequest.sourceRegion = "COS_REGION";
copyRequest.setFinish { (copyResult, error) in
if let copyResult = copyResult {
// 文件的 etag
let eTag = copyResult.eTag
} else {
print(error!);
}

}
// 注意如果是跨地域复制，这里使用的 transferManager 所在的 region 必须为目标桶所在的 regio
n
QCloudCOSTransferMangerService.defaultCOSTransferManager().copyObject(copyReques
t);
// 取消copy
// 若需要取消copy 调用cancel方法
copyRequest.cancel();
```

说明：
更多完整示例，请前往 GitHub 查看。

移动对象

移动对象主要包括两个操作：复制源对象到目标位置，删除源对象。

由于 COS 通过存储桶名称（Bucket）和对象键（ObjectKey）来标识对象。移动对象也就意味着修改这个对象的标识，COS Java SDK 目前没有提供修改对象唯一标识名的单独接口，但是可以通过组合**复制对象**加上**删除对象**的基本操作，来达到修改对象标识的目的，从而实现移动对象。

例如将 mybucket-1250000000 这个存储桶中的 picture.jpg 这个对象移动到同个存储桶的 doc 路径下。首先可以复制 picture.jpg 对象到存储桶的 doc 路径下，即对象键设定为 doc/picture.jpg，复制完成后删除 picture.jpg 这个对象，来实现“移动”的效果。

同样的，如果想将 mybucket-1250000000 这个存储桶里的 picture.jpg 这个对象移动到 myanothorbucket-1250000000 这个存储桶里，可以先将对象复制到 myanothorbucket-1250000000 存储桶，然后删除原来的对象。

示例代码

Objective-C

```
QCloudCOSXMLCopyObjectRequest* request = [[QCloudCOSXMLCopyObjectRequest alloc] i
nit];

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "dir1/object1"
request.object = @"exampleobject";

// 文件来源存储桶，需要是公有读或者在当前账号有权限
request.sourceBucket = @"sourcebucket-1250000000";

// 源文件名称
request.sourceObject = @"sourceObject";

// 源文件的 APPID
request.sourceAPPID = @"1250000000";

// 来源的地域
request.sourceRegion= @"COS_REGION";

[request setFinishBlock:^(QCloudCopyObjectResult* result, NSError* error) {
// 可以从 outputObject 中获取 response 中 etag 或者自定义头部等信息
if(!error){
QCloudDeleteObjectRequest* deleteObjectRequest = [QCloudDeleteObjectRequest new];

// 文件来源存储桶，需要是公有读或者在当前账号有权限
deleteObjectRequest.bucket = @"sourcebucket-1250000000";

// 源文件名称，是源对象在 COS 上的完整路径，如果带目录的话，格式为 "dir1/object1"
deleteObjectRequest.object = @"sourceObject";

[deleteObjectRequest setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];

[[QCloudCOSXMLService defaultCOSXML] DeleteObject:deleteObjectRequest];
}
}];

// 注意如果是跨地域移动，这里使用的 transferManager 所在的 region 必须为目标桶所在的 regio
```

```
n
[[QCloudCOSTransferMangerService defaultCOSTransferManager] CopyObject:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let copyRequest = QCloudCOSXMLCopyObjectRequest.init();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
copyRequest.bucket = "examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "dir1/object1"
copyRequest.object = "exampleobject";

// 文件来源存储桶，需要是公有读或者在当前账号有权限
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
copyRequest.sourceBucket = "sourcebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "dir1/object1"
copyRequest.sourceObject = "sourceObject";

// 源文件的 APPID
copyRequest.sourceAPPID = "1250000000";

// 来源的地域
copyRequest.sourceRegion = "COS_REGION";

copyRequest.setFinish { (copyResult, error) in
if let copyResult = copyResult {
// 文件的 etag
let deleteObject = QCloudDeleteObjectRequest.init();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
deleteObject.bucket = "sourcebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "dir1/object1"
deleteObject.object = "sourceObject";
```

```
deleteObject.finishBlock = {(result,error)in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().deleteObject(deleteObject);
} else {
print(error!);
}

}
// 注意如果是跨地域移动，这里使用的 transferManager 所在的 region 必须为目标桶所在的 regio
n
QCloudCOSTransferMangerService.defaultCOSTransferManager().copyObject(copyReques
t);
```

说明：
更多完整示例，请前往 GitHub 查看。

简单操作

复制对象（修改属性）

复制文件到目标路径（PUT Object-Copy）。

示例代码一: 复制对象时保留对象属性

Objective-C

```
QCloudPutObjectCopyRequest* request = [[QCloudPutObjectCopyRequest alloc] init];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 是否拷贝元数据，枚举值：Copy，Replaced，默认值 Copy。
// 假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制
// 假如标记为 Replaced，按 Header 信息修改元数据。当目标路径和原路径一致，即用户试图修改元数
据时，必须为 Replaced
request.metadataDirective = @"Copy";
```

```
// 定义 Object 的 ACL 属性，有效值：private，public-read，default。
// 默认值：default（继承 Bucket 权限）。
// 注意：如果您无需进行 Object ACL 控制，请填 default
// 或者此项不进行设置，默认继承 Bucket 权限。
request.accessControlList = @"default";
// 源对象所在的路径
request.objectCopySource =
@"sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject";
// 指定源文件的 versionID，只有开启或开启后暂停的存储桶，才会响应此参数
request.versionID = @"objectVersion1";
[request setFinishBlock:^(QCloudCopyObjectResult * _Nonnull result,
NSError * _Nonnull error) {
// result 返回具体信息
}];
[[QCloudCOSXMLService defaultCOSXML] PutObjectCopy:request];
```

说明：

更多完整示例，请前往 GitHub 查看。

Swift

```
let putObjectCopy = QCloudPutObjectCopyRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
putObjectCopy.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
putObjectCopy.object = "exampleobject";
// 源对象所在的路径
putObjectCopy.objectCopySource = "sourcebucket-1250000000.cos.ap-guangzhou.myqclo
ud.com/sourceObject";
// 是否拷贝元数据，枚举值：Copy，Replaced，默认值 Copy。
// 假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制
// 假如标记为 Replaced，按 Header 信息修改元数据。当目标路径和原路径一致，即用户试图修改元数
据时，必须为 Replaced
putObjectCopy.metadataDirective = "Copy";
// 定义 Object 的 ACL 属性，有效值：private，public-read，default。
// 默认值：default（继承 Bucket 权限）。
// 注意：如果您无需进行 Object ACL 控制，请填 default
// 或者此项不进行设置，默认继承 Bucket 权限。
putObjectCopy.accessControlList = "default";
// 指定源文件的 versionID，只有开启或开启后暂停的存储桶，才会响应此参数
putObjectCopy.versionID = "versionID";
putObjectCopy.setFinish { (result, error) in
```

```
if let result = result {
let eTag = result.eTag
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putObjectCopy(putObjectCopy);
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码二: 复制对象时替换对象属性

Objective-C

```
QCloudPutObjectCopyRequest* request = [[QCloudPutObjectCopyRequest alloc] init];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 是否拷贝元数据，枚举值：Copy，Replaced，默认值 Copy。
// 假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制
// 假如标记为 Replaced，按 Header 信息修改元数据。当目标路径和原路径一致，即用户试图修改元数据时，必须为 Replaced
request.metadataDirective = @"Replaced";
// 修改元数据
[request.customHeaders setValue:@"newValue" forKey:@"x-cos-meta-*"];
// 对象存储类型，枚举值请参见 存储类型概述 文档，
// STANDARD_IA，ARCHIVE。仅当对象不是标准存储（STANDARD）时才会返回该头部
// 修改存储类型
[request.customHeaders setValue:@"newValue" forKey:@"x-cos-storage-class"];
// 定义 Object 的 ACL 属性，有效值：private，public-read，default。
// 默认值：default（继承 Bucket 权限）。
// 注意：如果您无需进行 Object ACL 控制，请填 default
// 或者此项不进行设置，默认继承 Bucket 权限。
// 修改acl
request.accessControlList = @"private";
// 源对象所在的路径
request.objectCopySource =
@"sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject";
// 指定源文件的 versionID，只有开启或开启后暂停的存储桶，才会响应此参数
request.versionID = @"objectVersion1";
[request setFinishBlock:^(QCloudCopyObjectResult * _Nonnull result,
```

```
NSError * _Nonnull error) {
// result 返回具体信息

}];
[[QCloudCOSXMLService defaultCOSXML] PutObjectCopy:request];
```

说明：

更多完整示例，请前往 GitHub 查看。

Swift

```
let request : QCloudPutObjectCopyRequest = QCloudPutObjectCopyRequest();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
request.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = "exampleobject";
// 是否拷贝元数据，枚举值：Copy，Replaced，默认值 Copy。
// 假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制
// 假如标记为 Replaced，按 Header 信息修改元数据。当目标路径和原路径一致，即用户试图修改元数据时，必须为 Replaced
request.metadataDirective = "Replaced";
// 修改元数据
request.customHeaders.setValue("newValue", forKey: "x-cos-meta-*");
// 对象存储类型，枚举值请参见 存储类型概述 文档，
// STANDARD_IA，ARCHIVE。仅当对象不是标准存储（STANDARD）时才会返回该头部
// 修改存储类型
request.customHeaders.setValue("newValue", forKey: "x-cos-storage-class");
// 定义 Object 的 ACL 属性，有效值：private，public-read，default。
// 默认值：default（继承 Bucket 权限）。
// 注意：如果您无需进行 Object ACL 控制，请填 default
// 或者此项不进行设置，默认继承 Bucket 权限。
// 修改acl
request.accessControlList = "源文件acl";
// 源对象所在的路径
request.objectCopySource = "sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject";
// 指定源文件的 versionID，只有开启或开启后暂停的存储桶，才会响应此参数
request.versionID = "versionID";
request.setFinish { (result, error) in
if let result = result {
let eTag = result.eTag
} else {
```

```
print(error!);
}

}
QCloudCOSXMLService.defaultCOSXML().putObjectCopy(request);
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码三: 修改对象元数据

Objective-C

```
QCloudPutObjectCopyRequest* request = [[QCloudPutObjectCopyRequest alloc] init];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 是否拷贝元数据，枚举值：Copy，Replaced，默认值 Copy。
// 假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制
// 假如标记为 Replaced，按 Header 信息修改元数据。当目标路径和原路径一致，即用户试图修改元数据时，必须为 Replaced
request.metadataDirective = @"Replaced";
// 自定义对象header
[request.customHeaders setValue:@"newValue" forKey:@"x-cos-meta-*"];
// 定义 Object 的 ACL 属性，有效值：private，public-read，default。
// 默认值：default（继承 Bucket 权限）。
// 注意：如果您无需进行 Object ACL 控制，请填 default
// 或者此项不进行设置，默认继承 Bucket 权限。
request.accessControlList = @"default";
// 源对象所在的路径
request.objectCopySource =
@"examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobject";
[request setFinishBlock:^(QCloudCopyObjectResult * _Nonnull result,
NSError * _Nonnull error) {
// result 返回具体信息

}];
[[QCloudCOSXMLService defaultCOSXML] PutObjectCopy:request];
```

说明：

更多完整示例，请前往 GitHub 查看。

Swift

```
let request : QCloudPutObjectCopyRequest = QCloudPutObjectCopyRequest();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
request.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = "exampleobject";
// 是否拷贝元数据，枚举值：Copy，Replaced，默认值 Copy。
// 假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制
// 假如标记为 Replaced，按 Header 信息修改元数据。当目标路径和原路径一致
// 即用户试图修改元数据时，必须为 Replaced
request.metadataDirective = "Replaced";
// 自定义对象header
request.customHeaders.setValue("newValue", forKey: "x-cos-meta-*")
// 定义 Object 的 ACL 属性，有效值：private，public-read，default。
// 默认值：default（继承 Bucket 权限）。
// 注意：如果您无需进行 Object ACL 控制，请填 default
// 或者此项不进行设置，默认继承 Bucket 权限。
request.accessControlList = "default";
// 源对象所在的路径
request.objectCopySource =
"examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobject";
request.setFinish { (result, error) in
if let result = result {
// 生成的新文件的 etag
let eTag = result.eTag
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putObjectCopy(request);
```

说明：

更多完整示例，请前往 GitHub 查看。

示例代码四: 修改对象存储类型

Objective-C

```
QCloudPutObjectCopyRequest* request = [[QCloudPutObjectCopyRequest alloc] init];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 对象存储类型，枚举值请参见 存储类型概述 文档，
// STANDARD_IA，ARCHIVE。仅当对象不是标准存储（STANDARD）时才会返回该头部
[request.customHeaders setValue:@"ARCHIVE" forKey:@"x-cos-storage-class"];
// 源对象所在的路径
request.objectCopySource =
@"examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobject";
// 指定源文件的 versionID，只有开启或开启后暂停的存储桶，才会响应此参数
request.versionID = @"";
[request setFinishBlock:^(QCloudCopyObjectResult * _Nonnull result,
NSError * _Nonnull error) {
// result 返回具体信息

}];
[[QCloudCOSXMLService defaultCOSXML] PutObjectCopy:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let request : QCloudPutObjectCopyRequest = QCloudPutObjectCopyRequest();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
request.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = "exampleobject";
// 对象存储类型，枚举值请参见 存储类型概述 文档，
// STANDARD_IA，ARCHIVE。仅当对象不是标准存储（STANDARD）时才会返回该头部
request.customHeaders.setValue("newValue", forKey: "x-cos-storage-class");
// 源对象所在的路径
request.objectCopySource =
"examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobject";
request.setFinish { (result, error) in
if let result = result {
// 生成的新文件的 etag
let eTag = result.eTag
```

```
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putObjectCopy(request);
```

说明：
更多完整示例，请前往 GitHub 查看。

分块操作

这里主要介绍分块复制的流程。分块复制和分块上传在涉及接口及使用上具有一定重合度。

分块复制的流程

1. 初始化分块复制（Initiate Multipart Upload），得到 UploadId。
2. 使用 UploadId 复制分块（Upload Part - Copy）。
3. 完成分块复制（Complete Multipart Upload）。

分块复制的流程

1. 如果没有记录 UploadId，查询分块复制任务（List Multipart Uploads），得到对应文件的 UploadId。
2. 使用 UploadId 列出已复制的分块（List Parts）。
3. 使用 UploadId 复制剩余的分块（Upload Part Copy）。
4. 完成分块复制（Complete Multipart Upload）。

终止分块复制的流程

1. 如果没有记录 UploadId，查询分块复制任务（List Multipart Upload），得到对应文件的 UploadId
2. 终止分块复制并删除已复制分块（Abort Multipart Upload）

查询分块复制

功能说明

查询指定存储桶中正在进行的分块复制（List Multipart Uploads），该接口与分块上传的查询分块上传接口一致。

示例代码

Objective-C

```
QCloudListBucketMultipartUploadsRequest* uploads = [QCloudListBucketMultipartUploadsRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
uploads.bucket = @"examplebucket-1250000000";
// 设置最大返回的 multipart 数量，合法取值从 1 到 1000
uploads.maxUploads = 100;
[uploads setFinishBlock:^(QCloudListMultipartUploadsResult* result,
NSError *error) {
// 可以从 result 中返回分块信息
// 进行中的分块复制对象
NSArray<QCloudListMultipartUploadContent*> *uploads = result.uploads;
}];
[[QCloudCOSXMLService defaultCOSXML] ListBucketMultipartUploads:uploads];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let listParts = QCloudListBucketMultipartUploadsRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
listParts.bucket = "examplebucket-1250000000";
// 设置最大返回的 multipart 数量，合法取值从 1 到 1000
listParts.maxUploads = 100;
listParts.setFinish { (result, error) in
if let result = result {
// 未完成的所有分块复制任务
let uploads = result.uploads;
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().listBucketMultipartUploads(listParts);
```

说明：
更多完整示例，请前往 GitHub 查看。

初始化分块复制

功能说明

初始化 Multipart Upload 复制操作，获取对应的 uploadId（Initiate Multipart Upload），该接口与分块上传的初始化分块上传接口一致。

示例代码

Objective-C

```
QCloudInitiateMultipartUploadRequest* initRequest = [QCloudInitiateMultipartUploadRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
initRequest.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
initRequest.object = @"exampleobject";
// 将作为对象的元数据返回
initRequest.cacheControl = @"cacheControl";
initRequest.contentDisposition = @"contentDisposition";
// 定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private
initRequest.accessControlList = @"public";
// 赋予被授权者读的权限。
initRequest.grantRead = @"grantRead";
// 赋予被授权者全部权限
initRequest.grantFullControl = @"grantFullControl";
[initRequest setFinishBlock:^(QCloudInitiateMultipartUploadResult* outputObject,
NSError *error) {
// 获取分块复制的 uploadId，后续的分块复制都需要这个 ID，请保存以备后续使用
self->uploadId = outputObject.uploadId;

}];
[[QCloudCOSXMLService defaultCOSXML] InitiateMultipartUpload:initRequest];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let initRequest = QCloudInitiateMultipartUploadRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
```

```
tencent.com/cos5/bucket
initRequest.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
initRequest.object = "exampleobject";
initRequest.setFinish { (result, error) in
if let result = result {
// 获取分块复制的 uploadId，后续的分块复制都需要这个 ID，请保存以备后续使用
self.uploadId = result.uploadId;
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().initiateMultipartUpload(initRequest);
```

说明：
更多完整示例，请前往 GitHub 查看。

复制分块

功能说明

将其他对象复制为一个分块 （Upload Part-Copy）。

示例代码

Objective-C

```
QCloudUploadPartCopyRequest* request = [[QCloudUploadPartCopyRequest alloc] ini
t];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 源文件 URL 路径，可以通过 versionid 子资源指定历史版本
request.source = @"sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceOb
ject";
// 在初始化分块复制的响应中，会返回一个唯一的描述符（upload ID）
request.uploadID = uploadId;
// 标志当前分块的序号
request.partNumber = 1;
[request setFinishBlock:^(QCloudCopyObjectResult* result, NSError* error) {
QCloudMultipartInfo *part = [QCloudMultipartInfo new];
```

```
// 获取所复制分块的 etag
part.eTag = result.eTag;
part.partNumber = @"1";
// 保存起来用于最后完成复制时使用
self.parts=@[part];
}];
[[QCloudCOSXMLService defaultCOSXML]UploadPartCopy:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudUploadPartCopyRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
req.object = "exampleobject";
// 源文件 URL 路径，可以通过 versionid 子资源指定历史版本
req.source = "sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject"
;
// 在初始化分块复制的响应中，会返回一个唯一的描述符（upload ID）
if let uploadId = self.uploadId {
req.uploadID = uploadId;
}
// 标志当前分块的序号
req.partNumber = 1;
req.setFinish { (result, error) in
if let result = result {
let mutipartInfo = QCloudMultipartInfo.init();
// 获取所复制分块的 etag
mutipartInfo.eTag = result.eTag;
mutipartInfo.partNumber = "1";
// 保存起来用于最后完成复制时使用
self.parts = [mutipartInfo];
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().uploadPartCopy(req);
```

说明：
更多完整示例，请前往 GitHub 查看。

查询已复制的分块

功能说明

查询特定分块复制操作中的已复制的块（List Parts），该接口与分块上传的查询已上传的分块接口一致。

示例代码

Objective-C

```
QCloudListMultipartRequest* request = [QCloudListMultipartRequest new];
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 在初始化分块复制的响应中，会返回一个唯一的描述符（upload ID）
request.uploadId = uploadId;
[request setFinishBlock:^(QCloudListPartsResult * _Nonnull result,
NSError * _Nonnull error) {

// 从 result 中获取已复制分块信息
// 用来表示每一个块的信息
NSArray<QCloudMultipartUploadPart*> *parts = result.parts;
}];
[[QCloudCOSXMLService defaultCOSXML] ListMultipart:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudListMultipartRequest.init();
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
req.object = "exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
// 在初始化分块复制的响应中，会返回一个唯一的描述符（upload ID）
```

```
if let uploadId = self.uploadId {
req.uploadId = uploadId;
}
req.setFinish { (result, error) in
if let result = result {
// 所有已完成的分片
let parts = result.parts
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().listMultipart(req);
```

说明：
更多完整示例，请前往 GitHub 查看。

完成分块复制

功能说明

完成整个文件的分块复制（Complete Multipart Upload），该接口与分块上传的完成分块上传接口一致。

示例代码

Objective-C

```
QCloudCompleteMultipartUploadRequest *completeRequst = [QCloudCompleteMultipartUploadRequest new];
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
completeRequst.object = @"exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
completeRequst.bucket = @"examplebucket-1250000000";
// 本次要查询的分块复制的 uploadId，可从初始化分块复制的请求结果 QCloudInitiateMultipartUploadResult 中得到
completeRequst.uploadId = uploadId;
// 已复制分块的信息
QCloudCompleteMultipartUploadInfo *partInfo = [QCloudCompleteMultipartUploadInfo new];
NSMutableArray * parts = [self.parts mutableCopy];
// 对已复制的块进行排序
[parts sortUsingComparator:^NSComparisonResult(QCloudMultipartInfo* _Nonnull obj1,
QCloudMultipartInfo* _Nonnull obj2) {
```

```
int a = obj1.partNumber.intValue;
int b = obj2.partNumber.intValue;

if (a < b) {
return NSOrderedAscending;
} else {
return NSOrderedDescending;
}
}];
partInfo.parts = [parts copy];
completeRequst.parts = partInfo;
[completeRequst setFinishBlock:^(QCloudUploadObjectResult * _Nonnull result,
NSError * _Nonnull error) {
// 从 result 中获取复制结果
}];
[[QCloudCOSXMLService defaultCOSXML] CompleteMultipartUpload:completeRequst];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let complete = QCloudCompleteMultipartUploadRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
complete.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
complete.object = "exampleobject";
// 本次要查询的分块复制的 uploadId，可从初始化分块复制的请求结果
// QCloudInitiateMultipartUploadResult 中得到
complete.uploadId = "exampleUploadId";
if let uploadId = self.uploadId {
complete.uploadId = uploadId;
}
// 已复制分块的信息
let completeInfo = QCloudCompleteMultipartUploadInfo.init();
if self.parts == nil {
print("没有要完成的分块");
return;
}
if self.parts != nil {
completeInfo.parts = self.parts ?? [];
}
```

```
complete.parts = completeInfo;
complete.setFinish { (result, error) in
// 从 result 中获取复制结果
}
QCloudCOSXMLService.defaultCOSXML().completeMultipartUpload(complete);
```

说明：
更多完整示例，请前往 GitHub 查看。

终止分块复制

功能说明

终止一个分块复制操作并删除已复制的块（Abort Multipart Upload），该接口与分块上传的终止分块上传接口一致。

示例代码

Objective-C

```
QCloudAbortMultipfartUploadRequest *abortRequest = [QCloudAbortMultipfartUploadRequest new];
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
abortRequest.object = @"exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
abortRequest.bucket = @"examplebucket-1250000000";
// 本次要终止的分块复制的 uploadId
// 可从初始化分块复制的请求结果 QCloudInitiateMultipartUploadResult 中得到
abortRequest.uploadId = @"exampleUploadId";
[abortRequest setFinishBlock:^(id outputObject, NSError *error) {
// 可以从 outputObject 中获取 response 中 etag 或者自定义头部等信息
NSDictionary * result = (NSDictionary *)outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML]AbortMultipfartUpload:abortRequest];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let abort = QCloudAbortMultipfartUploadRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
abort.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
abort.object = "exampleobject";
// 本次要查询的分块复制的 uploadId，可从初始化分块复制的请求结果
// QCloudInitiateMultipartUploadResult 中得到
abort.uploadId = self.uploadId!;
abort.finishBlock = {(result,error)in
if let result = result {
// 可以从 result 中获取服务器返回的 header 信息
} else {
print(error!)
}
}
QCloudCOSXMLService.defaultCOSXML().abortMultipfartUpload(abort);
```

说明：

更多完整示例，请前往 GitHub 查看。

检索对象内容

最近更新时间：2022-01-10 12:43:39

简介

本文档提供关于检索对象内容操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
SELECT Object Content	检索对象内容	从指定对象（CSV 格式或者 JSON 格式）中检索内容

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

检索对象内容

功能说明

COS Select 支持检索以下格式的对象数据：

- CSV 格式：对象以 CSV 格式存储，并以固定的分隔符划分。
- JSON 格式：对象以 JSON 格式存储，可以是 JSON 文件或者 JSON 列表。

> 注意：
>
> - 使用 COS Select，您必须具有 `cos:GetObject` 的授权。
> - CSV、JSON 对象需要以 UTF-8 格式编码。
> - COS Select 支持检索 GZIP 或者 BZIP2 压缩的 CSV、JSON 对象。
> - COS Select 支持检索 SSE-COS 加密的 CSV、JSON 对象。

示例代码

```
QCloudSelectObjectContentRequest *request = [QCloudSelectObjectContentRequest ne
w];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
```

版权所有：腾讯云计算（北京）有限责任公司

```
request.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "dir1/object1"
request.object = @"exampleobject";
request.selectType = @"2";
// 选择文件 配置
QCloudSelectObjectContentConfig *config= [QCloudSelectObjectContentConfig new];
/**SQL 表达式，代表您需要发起的检索操作。例如SELECT s._1 FROM COSObject s。
这个表达式可以从 CSV 格式的对象中检索第一列内容。有关 SQL 表达式的详细介绍，
请参见 (Select)[https://cloud.tencent.com/document/product/436/37636] 命令
*/
config.express = @"select * from COSObject";
/**
表达式类型，该项为扩展项，目前只支持 SQL 表达式，仅支持 SQL 参数
*/
config.expressionType = QCloudExpressionTypeSQL;
/**
描述待检索对象的格式
*/
QCloudInputSerialization *inputS = [QCloudInputSerialization new];
inputS.compressionType = QCloudCOSXMLCompressionTypeNONE;
/**
描述在JSON对象格式下所需的文件参数。
*/
QCloudSerializationJSON *inputJson = [QCloudSerializationJSON new];
/**
SON 文件的类型：
DOCUMENT 表示 JSON 文件仅包含一个独立的 JSON 对象，且该对象可以被切割成多行
LINES 表示 JSON 对象中的每一行包含了一个独立的 JSON 对象
合法值为 DOCUMENT 、LINES
*/
inputJson.type = QCloudInputJSONFileTypeDocument;
inputS.serializationJSON = inputJson;
config.inputSerialization = inputS;
/**
描述检索结果的输出格式
*/
QCloudOutputSerialization *outputS = [QCloudOutputSerialization new];
QCloudSerializationJSON *outputJson = [QCloudSerializationJSON new];
/**
将输出结果中的记录分隔为不同行的字符，默认通过\n进行分隔。您可以指定任意8进制字符，
如逗号、分号、Tab 等。该参数最多支持2个字节，即您可以输入\r\n这类格式的分隔符。默认值为\n
*/
outputJson.outputRecordDelimiter = @"\n";
/**
描述在JSON对象格式下所需的文件参数。
*/
outputS.serializationJSON = outputJson;
```

```
config.outputSerialization = outputS;
/**
是否需要返回查询进度 QueryProgress 信息，如果选中 COS Select 将周期性返回查询进度
*/
QCloudRequestProgress *requestProgress = [QCloudRequestProgress new];
requestProgress.enabled = @"FALSE";
config.requestProgress =requestProgress;
request.selectObjectContentConfig = config;
/**
文件存储在本地的路径
*/
request.downloadingURL = [NSURL fileURLWithPath:QCloudFileInSubPath(@"test", @"2.
json")];
[request setDownProcessBlock:^(int64_t bytesDownload, int64_t totalBytesDownload,
int64_t totalBytesExpectedToDownload) {
NSLog(@"    DOWN [Total]%lld [Downloaded]%lld [Download]%lld", totalBytesExpected
ToDownload, totalBytesDownload, bytesDownload);
}];
[request setFinishBlock:^(id _Nonnull result, NSError * _Nonnull error) {
NSLog(@"result = %@",result);
}];
[[QCloudCOSXMLService defaultCOSXML] SelectObjectContent:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let request = QCloudSelectObjectContentRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
request.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "dir1/object1"
request.object = "exampleobject";
//代表这一接口的版本信息
request.selectType = "2";
// 选择文件 配置
let config = QCloudSelectObjectContentConfig();
/**SQL 表达式，代表您需要发起的检索操作。例如SELECT s._1 FROM COSObject s。
这个表达式可以从 CSV 格式的对象中检索第一列内容。有关 SQL 表达式的详细介绍，
请参见 (Select)[https://cloud.tencent.com/document/product/436/37636] 命令
*/
config.express = "Select * from COSObject";
```

```
/**
表达式类型，该项为扩展项，目前只支持 SQL 表达式，仅支持 SQL 参数
*/
config.expressionType = .SQL;
/**
描述待检索对象的格式
*/
let inputS = QCloudInputSerialization();
inputS.compressionType = .NONE;
/**
描述在JSON对象格式下所需的文件参数。
*/
let inputJson = QCloudSerializationJSON.init();
/**
SON 文件的类型：
DOCUMENT 表示 JSON 文件仅包含一个独立的 JSON 对象，且该对象可以被切割成多行
LINES 表示 JSON 对象中的每一行包含了一个独立的 JSON 对象
合法值为 DOCUMENT 、LINES
*/
inputJson.type = .document;
inputS.serializationJSON = inputJson;
config.inputSerialization = inputS;
/**
描述检索结果的输出格式
*/
let outputS = QCloudOutputSerialization.init();
let outputJson = QCloudSerializationJSON.init();
/**
将输出结果中的记录分隔为不同行的字符，默认通过\n进行分隔。您可以指定任意8进制字符，
如逗号、分号、Tab 等。该参数最多支持2个字节，即您可以输入\r\n这类格式的分隔符。默认值为\n
*/
outputJson.outputRecordDelimiter = "\n";
outputS.serializationJSON = outputJson;
config.outputSerialization = outputS;
/**
是否需要返回查询进度 QueryProgress 信息，如果选中 COS Select 将周期性返回查询进度
*/
let requestProgress = QCloudRequestProgress.init();
requestProgress.enabled = "FALSE";
config.requestProgress = requestProgress;
request.selectObjectContentConfig = config;
//文件在本地的存储路径
request.downloadingURL = NSURL.fileURL(withPath: QCloudFileInSubPath("test", "2.j
son"));
request.finishBlock = {(result,error) in
if error != nil{
print(error!)
```

```
}else{
print(result!);
}
}
QCloudCOSXMLService.defaultCOSXML().selectObjectContent(request);
```

说明：

更多完整示例，请前往 GitHub 查看。

检查对象是否存在

最近更新时间：2022-01-23 12:41:48

简介

本文档提供快捷查询存储桶中某个对象是否存在的示例代码。示例代码实际调用了 HEAD Object COS API，是该接口的简化版。

HEAD Object 除了检查对象是否存在，主要功能为返回对象元数据。如需查看包含了 HEAD Object 完整功能的 SDK 接口，请参见 查询对象元数据。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

查询对象元数据

功能说明

检查存储桶中是否存在某个对象。

示例代码

Objective-C

```
//存储桶名称，格式为 BucketName-APPID
NSString *bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
NSString *object = @"exampleobject";
[[QCloudCOSXMLService defaultCOSXML] doesObjectExistWithBucket:bucket object:object];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
// 存储桶名称，格式为 BucketName-APPID
let bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
let object = "exampleobject";
QCloudCOSXMLService.defaultCOSXML().doesObjectExist(withBucket: bucket, object: o
bject);
```

说明：

更多完整示例，请前往 GitHub 查看。

删除对象

最近更新时间：2021-11-30 12:36:32

简介

本文档提供关于对象的删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

删除对象

功能说明

删除指定的对象（DELETE Object）。

示例代码一：删除单个对象

Objective-C

```
QCloudDeleteObjectRequest* deleteObjectRequest = [QCloudDeleteObjectRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
deleteObjectRequest.bucket = @"examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
deleteObjectRequest.object = @"exampleobject";
[deleteObjectRequest setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteObject:deleteObjectRequest];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let deleteObject = QCloudDeleteObjectRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
deleteObject.bucket = "examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
deleteObject.object = "exampleobject";
deleteObject.finishBlock = {(result,error)in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().deleteObject(deleteObject);
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码二：删除目录

Objective-C

```
QCloudGetBucketRequest* request = [QCloudGetBucketRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 单次返回的最大条目数量，默认1000
request.maxKeys = 100;
//要删除的目录名称：带'/'
request.prefix = @"prefix";
[request setFinishBlock:^(QCloudListBucketResult * result, NSError* error) {
if(!error){
NSMutableArray *deleteInfosArr = [NSMutableArray array];
for (QCloudBucketContents *content in result.contents) {
QCloudDeleteMultipleObjectRequest *delteRequest = [QCloudDeleteMultipleObjectRequ
```

```
est new];
delteRequest.bucket = request.bucket;
QCloudDeleteObjectInfo *object = [QCloudDeleteObjectInfo new];
object.key = content.key;
[deleteInfosArr addObject:object];
}

QCloudDeleteInfo *deleteInfos = [QCloudDeleteInfo new];
deleteInfos.objects = [deleteInfosArr copy];

QCloudDeleteMultipleObjectRequest *delteRequest = [QCloudDeleteMultipleObjectRequ
est new];
delteRequest.bucket = @"examplebucket-1250000000";
delteRequest.deleteObjects = deleteInfos;
[delteRequest setFinishBlock:^(QCloudDeleteResult *outputObject, NSError *error)
{
NSLog(@"outputObject = %@",outputObject);
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteMultipleObject:delteRequest];

}
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucket:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getBucketReq = QCloudGetBucketRequest.init();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
getBucketReq.bucket = "examplebucket-1250000000";

// 单次返回的最大条目数量，默认1000
getBucketReq.maxKeys = 100;

//要删除的目录名称：带'/'
getBucketReq.prefix = "dir/";

getBucketReq.setFinish { (result, error) in
if let result = result {
```

```
let contents = result.contents;
let infos = NSMutableArray.init();
for content in contents {
let info = QCloudDeleteObjectInfo.init();
info.key = content.key;
infos.add(info);
}
let mutipleDel = QCloudDeleteMultipleObjectRequest.init();
// 要删除的文件集合
let deleteInfos = QCloudDeleteInfo.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
mutipleDel.bucket = "examplebucket-1250000000";

deleteInfos.objects = infos as! [QCloudDeleteObjectInfo];

// 布尔值，这个值决定了是否启动 Quiet 模式：
// true：启动 Quiet 模式
// false：启动 Verbose 模式
// 默认值为 False
deleteInfos.quiet = false;

// 封装了需要批量删除的多个对象的信息
mutipleDel.deleteObjects = deleteInfos;

mutipleDel.setFinish { (result, error) in
if let result = result {
let deleted = result.deletedObjects
let failed = result.deletedFailedObjects
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().deleteMultipleObject(mutipleDel);
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getBucket(getBucketReq);
```

说明：
更多完整示例，请前往 GitHub 查看。

删除多个对象

功能说明

批量删除多个对象（DELETE Multiple Objects）。

示例代码一：批量删除对象

Objective-C

```
QCloudDeleteMultipleObjectRequest* delteRequest = [QCloudDeleteMultipleObjectRequ
est new];
delteRequest.bucket = @"examplebucket-1250000000";
// 要删除的单个文件
QCloudDeleteObjectInfo* deletedObject0 = [QCloudDeleteObjectInfo new];
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
deletedObject0.key = @"exampleobject";
// 要删除的文件集合
QCloudDeleteInfo* deleteInfo = [QCloudDeleteInfo new];
// 布尔值，这个值决定了是否启动 Quiet 模式：
// true：启动 Quiet 模式
// false：启动 Verbose 模式
// 默认值为 False
deleteInfo.quiet = NO;
// 存放需要删除对象信息的数组
deleteInfo.objects = @[deletedObject0];
// 封装了需要批量删除的多个对象的信息
delteRequest.deleteObjects = deleteInfo;
[delteRequest setFinishBlock:^(QCloudDeleteResult* outputObject,
NSError *error) {
// 可以从 outputObject 中获取 response 中 etag 或者自定义头部等信息

}];
[[QCloudCOSXMLService defaultCOSXML] DeleteMultipleObject:delteRequest];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let mutipleDel = QCloudDeleteMultipleObjectRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
```

```
mutipleDel.bucket = "examplebucket-1250000000";
// 要删除的单个文件
let info1 = QCloudDeleteObjectInfo.init();
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
info1.key = "exampleobject";
let info2 = QCloudDeleteObjectInfo.init();
// 要删除的文件集合
let deleteInfos = QCloudDeleteInfo.init();
// 存放需要删除对象信息的数组
deleteInfos.objects = [info1,info2];
// 布尔值，这个值决定了是否启动 Quiet 模式：
// true：启动 Quiet 模式
// false：启动 Verbose 模式
// 默认值为 False
deleteInfos.quiet = false;
// 封装了需要批量删除的多个对象的信息
mutipleDel.deleteObjects = deleteInfos;
mutipleDel.setFinish { (result, error) in
if let result = result {
let deleted = result.deletedObjects
let failed = result.deletedFailedObjects
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().deleteMultipleObject(mutipleDel);
```

说明：
更多完整示例，请前往 GitHub 查看。

示例代码二：删除带指定前缀的对象

Objective-C

```
QCloudGetBucketRequest* request = [QCloudGetBucketRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 单次返回的最大条目数量，默认1000
request.maxKeys = 100;
//如果要删除指定前缀的文件:prefix为文件名前缀
request.prefix = @"prefix";
[request setFinishBlock:^(QCloudListBucketResult * result, NSError* error) {
if(!error){
```

```
NSMutableArray *deleteInfosArr = [NSMutableArray array];
for (QCloudBucketContents *content in result.contents) {
QCloudDeleteMultipleObjectRequest *delteRequest = [QCloudDeleteMultipleObjectRequ
est new];
delteRequest.bucket = request.bucket;
QCloudDeleteObjectInfo *object = [QCloudDeleteObjectInfo new];
object.key = content.key;
[deleteInfosArr addObject:object];
}

QCloudDeleteInfo *deleteInfos = [QCloudDeleteInfo new];
deleteInfos.objects = [deleteInfosArr copy];

QCloudDeleteMultipleObjectRequest *delteRequest = [QCloudDeleteMultipleObjectRequ
est new];
delteRequest.bucket = @"examplebucket-1250000000";
delteRequest.deleteObjects = deleteInfos;
[delteRequest setFinishBlock:^(QCloudDeleteResult *outputObject, NSError *error)
{
NSLog(@"outputObject = %@",outputObject);
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteMultipleObject:delteRequest];

}
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucket:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getBucketReq = QCloudGetBucketRequest.init();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
getBucketReq.bucket = "examplebucket-1250000000";

// 单次返回的最大条目数量，默认1000
getBucketReq.maxKeys = 100;

//如果要删除指定前缀的文件:prefix为文件名前缀
getBucketReq.prefix = "dir/";
```

```
getBucketReq.setFinish { (result, error) in
if let result = result {
let contents = result.contents;
let infos = NSMutableArray.init();
for content in contents {
let info = QCloudDeleteObjectInfo.init();
info.key = content.key;
infos.add(info);
}
let mutipleDel = QCloudDeleteMultipleObjectRequest.init();
// 要删除的文件集合
let deleteInfos = QCloudDeleteInfo.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
mutipleDel.bucket = "examplebucket-1250000000";

deleteInfos.objects = infos as! [QCloudDeleteObjectInfo];

// 布尔值，这个值决定了是否启动 Quiet 模式：
// true：启动 Quiet 模式
// false：启动 Verbose 模式
// 默认值为 False
deleteInfos.quiet = false;

// 封装了需要批量删除的多个对象的信息
mutipleDel.deleteObjects = deleteInfos;

mutipleDel.setFinish { (result, error) in
if let result = result {
let deleted = result.deletedObjects
let failed = result.deletedFailedObjects
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().deleteMultipleObject(mutipleDel);
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getBucket(getBucketReq);
```

说明：

更多完整示例，请前往 GitHub 查看。

恢复归档对象

最近更新时间：2021-11-30 12:36:32

简介

本文档提供关于恢复归档对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
POST Object restore	恢复归档对象	将归档类型的对象取回访问

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

恢复归档对象

功能说明

将归档类型的对象取回访问（POST Object restore）。

示例代码

Objective-C

```
QCloudPostObjectRestoreRequest *req = [QCloudPostObjectRestoreRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
req.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
req.object = @"exampleobject";
// 设置临时副本的过期时间
req.restoreRequest.days = 10;
// 复原的过程类型配置信息
req.restoreRequest.CASJobParameters.tier = QCloudCASTierStandard;
[req setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] PostObjectRestore:req];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let restore = QCloudPostObjectRestoreRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
restore.bucket = "examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
restore.object = "exampleobject";
// 设置临时副本的过期时间
restore.restoreRequest.days = 10;
// 复原的过程类型配置信息
restore.restoreRequest.casJobParameters.tier = .standard;
restore.finishBlock = {(result,error)in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().postObjectRestore(restore);
```

说明：
更多完整示例，请前往 GitHub 查看。

查询对象元数据

最近更新时间：2021-11-30 12:36:33

简介

本文档提供关于查询对象元数据操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

查询对象元数据

功能说明

查询 Object 的 Meta 信息。

示例代码

Objective-C

```
QCloudHeadObjectRequest* headerRequest = [QCloudHeadObjectRequest new];
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
headerRequest.object = @"exampleobject";
// versionId 当启用版本控制时，指定要查询的版本 ID，如不指定则查询对象的最新版本
headerRequest.versionID = @"versionID";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
headerRequest.bucket = @"examplebucket-1250000000";
[headerRequest setFinishBlock:^(NSDictionary* result, NSError *error) {
// result 返回具体信息
// 获取文件crc64
NSString * crc64 = [[outputObject __originHTTPURLResponse__].allHeaderFields valueForKey:@"x-cos-hash-crc64ecma"];
}];
[[QCloudCOSXMLService defaultCOSXML] HeadObject:headerRequest];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let headObject = QCloudHeadObjectRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
headObject.bucket = "examplebucket-1250000000";
// versionId 当启用版本控制时，指定要查询的版本 ID，如不指定则查询对象的最新版本
headObject.versionID = "versionID";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
headObject.object = "exampleobject";
headObject.finishBlock = {(result,error) in
if let result = result {
// 获取文件crc64
let crc64 = result?.__originHTTPURLResponse__.allHeaderFields["x-cos-hash-crc64ecma"];

} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().headObject(headObject);
```

说明：
更多完整示例，请前往 GitHub 查看。

服务端加密

最近更新时间：2021-12-27 12:43:14

简介

本文档提供关于如何使用在上传对象时开启服务端加密。服务端加密的密钥分为三种：

- COS 托管加密密钥
- 客户提供的加密密钥
- KMS 托管加密密钥

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

使用 COS 托管加密密钥的服务端加密（SSE-COS）保护数据

功能说明

由腾讯云 COS 托管主密钥和管理数据。COS 会帮助您在数据写入数据中心时自动加密，并在您取用该数据时自动解密。目前支持使用 COS 主密钥对数据进行 AES-256 加密。

示例代码

Objective-C

```
QCloudCOSXMLUploadObjectRequest *request = [QCloudCOSXMLUploadObjectRequest new];
[request setCOSServerSideEncyption];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
request.setCOSServerSideEncyption();
```

说明：
更多完整示例，请前往 GitHub 查看。

使用客户提供的加密密钥的服务端加密 （SSE-C）保护数据

功能说明

加密密钥由用户自己提供，用户在上传对象时，COS 将使用用户提供的加密密钥对用户的数据进行 AES-256 加密。

注意：

- 该加密所运行的服务需要使用 HTTPS 请求。
- 用户需要提供一个32字节的字符串作为密钥，支持数字、字母、字符的组合，不支持中文。
- 如果上传文件时设置了密钥加密，那么在使用 GET（下载）、HEAD（查询）源对象时也需要在请求中带上相同的密钥，才能正常响应。

示例代码

Objective-C

```
QCloudCOSXMLUploadObjectRequest *request = [QCloudCOSXMLUploadObjectRequest new];
NSString *customKey = @"123456qwertyuioplkjhgfdsazxcvbnm";
[request setCOSServerSideEncyptionWithCustomerKey:customKey];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let customKey = "123456qwertyuioplkjhgfdsazxcvbnm";
request.setCOSServerSideEncyptionWithCustomerKey(customKey);
```

说明：
更多完整示例，请前往 GitHub 查看。

使用 KMS 托管加密密钥的服务端加密（SSE-KMS）保护数据

功能说明

SSE-KMS 加密即使用 KMS 托管密钥的服务端加密。KMS 是腾讯云推出的一款安全管理类服务，使用经过第三方认证的硬件安全模块 HSM（Hardware Security Module）来生成和保护密钥。它能够帮助用户轻松创建和管理密钥，满足用户多应用多业务的密钥管理需求以及满足监管和合规要求。关于如何开通 KMS 服务请参考：服务端加密概述。

示例代码

Objective-C

```
QCloudCOSXMLUploadObjectRequest *request = [QCloudCOSXMLUploadObjectRequest new];
NSString *customKey = @"123456qwertyuioplkjhgfdsazxcvbnm";
NSString *arrJsonStr = @"{\"key\":\"value\"}";
[request setCOSServerSideEncyptionWithKMSCustomKey:customKey jsonStr:arrJsonStr];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let customKey = "123456qwertyuioplkjhgfdsazxcvbnm";
let arrJsonStr = "{\"key\":\"value\"}";
self.advancedRequest?.setCOSServerSideEncyptionWithKMSCustomKey(customKey, jsonStr: arrJsonStr);
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

对象访问 URL

最近更新时间：2022-01-17 12:27:02

简介

本文档提供获取对象访问 URL 的代码示例。

> 说明：
>
> - 查询对象访问的 URL，该接口不会判断对象是否真实存在。
> - 如果您的文件是私有读权限，那么本接口生成的 URL 不能直接用于访问资源，您可以通过 生成预签名链接 文档来生成 URL。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

获取对象访问 URL

示例代码一：获取对象访问 URL

Objective-C

```
QCloudGetPresignedURLRequest* getPresignedURLRequest = [[QCloudGetPresignedURLReq
uest alloc] init];
// 生成的链接不携带签名
getPresignedURLRequest.isUseSignature = false:
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
getPresignedURLRequest.bucket = @"examplebucket-1250000000";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
getPresignedURLRequest.object = @"exampleobject";
[getPresignedURLRequest setFinishBlock:^(QCloudGetPresignedURLResult * _Nonnull r
esult,
NSError * _Nonnull error) {
// 预签名 URL
NSString* presignedURL = result.presienedURL;
```

```
}];
[[QCloudCOSXMLService defaultCOSXML] getPresignedURL:getPresignedURLRequest];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getPresign = QCloudGetPresignedURLRequest.init();
// 生成的链接不携带签名
getPresign.isUseSignature = false;
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
getPresign.bucket = "examplebucket-1250000000" ;
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
getPresign.object = "exampleobject";
getPresign.setFinish { (result, error) in
if let result = result {
let url = result.presienedURL
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getPresignedURL(getPresign);
```

说明：
更多完整示例，请前往 GitHub 查看。

生成预签名链接

最近更新时间：2023-06-14 14:28:32

简介

本文档提供关于生成对象预签名链接的示例代码。

关于使用预签名 URL 上传的说明请参见 预签名授权上传， 使用预签名 URL 下载的说明请参见 预签名授权下载。

说明：

- 建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄露目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

生成对象预签名链接

示例代码一：生成预签名上传链接

Objective-C

```
QCloudGetPresignedURLRequest* getPresignedURLRequest = [[QCloudGetPresignedURLRequest alloc] init];

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
getPresignedURLRequest.bucket = @"examplebucket-1250000000";

// 使用预签名 URL 请求的 HTTP 方法。有效值（大小写敏感）为：@"GET"、@"PUT"、@"POST"、@"DELETE"
getPresignedURLRequest.HTTPMethod = @"PUT";
```

```
// 获取预签名函数，默认签入Header Host；您也可以选择不签入Header Host，但可能导致请求失败或安全漏洞
getPresignedURLRequest.signHost = YES;

// http 请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此HTTP请求的参数
getPresignedURLRequest.requestParameters = @{@"param1":@"value1",@"param1":@"value1"};

// http 请求头部，传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的HTTP请求头部
getPresignedURLRequest.requestHeaders = @{@"param1":@"value1",@"param1":@"value1"};

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
getPresignedURLRequest.object = @"exampleobject";

[getPresignedURLRequest setFinishBlock:^(QCloudGetPresignedURLResult * _Nonnull result,
NSError * _Nonnull error) {
// 预签名 URL
NSString* presignedURL = result.presienedURL;
// 使用预签名链接进行上传文件
NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:[NSURL URLWithString:result.presienedURL]];
// 指定HTTPMethod 为PUT
request.HTTPMethod = @"PUT";
// fromData 为需要上传的文件
[[[NSURLSession sharedSession]
uploadTaskWithRequest:request fromData:[@"testtest" dataUsingEncoding:NSUTF8StringEncoding] completionHandler:^(NSData * _Nullable data, NSURLResponse * _Nullable response, NSError * _Nullable error) {
// response中查看上传结果
}]resume];

}];

[[QCloudCOSXMLService defaultCOSXML] getPresignedURL:getPresignedURLRequest];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getPresign = QCloudGetPresignedURLRequest.init();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
getPresign.bucket = "examplebucket-1250000000" ;

// 使用预签名 URL 请求的 HTTP 方法。有效值（大小写敏感）为：
// @"GET"、@"PUT"、@"POST"、@"DELETE"
getPresign.httpMethod = "PUT";

// 获取预签名函数，默认签入Header Host；您也可以选择不签入Header Host，但可能导致请求失败或安全漏洞
getPresign.signHost = YES;

// http 请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此HTTP请求的参数
getPresign.requestParameters = {"param1":"value1","param1":"value1"};

// http 请求头部，传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的HTTP请求头部
getPresign.requestHeaders = {"param1":"value1","param1":"value1"};

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
getPresign.object = "exampleobject";
getPresign.setFinish { (result, error) in
if let result = result {
let url = result.presienedURL
// 使用预签名链接进行上传文件
let request = NSMutableURLRequest.init(url: NSURL.init(string:url) as! URL);
// 指定HTTPMethod 为PUT
request.httpMethod = "PUT";
// fromData 为需要上传的文件
URLSession.shared.uploadTask(with: request, from: "testtest".data(using: String.Encoding.utf8)) { data, res, err in
// response中查看上传结果
}.resume()
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getPresignedURL(getPresign);
```

说明：

更多完整示例，请前往 GitHub 查看。

示例代码二：生成预签名下载链接

Objective-C

```
QCloudGetPresignedURLRequest* getPresignedURLRequest = [[QCloudGetPresignedURLRequest alloc] init];

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
getPresignedURLRequest.bucket = @"examplebucket-1250000000";

// 使用预签名 URL 请求的 HTTP 方法。有效值（大小写敏感）为：@"GET"、@"PUT"、@"POST"、@"DELETE"
getPresignedURLRequest.HTTPMethod = @"GET";

// 获取预签名函数，默认签入Header Host；您也可以选择不签入Header Host，但可能导致请求失败或安全漏洞
getPresignedURLRequest.signHost = YES;

// http 请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此HTTP请求的参数
getPresignedURLRequest.requestParameters = @{@"param1":@"value1",@"param1":@"value1"};

// http 请求头部，传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的HTTP请求头部
getPresignedURLRequest.requestHeaders = @{@"param1":@"value1",@"param1":@"value1"};

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
getPresignedURLRequest.object = @"exampleobject";

[getPresignedURLRequest setFinishBlock:^(QCloudGetPresignedURLResult * _Nonnull result,
NSError * _Nonnull error) {
// 预签名 URL
NSString* presignedURL = result.presienedURL;
// 使用预签名链接进行下载文件
NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:[NSURL URLWithString:presignedURL]];
// 指定HTTPMethod为GET
request.HTTPMethod = @"GET";
[[[NSURLSession sharedSession]
downloadTaskWithRequest:request
completionHandler:^(NSURL *_Nullable location, NSURLResponse *_Nullable response,
NSError *_Nullable error) {
// location 下载成功后的本地文件路径
}] resume];
}];
```

```
[[QCloudCOSXMLService defaultCOSXML] getPresignedURL:getPresignedURLRequest];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getPresign = QCloudGetPresignedURLRequest.init();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
getPresign.bucket = "examplebucket-1250000000" ;

// 使用预签名 URL 请求的 HTTP 方法。有效值（大小写敏感）为：
// @"GET"、@"PUT"、@"POST"、@"DELETE"
getPresign.httpMethod = "GET";

// 获取预签名函数，默认签入Header Host；您也可以选择不签入Header Host，但可能导致请求失败或安全漏洞
getPresignedURLRequest.signHost = YES;

// http 请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此HTTP请求的参数
getPresignedURLRequest.requestParameters = {"param1":"value1","param1":"value1"};

// http 请求头部，传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的HTTP请求头部
getPresignedURLRequest.requestHeaders = {"param1":"value1","param1":"value1"};

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
getPresign.object = "exampleobject";
getPresign.setFinish { (result, error) in
if let result = result {
let url = result.presienedURL
// 使用预签名链接进行下载文件
let request = NSMutableURLRequest.init(url: NSURL.init(string: url) as! URL);
// 指定HTTPMethod为GET
request.httpMethod = "GET";
URLSession.shared.downloadTask(with: request) { location, response, error in
// location 下载成功后的本地文件路径
}.resume();
} else {
print(error!);
}
```

```
}
QCloudCOSXMLService.defaultCOSXML().getPresignedURL(getPresign);
```

说明：
更多完整示例，请前往 GitHub 查看。

预请求跨域配置

最近更新时间：2021-11-30 12:36:33

简介

本文档提供关于预请求跨域配置操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
Options Object	预请求跨域配置	用预请求来确认是否可以发送真正的跨域请求

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

预请求跨域配置

功能说明

获取预请求跨域配置（Options Object）。

示例代码

Objective-C

```
QCloudOptionsObjectRequest* request = [[QCloudOptionsObjectRequest alloc] init];
// 存储桶名称，格式：BucketName-APPID
request.bucket =@"examplebucket-1250000000";
// 模拟跨域访问的请求来源域名，请求 method，请求 host
request.origin = @"http://cloud.tencent.com";
request.accessControlRequestMethod = @"GET";
request.accessControlRequestHeaders = @"host";
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
[request setFinishBlock:^(id outputObject, NSError* error) {
// 可以从 outputObject 中获取 response 中 etag 或者自定义头部等信息
NSDictionary* info = (NSDictionary *) outputObject;

}];
[[QCloudCOSXMLService defaultCOSXML] OptionsObject:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let optionsObject = QCloudOptionsObjectRequest.init();
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
optionsObject.object = "exampleobject";
// 模拟跨域访问的请求来源域名，请求 method，请求头部
optionsObject.origin = "http://www.qcloud.com";
optionsObject.accessControlRequestMethod = "GET";
optionsObject.accessControlRequestHeaders = "origin";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
optionsObject.bucket = "examplebucket-1250000000";
optionsObject.finishBlock = {(result,error) in
if let result = result {
// 可以从 result 中获取服务器返回的 header 信息
}
}
QCloudCOSXMLService.defaultCOSXML().optionsObject(optionsObject);
```

说明：
更多完整示例，请前往 GitHub 查看。

异地容灾
版本控制

最近更新时间：2022-01-17 12:27:02

简介

本文档提供关于版本控制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置版本控制

功能说明

设置指定存储桶的版本控制功能。开启版本控制功能后，只能暂停，不能关闭。

示例代码

Objective-C

```
// 开启版本控制
QCloudPutBucketVersioningRequest* request = [[QCloudPutBucketVersioningRequest al
loc] init];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket =@"examplebucket-1250000000";
// 说明版本控制的具体信息
QCloudBucketVersioningConfiguration* versioningConfiguration =
[[QCloudBucketVersioningConfiguration alloc] init];
request.configuration = versioningConfiguration;
// 说明版本是否开启，枚举值：QCloudCOSBucketVersioningStatusEnabled、
```

```
// QCloudCOSBucketVersioningStatusSuspended
versioningConfiguration.status = QCloudCOSBucketVersioningStatusEnabled;
[request setFinishBlock:^(id outputObject, NSError* error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketVersioning:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
// 开启版本控制
let putBucketVersioning = QCloudPutBucketVersioningRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
putBucketVersioning.bucket = "examplebucket-1250000000";
// 说明版本控制的具体信息
let config = QCloudBucketVersioningConfiguration.init();
// 说明版本是否开启，枚举值：Suspended、Enabled
config.status = .enabled;
putBucketVersioning.configuration = config;
putBucketVersioning.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putBucketVersioning(putBucketVersioning);
```

说明：
更多完整示例，请前往 GitHub 查看。

查询版本控制

功能说明

查询指定存储桶的版本控制信息。

- 获取存储桶版本控制的状态，需要有该存储桶的读权限。
- 有三种版本控制状态：未启用版本控制、启用版本控制和暂停版本控制。

示例代码

Objective-C

```
QCloudGetBucketVersioningRequest* request =
[[QCloudGetBucketVersioningRequest alloc] init];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
[request setFinishBlock:^(QCloudBucketVersioningConfiguration* result,
NSError* error) {
// 获取多版本状态
QCloudCOSBucketVersioningStatus * status = result.status;
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketVersioning:request];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let getBucketVersioning = QCloudGetBucketVersioningRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
getBucketVersioning.bucket = "examplebucket-1250000000";
getBucketVersioning.setFinish { (config, error) in
if let config = config {
// 多版本状态
let status = config.status
} else {
print(error!);
}

}
QCloudCOSXMLService.defaultCOSXML().getBucketVersioning(getBucketVersioning);
```

说明：
更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

跨地域复制

最近更新时间：2022-01-17 12:27:02

简介

本文档提供关于跨地域复制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置跨地域复制

功能说明

设置指定存储桶的跨地域复制规则。

示例代码

Objective-C

```
QCloudPutBucketReplicationRequest* request = [[QCloudPutBucketReplicationRequest
alloc] init];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
// 说明所有跨地域配置信息
QCloudBucketReplicationConfiguation* replConfiguration =
[[QCloudBucketReplicationConfiguation alloc] init];
// 发起者身份标示
replConfiguration.role = @"qcs::cam::uin/100000000001:uin/100000000001";
// 具体配置信息
```

```
QCloudBucketReplicationRule* rule = [[QCloudBucketReplicationRule alloc] init];
// 用来标注具体 Rule 的名称
rule.identifier = @"identifier";
rule.status = QCloudCOSXMLStatusEnabled;
// 资源标识符
QCloudBucketReplicationDestination* destination = [[QCloudBucketReplicationDestin
ation alloc] init];
NSString* destinationBucket = @"destinationbucket-1250000000";
// 目标存储桶所在地域
NSString* region = @"ap-beijing";
destination.bucket = [NSString stringWithFormat:@"qcs::cos:%@::%@",region,destina
tionBucket];
// 目标存储桶信息
rule.destination = destination;
// 前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空
rule.prefix = @"prefix1";
replConfiguration.rule = @[rule];
request.configuation = replConfiguration;
[request setFinishBlock:^(id outputObject, NSError* error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;

}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketRelication:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let putBucketReplication = QCloudPutBucketReplicationRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
putBucketReplication.bucket = "examplebucket-1250000000";
// 说明所有跨地域配置信息
let config = QCloudBucketReplicationConfiguation.init();
config.role = "qcs::cam::uin/100000000001:uin/100000000001";
// 发起者身份标示
let rule = QCloudBucketReplicationRule.init();

// 用来标注具体 Rule 的名称
rule.identifier = "rule1";
// 规则开启状态，可选 .enabled, .disabled
```

```
rule.status = .enabled;
// 目标存储桶信息
let destination = QCloudBucketReplicationDestination.init();
let destinationBucket = "destinationbucket-1250000000";
let region = "ap-beijing";
destination.bucket = "qcs::cos:\(region)::\(destinationBucket)";
rule.destination = destination;
// 前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空
rule.prefix = "dir/";
config.rule = [rule];
putBucketReplication.configuation = config;
putBucketReplication.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putBucketRelication(putBucketReplication);
```

说明：
更多完整示例，请前往 GitHub 查看。

查询跨地域复制

功能说明

查询指定存储桶的跨地域复制规则。

示例代码

Objective-C

```
QCloudGetBucketReplicationRequest* request = [[QCloudGetBucketReplicationRequest
alloc] init];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
[request setFinishBlock:^(QCloudBucketReplicationConfiguation* result,
NSError* error) {
// 具体配置信息，最多支持 1000 个，所有策略只能指向一个目标存储桶
NSArray *rules = result.rule;
```

```
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketReplication:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getBucketReplication = QCloudGetBucketReplicationRequest.init();
getBucketReplication.bucket = "examplebucket-1250000000";
getBucketReplication.setFinish { (config, error) in
if let config = config {
// 配置信息列表
let rule = config.rule
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getBucketReplication(getBucketReplication);
```

说明：
更多完整示例，请前往 GitHub 查看。

删除跨地域复制

功能说明

删除指定存储桶的跨地域复制规则。

示例代码

Objective-C

```
QCloudDeleteBucketReplicationRequest* request =
[[QCloudDeleteBucketReplicationRequest alloc] init];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
[request setFinishBlock:^(id outputObject, NSError* error) {
```

```
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;

}];
[[QCloudCOSXMLService defaultCOSXML] DeleteBucketReplication:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let deleteBucketReplication = QCloudDeleteBucketReplicationRequest.init();
deleteBucketReplication.bucket = "examplebucket-1250000000";
deleteBucketReplication.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().deleteBucketReplication(deleteBucketReplicati
on);
```

说明：
更多完整示例，请前往 GitHub 查看。

数据管理
生命周期

最近更新时间：2022-01-17 12:27:03

简介

本文档提供关于生命周期的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket lifecycle	设置生命周期	设置存储桶的生命周期管理的配置
GET Bucket lifecycle	查询生命周期	查询存储桶生命周期管理的配置
DELETE Bucket lifecycle	删除生命周期	删除存储桶生命周期管理的配置

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置生命周期

功能说明

设置指定存储桶的生命周期配置信息。

示例代码

Objective-C

```
QCloudPutBucketLifecycleRequest* request = [QCloudPutBucketLifecycleRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
__block QCloudLifecycleConfiguration* lifecycleConfiguration = [[QCloudLifecycleConfiguration alloc] init];
// 规则描述
QCloudLifecycleRule* rule = [[QCloudLifecycleRule alloc] init];
// 用于唯一地标识规则
```

版权所有：腾讯云计算（北京）有限责任公司

```
rule.identifier = @"identifier";
// 指明规则是否启用，枚举值：Enabled，Disabled
rule.status = QCloudLifecycleStatueEnabled;
// Filter 用于描述规则影响的 Object 集合
QCloudLifecycleRuleFilter* filter = [[QCloudLifecycleRuleFilter alloc] init];
// 指定规则所适用的前缀。匹配前缀的对象受该规则影响，Prefix 最多只能有一个
filter.prefix = @"prefix1";
// Filter 用于描述规则影响的 Object 集合
rule.filter = filter;
// 规则转换属性，对象何时转换为 Standard_IA 或 Archive
QCloudLifecycleTransition* transition = [[QCloudLifecycleTransition alloc] init];
// 指明规则对应的动作在对象最后的修改日期过后多少天操作：
transition.days = 100;
// 指定 Object 转储到的目标存储类型，枚举值： STANDARD_IA，ARCHIVE
transition.storageClass = QCloudCOSStorageStandardIA;
rule.transition = transition;
request.lifeCycle = lifecycleConfiguration;
// 生命周期配置
request.lifeCycle.rules = @[rule];
[request setFinishBlock:^(id outputObject, NSError* error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketLifecycle:request];
```

说明：

更多完整示例，请前往 GitHub 查看。

Swift

```
let putBucketLifecycleReq = QCloudPutBucketLifecycleRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
putBucketLifecycleReq.bucket = "examplebucket-1250000000";
let config = QCloudLifecycleConfiguration.init();
// 规则描述
let rule = QCloudLifecycleRule.init();
// 用于唯一地标识规则
rule.identifier = "swift";
// 指明规则是否启用，枚举值：Enabled，Disabled
rule.status = .enabled;
// Filter 用于描述规则影响的 Object 集合
let fileter = QCloudLifecycleRuleFilter.init();
```

```
// 指定规则所适用的前缀。匹配前缀的对象受该规则影响，Prefix 最多只能有一个
fileter.prefix = "0";
// Filter 用于描述规则影响的 Object 集合
rule.filter = fileter;
// 规则转换属性，对象何时转换为 Standard_IA 或 Archive
let transition = QCloudLifecycleTransition.init();
// 指明规则对应的动作在对象最后的修改日期过后多少天操作：
transition.days = 100;
// 指定 Object 转储到的目标存储类型，枚举值： STANDARD_IA, ARCHIVE
transition.storageClass = .standardIA;
rule.transition = transition;
putBucketLifecycleReq.lifeCycle = config;
// 生命周期配置
putBucketLifecycleReq.lifeCycle.rules = [rule];
putBucketLifecycleReq.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putBucketLifecycle(putBucketLifecycleReq);
```

说明：
更多完整示例，请前往 GitHub 查看。

查询生命周期

功能说明

查询存储桶的生命周期管理配置。

示例代码

Objective-C

```
QCloudGetBucketLifecycleRequest* request = [QCloudGetBucketLifecycleRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
[request setFinishBlock:^(QCloudLifecycleConfiguration* result,NSError* error) {
// 可以从 result 中获取返回信息
```

```
// result.rules 规则描述集合的数组

}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketLifecycle:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getBucketLifeCycle = QCloudGetBucketLifecycleRequest.init();
getBucketLifeCycle.bucket = "examplebucket-1250000000";
getBucketLifeCycle.setFinish { (config, error) in
if let config = config {
// 生命周期规则
let rules = config.rules
} else {
print(error!);
}

};
QCloudCOSXMLService.defaultCOSXML().getBucketLifecycle(getBucketLifeCycle);
```

说明：
更多完整示例，请前往 GitHub 查看。

删除生命周期

功能说明

删除存储桶生命周期管理的配置。

示例代码

Objective-C

```
QCloudDeleteBucketLifeCycleRequest* request =
[[QCloudDeleteBucketLifeCycleRequest alloc ] init];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
```

```
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
[request setFinishBlock:^(QCloudLifecycleConfiguration* deleteResult, NSError* er
ror) {
// 返回删除结果
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteBucketLifeCycle:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let deleteBucketLifeCycle = QCloudDeleteBucketLifeCycleRequest.init();
deleteBucketLifeCycle.bucket = "examplebucket-1250000000";
deleteBucketLifeCycle.finishBlock = { (result, error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
};
QCloudCOSXMLService.defaultCOSXML().deleteBucketLifeCycle(deleteBucketLifeCycle);
```

说明：
更多完整示例，请前往 GitHub 查看。

日志管理

最近更新时间：2022-01-17 12:27:03

简介

本文档提供关于日志管理的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket logging	设置日志管理	为源存储桶开启日志记录
GET Bucket logging	查询日志管理	查询源存储桶的日志配置信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置日志管理

功能说明

PUT Bucket logging 用于为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中。

示例代码

Objective-C

```
QCloudPutBucketLoggingRequest *request = [QCloudPutBucketLoggingRequest new];
// 说明日志记录配置的状态，如果无子节点信息则意为关闭日志记录
QCloudBucketLoggingStatus *status = [QCloudBucketLoggingStatus new];
// 存储桶 logging 设置的具体信息，主要是目标存储桶
QCloudLoggingEnabled *loggingEnabled = [QCloudLoggingEnabled new];
// 存放日志的目标存储桶，可以是同一个存储桶（但不推荐），或同一账户下、同一地域的存储桶
loggingEnabled.targetBucket = @"examplebucket-1250000000";
// 日志存放在目标存储桶的指定路径
loggingEnabled.targetPrefix = @"mylogs";
status.loggingEnabled = loggingEnabled;
request.bucketLoggingStatus = status;
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
```

版权所有：腾讯云计算（北京）有限责任公司

```
request.bucket = @"examplebucket-1250000000";
[request setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketLogging:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudPutBucketLoggingRequest.init();
// 说明日志记录配置的状态，如果无子节点信息则意为关闭日志记录
let status = QCloudBucketLoggingStatus.init();
// 存储桶 logging 设置的具体信息，主要是目标存储桶
let loggingEnabled = QCloudLoggingEnabled.init();
// 存放日志的目标存储桶，可以是同一个存储桶（但不推荐），或同一账户下、同一地域的存储桶
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
loggingEnabled.targetBucket = "examplebucket-1250000000";
// 日志存放在目标存储桶的指定路径
loggingEnabled.targetPrefix = "logs/";
status.loggingEnabled = loggingEnabled;
req.bucketLoggingStatus = status;
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
req.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putBucketLogging(req);
```

说明：
更多完整示例，请前往 GitHub 查看。

查询日志管理

功能说明

GET Bucket logging 用于查询指定存储桶的日志配置信息。

示例代码

Objective-C

```
QCloudGetBucketLoggingRequest *getReq = [QCloudGetBucketLoggingRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
getReq.bucket = @"examplebucket-1250000000";
[getReq setFinishBlock:^(QCloudBucketLoggingStatus * _Nonnull result,
NSError * _Nonnull error) {
// 日志配置信息
QCloudLoggingEnabled *loggingEnabled = result.loggingEnabled;
}];
[[QCloudCOSXMLService defaultCOSXML]GetBucketLogging:getReq];
```

说明：

更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudGetBucketLoggingRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
req.setFinish { (result, error) in
if let result = result {
// 日志配置信息
let enabled = result.loggingEnabled
} else {
print(error!);
}
};
QCloudCOSXMLService.defaultCOSXML().getBucketLogging(req);
```

说明：

更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

存储桶标签

最近更新时间：2022-05-25 15:20:00

简介

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket tagging	设置存储桶标签	为已存在的存储桶设置标签
GET Bucket tagging	查询存储桶标签	查询指定存储桶下已有的存储桶标签
DELETE Bucket tagging	删除存储桶标签	删除指定的存储桶标签

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置存储桶标签

功能说明

PUT Bucket tagging 用于为已存在的存储桶设置标签。

示例代码

Objective-C

```
QCloudPutBucketTaggingRequest *putReq = [QCloudPutBucketTaggingRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
putReq.bucket = @"examplebucket-1250000000";
// 标签集合
QCloudTagging *taggings = [QCloudTagging new];
QCloudTag *tag1 = [QCloudTag new];
// 标签的 Key，长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、
// 冒号、斜线
tag1.key = @"age";
// 标签的 Value，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号
```

```
// 、冒号、斜线
tag1.value = @"20";
QCloudTag *tag2 = [QCloudTag new];
tag2.key = @"name";
tag2.value = @"karis";
// 标签集合，最多支持10个标签
QCloudTagSet *tagSet = [QCloudTagSet new];
tagSet.tag = @[tag1,tag2];
taggings.tagSet = tagSet;
// 标签集合
putReq.taggings = taggings;
[putReq setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketTagging:putReq];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudPutBucketTaggingRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
let taggings = QCloudTagging.init();
// 标签集合
let tagSet = QCloudTagSet.init();
taggings.tagSet = tagSet;
let tag1 = QCloudTag.init();
// 标签的 Key，长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、
// 冒号、斜线
tag1.key = "age";
// 标签的 Value，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号
// 、冒号、斜线
tag1.value = "20";
let tag2 = QCloudTag.init();
tag2.key = "name";
tag2.value = "karis";
// 标签集合，最多支持10个标签
tagSet.tag = [tag1,tag2];
// 标签集合
```

```
req.taggings = taggings;
req.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putBucketTagging(req);
```

说明：
更多完整示例，请前往 GitHub 查看。

查询存储桶标签

功能说明

GET Bucket tagging 用于查询指定存储桶下已有的存储桶标签。

示例代码

Objective-C

```
QCloudGetBucketTaggingRequest *getReq = [QCloudGetBucketTaggingRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
getReq.bucket = @"examplebucket-1250000000";
[getReq setFinishBlock:^(QCloudBucketTagging * result, NSError * error) {
// tag的集合
QCloudTagSet * tagSet = result.tagSet;
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketTagging:getReq];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudGetBucketTaggingRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
req.setFinish { (result, error) in
if let result = result {
// 标签集合
let tagSet = result.tagSet
} else {
print(error!);
}
};
QCloudCOSXMLService.defaultCOSXML().getBucketTagging(req);
```

说明：
更多完整示例，请前往 GitHub 查看。

删除存储桶标签

功能说明

DELETE Bucket tagging 用于删除指定存储桶下已有的存储桶标签。

示例代码

Objective-C

```
QCloudDeleteBucketTaggingRequest *delReq = [QCloudDeleteBucketTaggingRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
delReq.bucket = @"examplebucket-1250000000";
[delReq setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteBucketTagging:delReq];
```

说明：

更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudDeleteBucketTaggingRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
req.finishBlock = { (result, error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
};
QCloudCOSXMLService.defaultCOSXML().deleteBucketTagging(req);
```

说明：
更多完整示例，请前往 GitHub 查看。

静态网站

最近更新时间：2022-05-25 15:20:00

简介

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket website	设置静态网站	设置存储桶的静态网站配置
GET Bucket website	查询静态网站配置	查询存储桶的静态网站配置
DELETE Bucket website	删除静态网站配置	删除存储桶的静态网站配置

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置静态网站

功能说明

PUT Bucket website 用于为存储桶配置静态网站。

示例代码

Objective-C

```
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
NSString *bucket = @"examplebucket-1250000000";
NSString *indexDocumentSuffix = @"index.html";
NSString *errorDocKey = @"error.html";
NSString *derPro = @"https";
int errorCode = 451;
NSString * replaceKeyPrefixWith = @"404.html";
QCloudPutBucketWebsiteRequest *putReq = [QCloudPutBucketWebsiteRequest new];
putReq.bucket = bucket;
QCloudWebsiteConfiguration *config = [QCloudWebsiteConfiguration new];
```

版权所有：腾讯云计算（北京）有限责任公司

```
QCloudWebsiteIndexDocument *indexDocument = [QCloudWebsiteIndexDocument new];
// 指定索引文档的对象键后缀。例如指定为index.html，那么当访问到存储桶的根目录时，会自动返回
// index.html 的内容，或者当访问到article/目录时，会自动返回 article/index.html的内容
indexDocument.suffix = indexDocumentSuffix;
// 索引文档配置
config.indexDocument = indexDocument;
// 错误文档配置
QCloudWebisteErrorDocument *errDocument = [QCloudWebisteErrorDocument new];
errDocument.key = errorDocKey;
// 指定通用错误文档的对象键，当发生错误且未命中重定向规则中的错误码重定向时，将返回该对象键的内
容
config.errorDocument = errDocument;
// 重定向所有请求配置
QCloudWebsiteRedirectAllRequestsTo *redir = [QCloudWebsiteRedirectAllRequestsTo n
ew];
redir.protocol = derPro;
// 指定重定向所有请求的目标协议，只能设置为 https
config.redirectAllRequestsTo = redir;
// 单条重定向规则配置
QCloudWebsiteRoutingRule *rule = [QCloudWebsiteRoutingRule new];
// 重定向规则的条件配置
QCloudWebsiteCondition *contition = [QCloudWebsiteCondition new];
contition.httpErrorCodeReturnedEquals = errorCode;
rule.condition = contition;
// 重定向规则的具体重定向目标配置
QCloudWebsiteRedirect *webRe = [QCloudWebsiteRedirect new];
webRe.protocol = derPro;
// 指定重定向规则的具体重定向目标的对象键，替换方式为替换原始请求中所匹配到的前缀部分，
// 仅可在 Condition 为 KeyPrefixEquals 时设置
webRe.replaceKeyPrefixWith = replaceKeyPrefixWith;
rule.redirect = webRe;
QCloudWebsiteRoutingRules *routingRules = [QCloudWebsiteRoutingRules new];
routingRules.routingRule = @[rule];
// 重定向规则配置，最多设置100条 RoutingRule
config.rules = routingRules;
putReq.websiteConfiguration = config;
[putReq setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketWebsite:putReq];
```

说明：

更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudPutBucketWebsiteRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
let indexDocumentSuffix = "index.html";
let errorDocKey = "error.html";
let errorCode = 451;
let replaceKeyPrefixWith = "404.html";
let config = QCloudWebsiteConfiguration.init();
let indexDocument = QCloudWebsiteIndexDocument.init();
// 指定索引文档的对象键后缀。例如指定为index.html，那么当访问到存储桶的根目录时，会自动返回
// index.html 的内容，或者当访问到article/目录时，会自动返回 article/index.html的内容
indexDocument.suffix = indexDocumentSuffix;
// 索引文档配置
config.indexDocument = indexDocument;
// 错误文档配置
let errDocument = QCloudWebisteErrorDocument.init();
errDocument.key = errorDocKey;
// 指定通用错误文档的对象键，当发生错误且未命中重定向规则中的错误码重定向时，将返回该对象键的内容
config.errorDocument = errDocument;
// 重定向所有请求配置
let redir = QCloudWebsiteRedirectAllRequestsTo.init();
// 指定重定向所有请求的目标协议，只能设置为 https
redir.protocol = "https";
config.redirectAllRequestsTo = redir;
// 单条重定向规则配置
let rule = QCloudWebsiteRoutingRule.init();
// 重定向规则的条件配置
let contition = QCloudWebsiteCondition.init();
contition.httpErrorCodeReturnedEquals = Int32(errorCode);
rule.condition = contition;
// 重定向规则的具体重定向目标配置
let webRe = QCloudWebsiteRedirect.init();
webRe.protocol = "https";
// 指定重定向规则的具体重定向目标的对象键，替换方式为替换原始请求中所匹配到的前缀部分，
// 仅可在 Condition 为 KeyPrefixEquals 时设置
webRe.replaceKeyPrefixWith = replaceKeyPrefixWith;
rule.redirect = webRe;
let routingRules = QCloudWebsiteRoutingRules.init();
routingRules.routingRule = [rule];
// 重定向规则配置，最多设置100条 RoutingRule
config.rules = routingRules;
req.websiteConfiguration = config;
```

版权所有：腾讯云计算（北京）有限责任公司

```
req.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putBucketWebsite(req);
```

说明：
更多完整示例，请前往 GitHub 查看。

查询静态网站配置

功能说明

GET Bucket website 用于查询与存储桶关联的静态网站配置信息。

示例代码

Objective-C

```
QCloudGetBucketWebsiteRequest *getReq = [QCloudGetBucketWebsiteRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
getReq.bucket = @"examplebucket-1250000000";
[getReq setFinishBlock:^(QCloudWebsiteConfiguration * result,
NSError * error) {

// 设置重定向规则，最多设置100条RoutingRule
QCloudWebsiteRoutingRules *rules =result.rules;

// 索引文档
QCloudWebsiteIndexDocument *indexDocument = result.indexDocument;

// 错误文档
QCloudWebisteErrorDocument *errorDocument = result.errorDocument;

// 重定向所有请求
QCloudWebsiteRedirectAllRequestsTo *redirectAllRequestsTo = result.redirectAllRequestsTo;
```

```
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketWebsite:getReq];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudGetBucketWebsiteRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
req.setFinish {(result,error) in
if let result = result {
let rules = result.rules
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getBucketWebsite(req);
```

说明：
更多完整示例，请前往 GitHub 查看。

删除静态网站配置

功能说明

DELETE Bucket website 用于删除存储桶中的静态网站配置。

示例代码

Objective-C

```
QCloudDeleteBucketWebsiteRequest *delReq = [QCloudDeleteBucketWebsiteRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
delReq.bucket = @"examplebucket-1250000000";
```

```
[delReq setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteBucketWebsite:delReq];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let delReq = QCloudDeleteBucketWebsiteRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
delReq.bucket = "examplebucket-1250000000";
delReq.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().deleteBucketWebsite(delReq);
```

说明：
更多完整示例，请前往 GitHub 查看。

对象标签

最近更新时间：2022-05-25 15:20:00

简介

本文档提供关于对象标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Object tagging	设置对象标签	为已上传的对象设置标签
GET Object tagging	查询对象标签	查询指定对象下已有的对象标签
DELETE Object tagging	删除对象标签	删除指定对象下已有的对象标签

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置对象标签

功能说明

COS 支持为已存在的对象设置标签。通过为对象添加键值对作为对象标签，可以协助您分组管理已有的对象资源。

示例代码

Objective-C

```
QCloudPutObjectTaggingRequest *putReq = [QCloudPutObjectTaggingRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
putReq.bucket = @"examplebucket-1250000000";
// 标签集合
QCloudTagging *taggings = [QCloudTagging new];
QCloudTag *tag1 = [QCloudTag new];
// 标签的 Key，长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、
// 冒号、斜线
tag1.key = @"age";
// 标签的 Value，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号
```

```
// 、冒号、斜线
tag1.value = @"20";
QCloudTag *tag2 = [QCloudTag new];
tag2.key = @"name";
tag2.value = @"karis";
// 标签集合，最多支持10个标签
QCloudTagSet *tagSet = [QCloudTagSet new];
tagSet.tag = @[tag1,tag2];
taggings.tagSet = tagSet;
// 标签集合
putReq.taggings = taggings;
[putReq setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] PutObjectTagging:putReq];
```

Swift

```
let putReq = QCloudPutObjectTaggingRequest()
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
putReq.bucket = "examplebucket-1250000000";
// 标签集合
let taggings = QCloudTagging();
let tag1 = QCloudTag();
// 标签的 Key，长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、
// 冒号、斜线
tag1.key = "age";
// 标签的 Value，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号
// 、冒号、斜线
tag1.value = "20";
let tag2 = QCloudTag();
tag2.key = "name";
tag2.value = "karis";
// 标签集合，最多支持10个标签
let tagSet = QCloudTagSet();
tagSet.tag = [tag1,tag2];
taggings.tagSet = tagSet;
// 标签集合
putReq.taggings = taggings;
req.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
```

```
}
QCloudCOSXMLService.defaultCOSXML().putObjectTagging(putReq);
```

查询对象标签

功能说明

查询指定对象下已有的对象标签。

示例代码

Objective-C

```
QCloudGetObjectTaggingRequest *getReq = [QCloudGetObjectTaggingRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
getReq.bucket = @"examplebucket-1250000000";
[getReq setFinishBlock:^(QCloudTagging * result, NSError * error) {
// tag的集合
QCloudTagSet * tagSet = result.tagSet;
}];
[[QCloudCOSXMLService defaultCOSXML] GetObjectTagging:getReq];
```

Swift

```
let getReq = QCloudGetObjectTaggingRequest();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
getReq.bucket = "examplebucket-1250000000";
req.finishBlock = {(result,error) in
if let result = result {
// tag的集合
let tagSet = result.tagSet;
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getObjectTagging(getReq);
```

删除对象标签

> 注意：
> COS iOS SDK 版本需要大于等于 v5.9.9。

功能说明

删除指定对象下已有的对象标签。

请求示例

Objective-C

```
QCloudDeleteObjectTaggingRequest *request = [QCloudDeleteObjectTaggingRequest ne
w];
// 文件名
request.object = @"test.png";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";
request.versionId = @"versionId";
[request setFinishBlock:^(id * result, NSError * error) {
if(!error){
// 删除成功
}else{
// 删除失败
}
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteObjectTagging:request];
```

Swift

```
let request = QCloudDeleteObjectTaggingRequest();
// 文件名
request.object = "test.png";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
request.bucket = "examplebucket-1250000000";
request.versionId = "versionId";
req.finishBlock = {(result,error) in
if(!error){
// 删除成功
}else{
// 删除失败
}
}
QCloudCOSXMLService.defaultCOSXML().deleteObjectTagging(request);
```

版权所有：腾讯云计算（北京）有限责任公司

清单

最近更新时间：2022-05-25 15:20:01

简介

本文档提供关于清单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket inventory	设置清单任务	设置存储桶的清单任务
GET Bucket inventory	查询清单任务	查询存储桶的清单任务
DELETE Bucket inventory	删除清单任务	删除存储桶的清单任务

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置清单任务

功能说明

PUT Bucket inventory 用于在存储桶中创建清单任务。

示例代码

Objective-C

```
QCloudPutBucketInventoryRequest *putReq = [QCloudPutBucketInventoryRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
putReq.bucket= @"examplebucket-1250000000";
// 清单任务的名称
putReq.inventoryID = @"list1";
// 用户在请求体中使用 XML 语言设置清单任务的具体配置信息。配置信息包括清单任务分析的对象，
// 分析的频次，分析的维度，分析结果的格式及存储的位置等信息。
QCloudInventoryConfiguration *config = [QCloudInventoryConfiguration new];
// 清单的名称，与请求参数中的 id 对应
config.identifier = @"list1";
```

```
// 清单是否启用的标识：
// 如果设置为 true，清单功能将生效
// 如果设置为 false，将不生成任何清单
config.isEnabled = @"True";
// 描述存放清单结果的信息
QCloudInventoryDestination *des = [QCloudInventoryDestination new];
QCloudInventoryBucketDestination *btDes =[QCloudInventoryBucketDestination new];
// 清单分析结果的文件形式，可选项为 CSV 格式
btDes.cs = @"CSV";
// 存储桶的所有者 ID
btDes.account = @"1278687956";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
btDes.bucket = @"qcs::cos:ap-guangzhou::examplebucket-1250000000";
// 清单分析结果的前缀
btDes.prefix = @"list1";
// COS 托管密钥的加密方式
QCloudInventoryEncryption *enc = [QCloudInventoryEncryption new];
enc.ssecos = @"";
// 为清单结果提供服务端加密的选项
btDes.encryption = enc;
// 清单结果导出后存放的存储桶信息
des.bucketDestination = btDes;
// 描述存放清单结果的信息
config.destination = des;
// 配置清单任务周期
QCloudInventorySchedule *sc = [QCloudInventorySchedule new];
// 清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly
sc.frequency = @"Daily";
config.schedule = sc;
QCloudInventoryFilter *fileter = [QCloudInventoryFilter new];
fileter.prefix = @"myPrefix";
config.filter = fileter;
config.includedObjectVersions = QCloudCOSIncludedObjectVersionsAll;
QCloudInventoryOptionalFields *fields = [QCloudInventoryOptionalFields new];
fields.field = @[ @"Size",
@"LastModifiedDate",
@"ETag",
@"StorageClass",
@"IsMultipartUploaded",
@"ReplicationStatus"];
// 设置清单结果中应包含的分析项目
config.optionalFields = fields;
putReq.inventoryConfiguration = config;
[putReq setFinishBlock:^(id outputObject, NSError *error) {
// 可以从 outputObject 中获取 response 中 etag 或者自定义头部等信息
NSDictionary * result = (NSDictionary *)outputObject;
```

版权所有：腾讯云计算（北京）有限责任公司

```
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketInventory:putReq];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let putReq = QCloudPutBucketInventoryRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
putReq.bucket = "examplebucket-1250000000";
// 清单任务的名称
putReq.inventoryID = "list1";
// 用户在请求体中使用 XML 语言设置清单任务的具体配置信息。配置信息包括清单任务分析的对象，
// 分析的频次，分析的维度，分析结果的格式及存储的位置等信息。
let config = QCloudInventoryConfiguration.init();
// 清单的名称，与请求参数中的 id 对应
config.identifier = "list1";
// 清单是否启用的标识：
// 如果设置为 true，清单功能将生效
// 如果设置为 false，将不生成任何清单
config.isEnabled = "True";
// 描述存放清单结果的信息
let des = QCloudInventoryDestination.init();
let btDes = QCloudInventoryBucketDestination.init();
// 清单分析结果的文件形式，可选项为 CSV 格式
btDes.cs = "CSV";
// 存储桶的所有者 ID
btDes.account = "1278687956";
// 清单分析结果的存储桶名
btDes.bucket = "qcs::cos:ap-guangzhou::examplebucket-1250000000";
// 清单分析结果的前缀
btDes.prefix = "list1";
// COS 托管密钥的加密方式
let enc = QCloudInventoryEncryption.init();
enc.ssecos = "";
// 为清单结果提供服务端加密的选项
btDes.encryption = enc;
// 清单结果导出后存放的存储桶信息
des.bucketDestination = btDes;
// 描述存放清单结果的信息
config.destination = des;
```

```
// 配置清单任务周期
let sc = QCloudInventorySchedule.init();
// 清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly
sc.frequency = "Daily";
config.schedule = sc;
let fileter = QCloudInventoryFilter.init();
fileter.prefix = "myPrefix";
config.filter = fileter;
config.includedObjectVersions = .all;
let fields = QCloudInventoryOptionalFields.init();
fields.field = [ "Size",
"LastModifiedDate",
"ETag",
"StorageClass",
"IsMultipartUploaded",
"ReplicationStatus"];
// 设置清单结果中应包含的分析项目
config.optionalFields = fields;
putReq.inventoryConfiguration = config;
putReq.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putBucketInventory(putReq);
```

说明：
更多完整示例，请前往 GitHub 查看。

错误码说明

该请求可能会发生的一些常见的特殊错误如下：

错误码	描述	状态码
InvalidArgument	不合法的参数值	HTTP 400 Bad Request
TooManyConfigurations	清单数量已经达到1000条的上限	HTTP 400 Bad Request
AccessDenied	未授权的访问。您可能不具备访问该存储桶的权限	HTTP 403 Forbidden

版权所有：腾讯云计算（北京）有限责任公司

查询清单任务

功能说明

GET Bucket inventory 用于查询存储桶中用户的清单任务信息。

示例代码

Objective-C

```
QCloudGetBucketInventoryRequest *getReq = [QCloudGetBucketInventoryRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
getReq.bucket = @"examplebucket-1250000000";
// 清单任务的名称
getReq.inventoryID = @"list1";
[getReq setFinishBlock:^(QCloudInventoryConfiguration * _Nonnull result, NSError * _Nonnull error) {
// result 包含清单的信息
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketInventory:getReq];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudGetBucketInventoryRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
// 清单任务的名称
req.inventoryID = "list1";
req.setFinish {(result,error) in
if let result = result {
// 任务信息
let enabled = result.isEnabled
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getBucketInventory(req);
```

版权所有：腾讯云计算（北京）有限责任公司

> 说明：
> 更多完整示例，请前往 GitHub 查看。

删除清单任务

功能说明

DELETE Bucket inventory 用于删除存储桶中指定的清单任务。

示例代码

Objective-C

```
QCloudDeleteBucketInventoryRequest *delReq = [QCloudDeleteBucketInventoryRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
delReq.bucket = @"examplebucket-1250000000";
// 清单任务的名称
delReq.inventoryID = @"list1";
[delReq setFinishBlock:^(id outputObject, NSError *error) {
// 可以从 outputObject 中获取 response 中 etag 或者自定义头部等信息
NSDictionary * result = (NSDictionary *)outputObject;

}];
[[QCloudCOSXMLService defaultCOSXML] DeleteBucketInventory:delReq];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let delReq = QCloudDeleteBucketInventoryRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
delReq.bucket = "examplebucket-1250000000";
// 清单任务的名称
delReq.inventoryID = "list1";
delReq.finishBlock = {(result,error) in
```

```
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().deleteBucketInventory(delReq);
```

说明：
更多完整示例，请前往 GitHub 查看。

访问管理
存储桶策略

最近更新时间：2023-04-07 14:34:08

简介

本文档提供关于存储桶策略的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket policy	设置存储桶策略	设置指定存储桶的权限策略
GET Bucket policy	查询存储桶策略	查询指定存储桶的权限策略
DELETE Bucket policy	删除存储桶策略	删除指定存储桶的权限策略

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置存储桶策略

功能说明

设置指定存储桶的权限策略（PUT Bucket policy）。

> 注意：
> COS iOS SDK 版本需要大于等于 v6.1.8。

示例代码

```
QCloudPutBucketPolicyRequest * request = [QCloudPutBucketPolicyRequest new];
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https://console.intl.cloud.tencent.com/cos5/bucket
request.bucket = @"0-1250000000";
request.regionName = @"ap-chengdu";
```

```
// 权限策略，详情请参见 访问管理策略语法 https://intl.cloud.tencent.com/document/product/436/12469
request.policyInfo = @{
@"Statement": @[
@{
@"Principal": @{
@"qcs": @[
@"qcs::cam::uin/100000000001:uin/100000000001"
]
},
@"Effect": @"allow",
@"Action": @[
@"name/cos:GetBucket"
],
@"Resource": @[
@"qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
]
}
],
@"version": @"2.0"
};
[request setFinishBlock:^(id _Nullable outputObject, NSError * _Nullable error) {

}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketPolicy:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

查询存储桶策略

功能说明

查询指定存储桶的权限策略（GET Bucket policy）。

注意：
COS Android SDK 版本需要大于等于 v6.1.8。

示例代码

```
QCloudGetBucketPolicyRequest * request = [QCloudGetBucketPolicyRequest new];
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https://console.intl.cloud.tencent.com/cos5/bucket
request.bucket = @"bucketname-appid";
request.regionName = @"ap-chengdu";
[request setFinishBlock:^(QCloudBucketPolicyResult * _Nullable outputObject, NSError * _Nullable error) {
// QCloudBucketPolicyResult 详细字段请查看api文档或者SDK源码
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketPolicy:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

删除存储桶策略

功能说明

删除指定存储桶的权限策略（DELETE Bucket policy）。

注意：
COS Android SDK 版本需要大于等于 v6.1.8。

示例代码

```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https://console.intl.cloud.tencent.com/cos5/bucket
QCloudDeleteBucketPolicyRequest * request = [QCloudDeleteBucketPolicyRequest new];
request.bucket = @"0-1253960454";
request.regionName = @"ap-chengdu";
[request setFinishBlock:^(id _Nullable outputObject, NSError * _Nullable error) {
/// error 为空则表示成功
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteBucketPolicy:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

跨域访问

最近更新时间：2022-05-25 15:22:40

简介

本文档提供关于跨域访问的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域名访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域名访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域名访问配置信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置跨域配置

功能说明

设置指定存储桶的跨域名访问配置信息（PUT Bucket cors）。

示例代码

Objective-C

```
QCloudPutBucketCORSRequest* putCORS = [QCloudPutBucketCORSRequest new];
QCloudCORSConfiguration* cors = [QCloudCORSConfiguration new];
QCloudCORSRule* rule = [QCloudCORSRule new];
// 配置规则 ID
rule.identifier = @"rule1";
// 跨域请求可以使用的 HTTP 请求头部，支持通配符 *
rule.allowedHeader = @[@"origin",@"host",@"accept",
@"content-type",@"authorization"];
rule.exposeHeader = @"ETag";
// 跨域请求允许的 HTTP 操作，例如：GET，PUT，HEAD，POST，DELETE
rule.allowedMethod = @[@"GET",@"PUT",@"POST", @"DELETE", @"HEAD"];
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 跨域请求得到结果的有效期
rule.maxAgeSeconds = 3600;
// 允许的访问来源，支持通配符 *，格式为：协议://域名[:端口]
rule.allowedOrigin = @"http://cloud.tencent.com";
cors.rules = @[rule];
putCORS.corsConfiguration = cors;
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
putCORS.bucket = @"examplebucket-1250000000";
[putCORS setFinishBlock:^(id outputObject, NSError *error) {
// 可以从 outputObject 中获取服务器返回的 header 信息
NSDictionary * result = (NSDictionary *)outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketCORS:putCORS];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let putBucketCorsReq = QCloudPutBucketCORSRequest.init();
let corsConfig = QCloudCORSConfiguration.init();
let rule = QCloudCORSRule.init();
// 配置规则的 ID
rule.identifier = "rule1";
// 跨域请求可以使用的 HTTP 请求头部，支持通配符 *
rule.allowedHeader = ["origin","host","accept","content-type","authorization"];
rule.exposeHeader = "Etag";
// 跨域请求允许的 HTTP 操作，例如：GET，PUT，HEAD，POST，DELETE
rule.allowedMethod = ["GET","PUT","POST", "DELETE", "HEAD"];
// 跨域请求得到结果的有效期
rule.maxAgeSeconds = 3600;
// 允许的访问来源，支持通配符 *，格式为：协议://域名[:端口]
rule.allowedOrigin = "*";
corsConfig.rules = [rule];
putBucketCorsReq.corsConfiguration = corsConfig;
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
putBucketCorsReq.bucket = "examplebucket-1250000000";
putBucketCorsReq.finishBlock = {(result,error) in
if let result = result {
// 可以从 result 中获取服务器返回的 header 信息
} else {
```

```
print(error!)
}
}
QCloudCOSXMLService.defaultCOSXML().putBucketCORS(putBucketCorsReq);
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

查询跨域配置

功能说明

查询指定存储桶的跨域名访问配置信息（GET Bucket cors）。

示例代码

Objective-C

```
QCloudGetBucketCORSRequest* corsReqeust = [QCloudGetBucketCORSRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
corsReqeust.bucket = @"examplebucket-1250000000";
[corsReqeust setFinishBlock:^(QCloudCORSConfiguration * _Nonnull result,
NSError * _Nonnull error) {
// 跨域规则列表
NSArray<QCloudCORSRule*> *rules = result.rules;

}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketCORS:corsReqeust];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let getBucketCorsRes = QCloudGetBucketCORSRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
getBucketCorsRes.bucket = "examplebucket-1250000000";
```

```
getBucketCorsRes.setFinish { (corsConfig, error) in
if let corsConfig = corsConfig {
// 跨域规则列表
let rules = corsConfig.rules
} else {
print(error!)
}
}
QCloudCOSXMLService.defaultCOSXML().getBucketCORS(getBucketCorsRes);
```

说明：
更多完整示例，请前往 GitHub 查看。

删除跨域配置

功能说明

删除指定存储桶的跨域名访问配置（DELETE Bucket cors）。

示例代码

Objective-C

```
QCloudDeleteBucketCORSRequest* deleteCORS = [QCloudDeleteBucketCORSRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
deleteCORS.bucket = @"examplebucket-1250000000";
[deleteCORS setFinishBlock:^(id outputObject, NSError *error) {
// 可以从 outputObject 中获取服务器返回的 header 信息
NSDictionary* info = (NSDictionary *) outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteBucketCORS:deleteCORS];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let deleteBucketCorsRequest = QCloudDeleteBucketCORSRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
deleteBucketCorsRequest.bucket = "examplebucket-1250000000";
deleteBucketCorsRequest.finishBlock = {(result,error) in
if let result = result {
// 可以从 result 中获取服务器返回的 header 信息
} else {
print(error!)
}
}
QCloudCOSXMLService.defaultCOSXML().deleteBucketCORS(deleteBucketCorsRequest);
```

说明：
更多完整示例，请前往 GitHub 查看。

自定义域名

最近更新时间：2022-05-25 15:22:40

简介

本文档提供关于自定义域名的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket domain	设置自定义域名	设置存储桶的自定义域名信息
GET Bucket domain	查询自定义域名	查询存储桶的自定义域名信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置自定义域名

功能说明

PUT Bucket domain 用于为存储桶配置自定义域名。

示例代码

Objective-C

```
QCloudPutBucketDomainRequest *req = [QCloudPutBucketDomainRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
req.bucket = @"examplebucket-1250000000";
QCloudDomainConfiguration *config = [QCloudDomainConfiguration new];
QCloudDomainRule *rule = [QCloudDomainRule new];
// 源站状态，可选 QCloudDomainStatueEnabled、 QCloudDomainStatueDisabled
rule.status = QCloudDomainStatueEnabled;
// 域名信息
rule.name = @"www.baidu.com";
// 替换已存在的配置、有效值CNAME/TXT 填写则强制校验域名所有权之后，再下发配置
rule.replace = QCloudCOSDomainReplaceTypeTxt;
rule.type = QCloudCOSDomainTypeRest;
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 规则描述集合的数组
config.rules = @[rule];
// 域名配置的规则
req.domain = config;
[req setFinishBlock:^(id outputObject, NSError *error) {
// outputObject 包含所有的响应 http 头部
NSDictionary* info = (NSDictionary *) outputObject;

}];
[[QCloudCOSXMLService defaultCOSXML]PutBucketDomain:req];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudPutBucketDomainRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
let config = QCloudDomainConfiguration.init();
let rule = QCloudDomainRule.init();
// 开启状态，可选 .enabled, .disabled
rule.status = .enabled;
rule.name = "www.baidu.com";
// 替换已存在的配置、有效值CNAME/TXT 填写则强制校验域名所有权之后，再下发配置
rule.replace = .txt;
rule.type = .rest;
// 规则描述集合的数组
config.rules = [rule];
// 域名配置的规则
req.domain = config;
req.finishBlock = {(result,error) in
if let result = result {
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putBucketDomain(req);
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

返回错误码说明

该请求可能会发生的一些常见的特殊错误如下：

状态码	说明
HTTP 409 Conflict	该域名记录已存在，且请求中没有设置强制覆盖。或者该域名记录不存在，且请求中设置了强制覆盖
HTTP 451 Unavailable For Legal Reasons	该域名是中国境内域名，并且没有备案

查询自定义域名

功能说明

GET Bucket domain 用于查询存储桶的自定义域名信息。

示例代码

Objective-C

```
QCloudGetBucketDomainRequest *getReq = [QCloudGetBucketDomainRequest new];
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
getReq.bucket = @"examplebucket-1250000000";
[getReq setFinishBlock:^(QCloudDomainConfiguration * _Nonnull result,
NSError * _Nonnull error) {
// 规则描述集合的数组
NSArray *rules = result.rules;
}];
[[QCloudCOSXMLService defaultCOSXML]GetBucketDomain:getReq];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let req = QCloudGetBucketDomainRequest.init();
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
req.bucket = "examplebucket-1250000000";
req.finishBlock = {(result,error) in
if let result = result {
// result 包含源站信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getBucketDomain(req);
```

说明：
更多完整示例，请前往 GitHub 查看。

返回参数说明

参数名称	描述	类型
x-cos-domain-txt-verification	域名校验信息，该字段是一个 MD5 校验值，原串格式为：`cos[Region][BucketName-APPID][BucketCreateTime]`，其中 Region 为存储桶所在地域，BucketCreateTime 为存储桶 GMT 创建时间	String

访问控制

最近更新时间：2023-03-16 16:03:13

简介

本文档提供关于存储桶、对象的访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

存储桶 ACL

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶的访问权限控制列表（ACL）
GET Bucket acl	查询存储桶 ACL	查询指定存储桶的访问权限控制列表（ACL）

对象 ACL

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表
GET Object acl	查询对象 ACL	查询对象的访问控制列表

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

存储桶 ACL

设置存储桶 ACL

功能说明

设置指定存储桶的访问权限控制列表（ACL）。

示例代码

Objective-C

```
QCloudPutBucketACLRequest* putACL = [QCloudPutBucketACLRequest new];

// 授予权限的账号 ID
NSString* uin = @"100000000001";
NSString *ownerIdentifier = [NSString stringWithFormat:@"qcs::cam::uin/%@:uin/%@"
, uin,uin];
NSString *grantString = [NSString stringWithFormat:@"id=\"%@\"",ownerIdentifier];

// 赋予被授权者读写权限
putACL.grantFullControl = grantString;

// 赋予被授权者读权限
putACL.grantRead = grantString;

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
putACL.bucket = @"examplebucket-1250000000";

[putACL setFinishBlock:^(id outputObject, NSError *error) {
// 可以从 outputObject 中获取服务器返回的 header 信息
NSDictionary * result = (NSDictionary *)outputObject;

}];
// 设置acl
[[QCloudCOSXMLService defaultCOSXML] PutBucketACL:putACL];
```

说明：

更多完整示例，请前往 GitHub 查看。

Swift

```
let putBucketACLReq = QCloudPutBucketACLRequest.init();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket
putBucketACLReq.bucket = "examplebucket-1250000000";

// 授予权限的账号 ID
let appTD = "100000000001";
let ownerIdentifier = "qcs::cam::uin/\(appTD):uin/\(appTD)";
let grantString = "id=\"\(ownerIdentifier)\"";
// 赋予被授权者写权限
```

```
putBucketACLReq.grantWrite = grantString;

// 赋予被授权者读权限
putBucketACLReq.grantRead = grantString;

// 赋予被授权者读写权限 grantFullControl == grantRead + grantWrite
putBucketACLReq.grantFullControl = grantString;

putBucketACLReq.finishBlock = {(result,error) in
if let result = result {
// 可以从 result 中获取服务器返回的 header 信息
} else {
print(error!)
}
}
QCloudCOSXMLService.defaultCOSXML().putBucketACL(putBucketACLReq);
```

说明：
更多完整示例，请前往 GitHub 查看。

查询存储桶 ACL

功能说明

查询指定存储桶的访问权限控制列表（ACL）。

示例代码

Objective-C

```
QCloudGetBucketACLRequest* getBucketACl = [QCloudGetBucketACLRequest new];

// 存储桶名称，格式：BucketName-APPID
getBucketACl.bucket = @"examplebucket-1250000000";

[getBucketACl setFinishBlock:^(QCloudACLPolicy * _Nonnull result,
NSError * _Nonnull error) {
// 被授权者与权限的信息
QCloudAccessControlList *acl = result.accessControlList;
}];

[[QCloudCOSXMLService defaultCOSXML] GetBucketACL:getBucketACl];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getBucketACLReq = QCloudGetBucketACLRequest.init();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
getBucketACLReq.bucket = "examplebucket-1250000000";

getBucketACLReq.setFinish { (result, error) in
if let result = result {
// ACL 授权信息
let acl = result.accessControlList;
} else {
print(error!)
}
}
QCloudCOSXMLService.defaultCOSXML().getBucketACL(getBucketACLReq)
```

说明：
更多完整示例，请前往 GitHub 查看。

对象 ACL

设置对象 ACL

功能说明

设置存储桶中某个对象的访问控制列表（ACL）。

示例代码

Objective-C

```
QCloudPutObjectACLRequest* request = [QCloudPutObjectACLRequest new];

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";
```

```
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";

NSString *grantString = [NSString stringWithFormat:@"id=\"%@\"",@"100000000001"];

// grantFullControl 等价于 grantRead + grantWrite
// 赋予被授权者读写权限。
request.grantFullControl = grantString;
// 赋予被授权者读权限。
request.grantRead = grantString;

[request setFinishBlock:^(id outputObject, NSError *error) {
// 可以从 outputObject 中获取 response 中 etag 或者自定义头部等信息
NSDictionary* info = (NSDictionary *) outputObject;
}];

[[QCloudCOSXMLService defaultCOSXML] PutObjectACL:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let putObjectACl = QCloudPutObjectACLRequest.init();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
putObjectACl.bucket = "examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
putObjectACl.object = "exampleobject";
let grantString = "id=\"100000000001\"";

// grantFullControl 等价于 grantRead + grantWrite
putObjectACl.grantFullControl = grantString;
// 赋予被授权者读权限。
putObjectACl.grantRead = grantString;


putObjectACl.finishBlock = {(result,error)in
if let result = result {
```

```
// result 包含响应的 header 信息
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().putObjectACL(putObjectACl);
```

说明：
更多完整示例，请前往 GitHub 查看。

查询对象 ACL

功能说明

查询对象的访问控制列表。

示例代码

Objective-C

```
QCloudGetObjectACLRequest *request = [QCloudGetObjectACLRequest new];

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
request.object = @"exampleobject";

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
request.bucket = @"examplebucket-1250000000";

__block QCloudACLPolicy* policy;
[request setFinishBlock:^(QCloudACLPolicy * _Nonnull result,
NSError * _Nonnull error) {

policy = result;
// result.accessControlList; 被授权者与权限的信息
// result.owner; 持有者的信息
}];

[[QCloudCOSXMLService defaultCOSXML] GetObjectACL:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let getObjectACL = QCloudGetObjectACLRequest.init();

// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
getObjectACL.bucket = "examplebucket-1250000000";

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
getObjectACL.object = "exampleobject";
getObjectACL.setFinish { (result, error) in
if let result = result {
// 对象授权信息
let acl = result.accessControlList
} else {
print(error!);
}
}
QCloudCOSXMLService.defaultCOSXML().getObjectACL(getObjectACL);
```

说明：
更多完整示例，请前往 GitHub 查看。

防盗链

最近更新时间：2021-12-21 15:07:08

简介

本文档提供关于存储桶 Referer 白名单或者黑名单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket referer	设置存储桶 Referer	设置存储桶 Referer 白名单或者黑名单
GET Bucket referer	查询存储桶 Referer	查询存储桶 Referer 白名单或者黑名单

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

设置存储桶 Referer

功能说明

设置指定存储桶的 Referer 白名单或者黑名单（PUT Bucket referer）。

> 注意：
> COS iOS SDK 版本需要大于等于 v5.9.6。

请求示例

Objective-C

```
QCloudPutBucketRefererRequest* request = [QCloudPutBucketRefererRequest new];
// 防盗链类型，枚举值：Black-List、White-List
request.refererType = QCloudBucketRefererTypeBlackList;
// 是否开启防盗链，枚举值：Enabled、Disabled
request.status = QCloudBucketRefererStatusEnabled;
// 是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny
request.configuration = QCloudBucketRefererConfigurationDeny;
```

```
// 生效域名列表， 支持多个域名且为前缀匹配， 支持带端口的域名和 IP， 支持通配符*，做二级域名或多级域名的通配
request.domainList = @[@"*.com",@"*.qq.com"];
// 存储桶名称，格式为 BucketName-APPID
request.bucket = @"examplebucket-1250000000";
[request setFinishBlock:^(id outputObject, NSError *error) {
if (error){
// 添加防盗链失败
}else{
// 添加防盗链失败
}
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketReferer:request];
```

说明：

更多完整示例，请前往 GitHub 查看。

Swift

```
let request = QCloudPutBucketRefererRequest.init();
// 防盗链类型，枚举值：Black-List、White-List
request.refererType = QCloudBucketRefererType.blackList;
// 是否开启防盗链，枚举值：Enabled、Disabled
request.status = QCloudBucketRefererStatus.enabled;
// 是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny
request.configuration = QCloudBucketRefererConfiguration.allow;
// 生效域名列表， 支持多个域名且为前缀匹配， 支持带端口的域名和 IP， 支持通配符*，做二级域名或多级域名的通配
request.domainList = ["*.com","*.qq.com"];
// 存储桶名称，格式为 BucketName-APPID
request.bucket = "examplebucket-1250000000";
request.finishBlock = {(result,error) in
if (error != nil){
// 添加防盗链失败
}else{
// 添加防盗链失败
}
}
QCloudCOSXMLService.defaultCOSXML().putBucketReferer(request);
```

说明：

更多完整示例，请前往 GitHub 查看。

查询存储桶 Referer

功能说明

查询指定存储桶 Referer 白名单或者黑名单（GET Bucket referer）。

注意：
COS iOS SDK 版本需要大于等于 v5.9.6。

请求示例

Objective-C

```
QCloudGetBucketRefererRequest* request = [QCloudGetBucketRefererRequest new];
// 存储桶名称，格式为 BucketName-APPID
request.bucket = @"examplebucket-1250000000";
[request setFinishBlock:^(QCloudBucketRefererInfo * outputObject, NSError *error)
{
// outputObject 请求到的防盗链，详细字段请查看api文档或者SDK源码
// QCloudBucketRefererInfo 类；
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketReferer:request];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let request = QCloudGetBucketRefererRequest.init();
// 存储桶名称，格式为 BucketName-APPID
request.bucket = "examplebucket-1250000000";
request.finishBlock = {(result,error) in
// outputObject 请求到的防盗链，详细字段请查看api文档或者SDK源码
// QCloudBucketRefererInfo 类；
}
QCloudCOSXMLService.defaultCOSXML().getBucketReferer(request);
```

说明：
更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

图片处理
基础图片处理

最近更新时间：2022-04-06 16:55:21

简介

本文档提供关于基础图片处理的相关的 API 概览以及 SDK 示例代码。

服务	功能	说明
基础图片处理服务	缩放	等比缩放、设定目标宽高缩放等多种方式
	裁剪	普通裁剪、缩放裁剪、内切圆、人脸智能裁剪
	旋转	自适应旋转、普通旋转
	格式转换	格式转换、GIF 格式优化、渐进显示
	质量变换	针对 JPG 和 WEBP 图片进行质量变换
	高斯模糊	对图片进行模糊处理
	锐化	对图片进行锐化处理
	添加水印	图片水印、文字水印
	获取图片信息	基本信息、EXIF 信息、主色调
	去除元信息	包括 EXIF 信息
	快速缩略模板	快速实现图片格式转换、缩略、剪裁等功能，生成缩略图

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

上传时使用图片处理

下面示例展示了如何在上传图片时自动实现图片处理。

版权所有：腾讯云计算（北京）有限责任公司

图片上传完成后，COS 会存储原始图片和已处理过的图片。后续用户可以通过普通的下载请求获取处理结果。

示例代码

Objective-C

```
QCloudPutObjectWatermarkRequest* put = [QCloudPutObjectWatermarkRequest new];
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = @"exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
put.bucket = @"examplebucket-1250000000";
put.body = [@"123456789" dataUsingEncoding:NSUTF8StringEncoding];
QCloudPicOperations * op = [[QCloudPicOperations alloc]init];
// 是否返回原图信息。0表示不返回原图信息，1表示返回原图信息，默认为0
op.is_pic_info = NO;
QCloudPicOperationRule * rule = [[QCloudPicOperationRule alloc]init];
// 处理结果的文件路径名称，如以/开头，则存入指定文件夹中，否则，存入原图文件存储的同目录
rule.fileid = @"test";
// 盲水印文字，需要经过 URL 安全的 Base64 编码。当 type 为3时必填，type
rule.text = @"123"; // 水印文字只能是 [a-zA-Z0-9]
// 盲水印类型，有效值：1 半盲；2 全盲；3 文字
rule.type = QCloudPicOperationRuleText;
op.rule = @[rule];
put.picOperations = op;
[put setFinishBlock:^(id outputObject, NSError *error) {

}];
[[QCloudCOSXMLService defaultCOSXML] PutWatermarkObject:put];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let put = QCloudPutObjectWatermarkRequest<AnyObject>();
// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = "exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
put.bucket = "examplebucket-1250000000";
put.body = "123456789".data(using: .utf8)! as NSData;
let op = QCloudPicOperations.init();
```

```
// 是否返回原图信息。0表示不返回原图信息，1表示返回原图信息，默认为0
op.is_pic_info = false;
let rule = QCloudPicOperationRule.init();
// 处理结果的文件路径名称，如以/开头，则存入指定文件夹中，否则，存入原图文件存储的同目录
rule.fileid = "test";
// 盲水印文字，需要经过 URL 安全的 Base64 编码。当 type 为3时必填，type 为1或2时无效。
rule.text = "123";
// 盲水印类型，有效值：1 半盲；2 全盲；3 文字
rule.type = .text;
op.rule = [rule];
put.picOperations = op;
put.setFinish { (outoutObject, error) in

};
QCloudCOSXMLService.defaultCOSXML().putWatermarkObject(put);
```

说明：

更多完整示例，请前往 GitHub 查看。

云上数据处理

下面示例展示了如何在上传图片时自动实现图片处理。

图片上传完成后，COS 会存储原始图片和已处理过的图片。后续用户可以通过普通的下载请求获取处理结果。

示例代码

Objective-C

```
QCloudCICloudDataOperationsRequest* put = [QCloudCICloudDataOperationsRequest ne
w];

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = @"exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tence
nt.com/cos5/bucket
put.bucket = @"examplebucket-1250000000";

QCloudPicOperations * op = [[QCloudPicOperations alloc]init];

// 是否返回原图信息。0表示不返回原图信息，1表示返回原图信息，默认为0
op.is_pic_info = NO;
```

```
QCloudPicOperationRule * rule = [[QCloudPicOperationRule alloc]init];

// 处理结果的文件路径名称，如以/开头，则存入指定文件夹中，否则，存入原图文件存储的同目录
rule.fileid = @"test";

// 盲水印文字，需要经过 URL 安全的 Base64 编码。当 type 为3时必填，type
rule.text = @"123"; // 水印文字只能是 [a-zA-Z0-9]

// 盲水印类型，有效值：1 半盲；2 全盲；3 文字
rule.type = QCloudPicOperationRuleText;
op.rule = @[rule];
put.picOperations = op;
[put setFinishBlock:^(QCloudPutObjectWatermarkResult *outputObject, NSError *erro
r) {

}];
[[QCloudCOSXMLService defaultCOSXML] CloudDataOperations:put];
```

说明：
更多完整示例，请前往 GitHub 查看。

Swift

```
let put = QCloudCICloudDataOperationsRequest<AnyObject>();

// 对象键，是对象在 COS 上的完整路径，如果带目录的话，格式为 "video/xxx/movie.mp4"
put.object = "exampleobject";
// 存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.
tencent.com/cos5/bucket

put.bucket = "examplebucket-1250000000";
let op = QCloudPicOperations.init();

// 是否返回原图信息。0表示不返回原图信息，1表示返回原图信息，默认为0
op.is_pic_info = false;

let rule = QCloudPicOperationRule.init();

// 处理结果的文件路径名称，如以/开头，则存入指定文件夹中，否则，存入原图文件存储的同目录

rule.fileid = "test";

// 盲水印文字，需要经过 URL 安全的 Base64 编码。当 type 为3时必填，type 为1或2时无效。
```

```
rule.text = "123";

// 盲水印类型，有效值：1 半盲；2 全盲；3 文字
rule.type = .text;

op.rule = [rule];
put.picOperations = op;
put.setFinish { (outoutObject, error) in

};
QCloudCOSXMLService.defaultCOSXML().cloudDataOperations(put);
```

说明：
更多完整示例，请前往 GitHub 查看。

内容识别
语音识别
队列接口

最近更新时间：2022-09-26 16:20:41

简介

本文档提供关于查询语音识别队列、更新语音识别队列 相关的 API 概览以及 SDK 示例代码。

API	说明
查询语音识别队列	接口用于查询语音识别队列。
更新语音识别队列	接口用于更新语音识别队列。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

查询语音识别队列

功能说明

接口用于查询语音识别队列。

> 注意：
> COS iOS SDK 版本需要大于等于 v6.1.3。

示例代码

Objective-C

```
QCloudGetAudioDiscernTaskQueueRequest * request = [[QCloudGetAudioDiscernTaskQueu
eRequest alloc]init];
// 存储桶名称，格式为 BucketName-APPID
```

```
request.bucket = @"examplebucket-1250000000";
request.regionName = @"regionName";
// 队列 ID，以“,”符号分割字符串
request.queueIds = @"1,2,3";
// 1. Active 表示队列内的作业会被语音识别服务调度执行
// 2. Paused 表示队列暂停，作业不再会被语音识别服务调度执行，队列内的所有作业状态维持在暂停状态，已经处于识别中的任务将继续执行，不受影响
request.state = 1;
request.finishBlock = ^(QCloudAudioAsrqueueResult * outputObject, NSError *error)
{
// outputObject 审核结果 包含用于查询的job id，详细字段请查看api文档或者SDK源码
// QCloudAudioAsrqueueResult 类；
};
[[QCloudCOSXMLService defaultCOSXML] GetAudioDiscernTaskQueue:request];
```

说明：

更多完整示例，请前往 GitHub 查看。

Swift

```
let request = QCloudGetAudioDiscernTaskQueueRequest.init();
// 存储桶名称，格式为 BucketName-APPID
request.bucket = "examplebucket-1250000000";
request.regionName = "regionName";
// 队列 ID，以“,”符号分割字符串
request.queueIds = "1,2,3";
// 1. Active 表示队列内的作业会被语音识别服务调度执行
// 2. Paused 表示队列暂停，作业不再会被语音识别服务调度执行，队列内的所有作业状态维持在暂停状态，已经处于识别中的任务将继续执行，不受影响
request.state = 1;
request.setFinish { outputObject, error in
// outputObject 审核结果 包含用于查询的job id，详细字段请查看api文档或者SDK源码
// QCloudAudioAsrqueueResult 类；
};
QCloudCOSXMLService.defaultCOSXML().getAudioDiscernTaskQueue(request);
```

说明：

更多完整示例，请前往 GitHub 查看。

更新语音识别队列

功能说明

接口用于更新语音识别队列。

> 注意：
> COS iOS SDK 版本需要大于等于 v6.1.3。

示例代码

Objective-C

```
QCloudUpdateAudioDiscernTaskQueueRequest * request = [[QCloudUpdateAudioDiscernTa
skQueueRequest alloc]init];
// 存储桶名称，格式为 BucketName-APPID
request.bucket = @"examplebucket-1250000000";
request.regionName = @"regionName";
// 模板名称
request.name = @"name";
// 1. Active 表示队列内的作业会被语音识别服务调度执行
// 2. Paused 表示队列暂停，作业不再会被语音识别服务调度执行，队列内的所有作业状态维持在暂停状
态，已经处于识别中的任务将继续执行，不受影响
request.state = 1;
// 管道 ID
request.queueID = @"queueID";
// 其他更多参数请查看sdk文档或源码注释
request.finishBlock = ^(QCloudAudioAsrqueueUpdateResult * outputObject, NSError *
error) {
// outputObject 详细字段请查看api文档或者SDK源码
// QCloudAudioAsrqueueUpdateResult 类；
};
[[QCloudCOSXMLService defaultCOSXML] UpdateAudioDiscernTaskQueue:request];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let request = QCloudUpdateAudioDiscernTaskQueueRequest.init();
// 存储桶名称，格式为 BucketName-APPID
request.bucket = "examplebucket-1250000000";
request.regionName = "regionName";
// 模板名称
request.name = "name";
// 1. Active 表示队列内的作业会被语音识别服务调度执行
// 2. Paused 表示队列暂停，作业不再会被语音识别服务调度执行，队列内的所有作业状态维持在暂停状态，已经处于识别中的任务将继续执行，不受影响
request.state = 1;
// 管道 ID
request.queueID = "queueID";
// 其他更多参数请查看sdk文档或源码注释
request.setFinish { outputObject, error in
// outputObject 详细字段请查看api文档或者SDK源码
// QCloudAudioAsrqueueUpdateResult 类；
};
QCloudCOSXMLService.defaultCOSXML().updateAudioDiscernTaskQueue(request);
```

说明：

更多完整示例，请前往 GitHub 查看。

设置自定义头部

最近更新时间：2020-08-25 17:48:41

简介

本文档主要介绍 SDK 如何在请求时携带自定义头部。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

功能说明

COS 在上传对象时可以携带以 `x-cos-meta-` 开头的自定义头部，包括用户自定义元数据头部后缀和用户自定义元数据信息，这些头部将作为对象元数据保存。

如果您开通了万象服务，可以携带 `Pic-Operations` 头部，实现后台自动图片处理，详细的 API 说明请参考 数据万象持久化。

示例代码

Objective-C

```
request.customHeaders[@"custom-key"] = @"custom-value";
```

说明：

更多完整示例，请前往 GitHub 查看。

Swift

```
request.customHeaders["custom-key"] = "custom-value";
```

说明：

更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

设置访问域名（CDN/全球加速）

最近更新时间：2022-01-06 16:15:03

简介

本文档提供关于如何使用非默认域名请求 COS 服务。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

CDN 默认加速域名

关于如何开启默认加速域名请参考 开启默认 CDN 加速域名。

以下代码展示了如何使用默认加速域名访问 COS 服务。

示例代码

Objective-C

```
QCloudCOSXMLEndPoint *endpoint = [[QCloudCOSXMLEndPoint alloc] init];
endpoint.suffix = @"file.myqcloud.com";
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let endpoint = QCloudCOSXMLEndPoint();
endpoint.suffix = "file.myqcloud.com";
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

CDN 自定义加速域名

关于如何开启 CDN 自定义加速域名请参考 开启自定义 CDN 加速域名。

以下代码展示了如何使用自定义加速域名访问 COS 服务。

示例代码

Objective-C

```
QCloudCOSXMLEndPoint *endpoint = [[QCloudCOSXMLEndPoint alloc] initWithLiteralURL:[NSURL URLWithString:@"exampledomain.com"]];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let endpoint = QCloudCOSXMLEndPoint.init(literalURL: NSURL.init(string: "exampledomain.com") as URL?);
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

自定义源站域名

关于如何设置自定义源站域名请参考 自定义源站域名。

以下代码展示了如何使用自定义源站域名访问 COS 服务。

示例代码

Objective-C

```
NSString *customDomain = @"exampledomain.com"; // 自定义加速域名
QCloudCOSXMLEndPoint *endpoint = [[QCloudCOSXMLEndPoint alloc] initWithLiteralURL:[NSURL URLWithString:customDomain]];
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let endpoint = QCloudCOSXMLEndPoint.init(literalURL: NSURL.init(string: "exampledomain.com") as URL?);
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

全球加速域名

关于全球加速功能请参考 全球加速功能概述。

以下代码展示了如何使用全球加速域名访问 COS 服务。

示例代码

Objective-C

```
QCloudCOSXMLEndPoint *endpoint = [[QCloudCOSXMLEndPoint alloc]init];
endpoint.suffix = @"cos.accelerate.myqcloud.com";
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。
> **Swift**

```
let endpoint = QCloudCOSXMLEndPoint();
endpoint.suffix = "cos.accelerate.myqcloud.com";
```

说明：
更多完整示例，请前往 GitHub 查看。

异常处理

最近更新时间：2023-05-24 14:30:13

当 SDK 请求失败的时候，返回的 error 将不为空，并且包括了错误码、错误描述和其它一些调试必备的信息，以帮助开发者快速解决问题。返回错误码（封装在返回的 error 里）主要包括两类：客户端错误和服务端错误。

客户端错误

- 对于设备本身因为网络原因产生的错误码，都是负数并且是四位数，例如-1001，这类错误码由苹果公司定义，可以参考 Foundation 框架中的 NSURLError.h 头文件内的定义，或者是 苹果官方文档说明。
- 对于腾讯云 SDK 网络层本地客户端自定义错误：主要是指网络异常、证书无效、参数校验失败等，如下表所示：

错误码	错误信息	错误描述
10000	InvalidArgument	参数错误
10001	InvalidCredentials	证书无效
10004	UnsupportOperation	无法支持的操作
20001	InvalidArgument	服务器返回了不合法的数据
20004	PoorNetwork	数据完整性校验失败
30000	UserCancelled	用户取消
30002	AlreadyFinished	任务已完成

服务端错误

对于 COS 返回的错误码，是基于 HTTP 的状态码而来的，也就是404、503这类。对于这类错误码，请参见 错误码 文档寻求解决方案。您也可以使用自助诊断工具，调试报错代码，快速定位问题。

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。

版权所有：腾讯云计算（北京）有限责任公司

COS 自助诊断工具

点击自助诊断

输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。

3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**，请您耐心等待几分钟，便能看到相应的智能诊断结果。

Java SDK
快速入门

最近更新时间：2023-05-23 16:11:02

下载与安装

相关资源

- 对象存储（Cloud Object Storage，COS）服务的 XML Java SDK 源码下载地址：XML Java SDK。
- SDK 快速下载地址：XML Java SDK。
- 示例 Demo 下载地址：COS XML Java SDK 示例。
- SDK 文档中的所有示例代码请参见 SDK 代码示例。
- SDK 更新日志请参见 ChangeLog。
- SDK 常见问题请参见：Java SDK 常见问题。

> 说明：
>
> - 为了避免不必要的繁琐代码排查与程序定位问题，如果您初次使用 COS 或者数据量不大，推荐使用更加友好、图形化界面的 控制台、COSBrowser，或者可批量操作的命令行工具 COSCLI。
> - 如果您在使用 XML 版本 SDK 时遇到函数或方法不存在等错误，请先将 XML 版本 SDK 升级到最新版再重试。

环境依赖

- SDK 支持 JDK 1.8 及以上版本，您可以通过命令 java -version 查看 Java 版本。
- JDK 安装方式请参见 Java 安装与配置。

> 说明：
>
> - 关于文章中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参见 COS 术语信息。
> - COS Java SDK 中的常见类所在包分别为：

- 客户端配置相关类在 com.qcloud.cos.* 包下。
- 权限相关类在 com.qcloud.cos.auth.* 子包下。
- 异常相关类在 com.qcloud.cos.exception.* 子包下。

版权所有：腾讯云计算（北京）有限责任公司

- 请求相关类在 com.qcloud.cos.model.* 子包下。
- 地域相关类在 com.qcloud.cos.region.* 子包下。
- 高级 API 接口在 com.qcloud.cos.transfer.* 子包下。

安装 SDK

用户可以通过 maven 和源码两种方式安装 Java SDK：

- maven 安装
  在 maven 工程的 pom.xml 文件中添加相关依赖，内容如下：

```
<dependency>
<groupId>com.qcloud</groupId>
<artifactId>cos_api</artifactId>
<version>5.6.133</version>
</dependency>
```

- 源码安装
  从 Github XML Java SDK 或 快速下载地址 下载源码，通过 maven 导入。例如 eclipse，依次选择 **File > Import > maven > Existing Maven Projects**。

卸载 SDK

通过删除 pom 依赖或源码即可卸载 SDK。

开始使用

正式使用前，先别急着书写第一行代码，请先花两分钟做下面的测试，确保发送的请求能够到达 COS 服务，这可以帮助您确认是否遇到了大部分用户在使用 Java SDK 时碰到的问题。
下面测试中假设您尝试访问广州地域的 COS 服务。

使用 ping 测试

```
[root@VM_centos /data/home/]# ping cos.ap-guangzhou.myqcloud.com
PING cos.ap-guangzhou.myqcloud.com (9.*.*.*) xx(xx) bytes of data.
64 bytes from 9.*.*.* (9.*.*.*): icmp_seq=1 ttl=xx time=0.xxx ms
64 bytes from 9.*.*.* (9.*.*.*): icmp_seq=2 ttl=xx time=0.xxx ms
64 bytes from 9.*.*.* (9.*.*.*): icmp_seq=3 ttl=xx time=0.xxx ms
64 bytes from 9.*.*.* (9.*.*.*): icmp_seq=4 ttl=xx time=0.xxx ms
64 bytes from 9.*.*.* (9.*.*.*): icmp_seq=5 ttl=xx time=0.xxx ms
64 bytes from 9.*.*.* (9.*.*.*): icmp_seq=6 ttl=xx time=0.xxx ms
```

版权所有：腾讯云计算（北京）有限责任公司

```
64 bytes from 9.*.*.* (9.*.*.*): icmp_seq=7 ttl=xx time=0.xxx ms
^C
--- cos.ap-guangzhou.myqcloud.com ping statistics ---
x packets transmitted, x received, 0% packet loss, time xxxms
rtt min/avg/max/mdev = 0.xxx/0.xxx/0.xxx/0.xxx ms
```

如果显示类似上面结果，则基本网络连通与名字服务都是正常的。如果返回其他异常结果，请优先排查环境网络问题或者联系本地网络管理员，确认全部正常后，再继续下一步。

如果是在 Windows 下，也可以单击**开始 （或快捷键：Win+R）> 运行 （输入 cmd）> 确定（或按 Enter 键）**，输入命令 `ping cos.ap-guangzhou.myqcloud.com` 并回车进行测试。

使用 curl 测试

如果使用 HTTP 访问：

```
[root@VM_centos /data/home/]# curl http://cos.ap-guangzhou.myqcloud.com -v
* About to connect() to cos.ap-guangzhou.myqcloud.com port 80 (#0)
* Trying 9.*.*.*...
* Connected to cos.ap-guangzhou.myqcloud.com (9.*.*.*) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/*.*.0
> Host: cos.ap-guangzhou.myqcloud.com
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Content-Type: application/xml
< Content-Length: XXX
< Connection: keep-alive
< Date: XXX XXX GMT
< Server: tencent-cos
< x-cos-request-id: NWE2MWQ5MjZfMTBhYzM1MGFfMTA5ODVfMTVj****
<
<?xml version='1.0' encoding='utf-8' ?>
<Error>
<Code>AccessDenied</Code>
<Message>Check auth failure because signture empty.</Message>
<ServerTime>XXX XXX</ServerTime>
<Resource>cos.ap-guangzhou.myqcloud.com/</Resource>
<RequestId>NWE2MWQ5MjZfMTBhYzM1MGFfMTA5ODVfMTVj****==</RequestId>
</Error>

* Connection #0 to host cos.ap-guangzhou.myqcloud.com left intact
```

如果使用 HTTPS 访问：

```
[root@VM_centos /data/home/]# curl https://cos.ap-guangzhou.myqcloud.com -vk
* About to connect() to cos.ap-guangzhou.myqcloud.com port 443 (#0)
* Trying 9.*.*.*...
* Connected to cos.ap-guangzhou.myqcloud.com (9.*.*.*) port 443 (#0)
* Initializing NSS with certpath: XXXX
* skipping SSL peer certificate verification
* SSL connection using ******
* Server certificate:
* subject: CN=*.*.*.
* start date: XXX XXX GMT
* expire date: XXX XXX GMT
* common name: *.cos.ap-guangzhou.myqcloud.com
* issuer: XXX
> GET / HTTP/1.1
> User-Agent: curl/*.*.0
> Host: cos.ap-guangzhou.myqcloud.com
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Content-Type: application/xml
< Content-Length: XXX
< Connection: keep-alive
< Date: XXX XXX GMT
< Server: tencent-cos
< x-cos-request-id: NWE2MWQ5MjZfMTBhYzM1MGFfMTA5ODVfMTVj****
<
<?xml version='1.0' encoding='utf-8' ?>
<Error>
<Code>AccessDenied</Code>
<Message>Check auth failure because signture empty.</Message>
<ServerTime>XXX XXX</ServerTime>
<Resource>cos.ap-guangzhou.myqcloud.com/</Resource>
<RequestId>NWE2MWQ5MjZfMTBhYzM1MGFfMTA5ODVfMTVj****</RequestId>
</Error>

* Connection #0 to host cos.ap-guangzhou.myqcloud.com left intact
```

如果显示类似上面结果，则说明端口连接正常。如果返回其他异常结果，请优先排查环境网络问题或者联系本地网络管理员，确认全部正常后，再继续下一步。

在 Windows 环境下，某些版本可能需要额外安装 curl 工具才能测试。

下面为您介绍如何使用 COS Java SDK 完成一个基础操作，例如初始化客户端、创建存储桶、查询存储桶列表、上传对象、查询对象列表、下载对象和删除对象。对于示例中出现的类，在 IDE 中，您可以单击该类，查看该类的所有字段和函数定义，类中有较为详细的注释。

导入类名

COS Java SDK 的包名为 `com.qcloud.cos.*`，您可以通过 Eclipse 或者 Intellij 等 IDE 工具，导入程序运行所需要的类。

初始化客户端

> 注意：
>
> - 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
> - 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

在执行任何和 COS 服务相关请求之前，都需要先生成 COSClient 类的对象， COSClient 是调用 COS API 接口的对象。

> 注意：
> COSClient 是线程安全的类，允许多线程访问同一实例。因为实例内部维持了一个连接池，创建多个实例可能导致程序资源耗尽，**请确保程序生命周期内实例只有一个**，并在不再需要使用时，调用 shutdown 方法将其关闭。如果需要新建实例，请先将之前的实例关闭。

若您使用临时密钥初始化 COSClient（**推荐**），临时密钥生成和使用可参见 临时密钥生成及使用指引，参考示例如下：

```
// 1 传入获取到的临时密钥 (tmpSecretId, tmpSecretKey, sessionToken)
String tmpSecretId = "SECRETID";
String tmpSecretKey = "SECRETKEY";
String sessionToken = "TOKEN";
BasicSessionCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecret
Key, sessionToken);
// 2 设置 bucket 的地域
// clientConfig 中包含了设置 region, https(默认 http), 超时, 代理等 set 方法, 使用可参
见源码或者常见问题 Java SDK 部分
Region region = new Region("COS_REGION"); //COS_REGION 参数：配置成存储桶 bucket 的
实际地域, 例如 ap-beijing, 更多 COS 地域的简称请参见 https://intl.cloud.tencent.com/doc
ument/product/436/6224
ClientConfig clientConfig = new ClientConfig(region);
// 3 生成 cos 客户端
COSClient cosClient = new COSClient(cred, clientConfig);
```

您也可以使用永久密钥初始化 COSClient（**不推荐，有主密钥泄露风险，不建议在生产环境使用**），可以先在访问管理控制台中的 API 密钥管理 页面获取 APPId、SecretId、SecretKey，参考示例如下：

```
// 1 初始化用户身份信息（secretId, secretKey）。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
String secretId = System.getenv("secretId");//用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
String secretKey = System.getenv("secretKey");//用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 2 设置 bucket 的地域，COS 地域的简称请参见 https://intl.cloud.tencent.com/document/product/436/6224
// clientConfig 中包含了设置 region, https(默认 http), 超时, 代理等 set 方法, 使用可参见源码或者常见问题 Java SDK 部分。
Region region = new Region("COS_REGION");
ClientConfig clientConfig = new ClientConfig(region);
// 这里建议设置使用 https 协议
// 从 5.6.54 版本开始，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 3 生成 cos 客户端。
COSClient cosClient = new COSClient(cred, clientConfig);
```

ClientConfig 类为配置信息类，主要的成员如下：

成员名	设置方法	描述	类型
region	构造函数或 set 方法	存储桶所在的地域，COS 地域的简称请参见 地域和访问域名 文档	Region
httpProtocol	set 方法	请求所使用的协议，默认使用 HTTPS 协议与 COS 交互	HttpProtocol
signExpired	set 方法	请求签名的有效时间，单位：秒，默认为3600s	long
connectionTimeout	set 方法	连接 COS 服务的超时时间，单位：毫秒，默认为 30000ms	int
socketTimeout	set 方法	客户端读取数据的超时时间，单位：毫秒，默认为 30000ms	int
httpProxyIp	set 方法	代理服务器的 IP	String
httpProxyPort	set 方法	代理服务器的端口	int

创建存储桶

用户确定地域和存储桶名称后，参考如下示例创建存储桶：

```
String bucket = "examplebucket-1250000000"; //存储桶名称，格式：BucketName-APPID
CreateBucketRequest createBucketRequest = new CreateBucketRequest(bucket);
// 设置 bucket 的权限为 Private(私有读写)、其他可选有 PublicRead（公有读私有写）、PublicReadWrite（公有读写）
createBucketRequest.setCannedAcl(CannedAccessControlList.Private);
try{
Bucket bucketResult = cosClient.createBucket(createBucketRequest);
} catch (CosServiceException serverException) {
serverException.printStackTrace();
} catch (CosClientException clientException) {
clientException.printStackTrace();
}
```

查询存储桶列表

查询用户的存储桶列表，参考示例如下：

```
List<Bucket> buckets = cosClient.listBuckets();
for (Bucket bucketElement : buckets) {
String bucketName = bucketElement.getName();
String bucketLocation = bucketElement.getLocation();
}
```

上传对象

将本地文件或者已知长度的输入流内容上传到 COS，适用于20M以下图片类小文件上传，最大支持上传不超过5GB文件。5GB以上的文件必须使用分块上传或高级 API 接口上传。

> 说明：
> 高级 API 接口在 `com.qcloud.cos.transfer.\*` 子包下。

- 若本地文件大部分在20M以上，建议您参考使用高级 API 接口进行上传。
- 若 COS 上已存在同样 Key 的对象，上传时则会覆盖旧的对象。
- 若要创建目录对象，请参见 SDK 如何创建目录。
- 对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/images/picture.jpg` 中，对象键为 images/picture.jpg，详情请参见 对象键 的说明。

版权所有：腾讯云计算（北京）有限责任公司

上传不超过5GB的文件，参考示例如下：

```
// 指定要上传的文件
File localFile = new File(localFilePath);
// 指定文件将要存放的存储桶
String bucketName = "examplebucket-1250000000";
// 指定文件上传到 COS 上的路径，即对象键。例如对象键为 folder/picture.jpg，则表示将文件 pic
ture.jpg 上传到 folder 路径下
String key = "exampleobject";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFi
le);
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
```

查询对象列表

查询存储桶中对象列表，参考示例如下：

```
// Bucket 的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
// 设置 bucket 名称
listObjectsRequest.setBucketName(bucketName);
// prefix 表示列出的 object 的 key 以 prefix 开始
listObjectsRequest.setPrefix("images/");
// deliter 表示分隔符，设置为/表示列出当前目录下的 object，设置为空表示列出所有的 object
listObjectsRequest.setDelimiter("/");
// 设置最大遍历出多少个对象，一次 listobject 最大支持1000
listObjectsRequest.setMaxKeys(1000);
ObjectListing objectListing = null;
do {
try {
objectListing = cosClient.listObjects(listObjectsRequest);
} catch (CosServiceException e) {
e.printStackTrace();
return;
} catch (CosClientException e) {
e.printStackTrace();
return;
}
// common prefix 表示表示被 delimiter 截断的路径，如 delimter 设置为/，common prefix
则表示所有子目录的路径
List<String> commonPrefixs = objectListing.getCommonPrefixes();

// object summary 表示所有列出的 object 列表
List<COSObjectSummary> cosObjectSummaries = objectListing.getObjectSummaries();
for (COSObjectSummary cosObjectSummary : cosObjectSummaries) {
```

```
// 文件的路径 key
String key = cosObjectSummary.getKey();
// 文件的 etag
String etag = cosObjectSummary.getETag();
// 文件的长度
long fileSize = cosObjectSummary.getSize();
// 文件的存储类型
String storageClasses = cosObjectSummary.getStorageClass();
}

String nextMarker = objectListing.getNextMarker();
listObjectsRequest.setMarker(nextMarker);
} while (objectListing.isTruncated());
```

下载对象

上传对象后，您可以用同样的 key，调用 GetObject 接口将对象下载到本地，也可以生成预签名链接（下载请指定 method 为 GET，详情请参见 预签名 URL），分享到其他终端来进行下载。但如果您的文件设置了私有读权限，那么请注意预签名链接的有效期。
将文件下载到本地指定路径，参考示例如下：

```
// Bucket 的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 指定文件在 COS 上的路径，即对象键。例如对象键为 folder/picture.jpg，则表示下载的文件 pic
ture.jpg 在 folder 路径下
String key = "exampleobject";
// 方法1 获取下载输入流
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
COSObject cosObject = cosClient.getObject(getObjectRequest);
COSObjectInputStream cosObjectInput = cosObject.getObjectContent();
// 下载对象的 CRC64
String crc64Ecma = cosObject.getObjectMetadata().getCrc64Ecma();
// 关闭输入流
cosObjectInput.close();

// 方法2 下载文件到本地的路径，例如 D 盘的某个目录
String outputFilePath = "exampleobject";
File downFile = new File(outputFilePath);
getObjectRequest = new GetObjectRequest(bucketName, key);
ObjectMetadata downObjectMeta = cosClient.getObject(getObjectRequest, downFile);
```

删除对象

删除 COS 上指定路径的对象，代码如下：

```
// Bucket 的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 指定被删除的文件在 COS 上的路径，即对象键。例如对象键为 folder/picture.jpg，则表示删除位于 folder 路径下的文件 picture.jpg
String key = "exampleobject";
cosClient.deleteObject(bucketName, key);
```

默认请求重试策略

使用 SDK 生成的 cosClient 发起的请求，默认已经对响应错误的请求进行了重试。重试规则如下：

- 重试的次数：默认值为 3，可以通过 clientConfig.setMaxErrorRetry 进行设置。
  如果设置为 0，则所有类型的错误请求都不进行重试。
- 重试的错误类型：客户端异常中所有的报 IOException 的错误。服务端异常中状态码为 500, 502, 503 504 的错误。

用户也可以根据自己的需求，设置重试策略。代码如下：

设置重试次数：

```
Region region = new Region("COS_REGION");
ClientConfig clientConfig = new ClientConfig(region);
// 设置最大重试次数为 4 次
clientConfig.setMaxErrorRetry = 4;
```

设置重试策略：

```
// 自定义重试策略
public class OnlyIOExceptionRetryPolicy extends RetryPolicy {
@Override
public <X extends CosServiceRequest> boolean shouldRetry(CosHttpRequest<X> reques
t,
HttpResponse response,
Exception exception,
int retryIndex) {
// 如果是客户端的 IOException 异常则重试，否则不重试
if (exception.getCause() instanceof IOException) {
return true;
}
return false;
}
}

Region region = new Region("COS_REGION");
```

```
ClientConfig clientConfig = new ClientConfig(region);
RetryPolicy myRetryPolicy = new OnlyIOExceptionRetryPolicy();
// 设置自定义的重试策略
clientConfig.setRetryPolicy(myRetryPolicy);
```

关闭客户端

关闭 cosClient，并释放 HTTP 连接的后台管理线程，代码如下：

```
// 关闭客户端(关闭后台线程)
cosClient.shutdown();
```

常见问题

您在使用过程中可能会碰到的一些常见问题，相关的解决办法可参见 Java SDK 常见问题。

Java SDK 常见问题

最近更新时间：2023-01-19 14:33:14

客户端网络正常，但是通过 HTTP 访问 COS 非常慢，或者报错 Connection reset，该如何处理？

部分区域的运营商可能会对 COS 的域名进行劫持，因此尽量通过 HTTPS 来访问 COS。

引入 SDK 运行后，出现 java.lang.NoSuchMethodError 的异常，该如何处理？

原因：一般是发生了 JAR 包冲突，例如，用户的工程中的 httpclient 库中 的 JAR 包版本没有 A 方法，但是 SDK 依赖的 JAR 包使用了 A 方法。此时，由于运行时加载顺序的问题，加载了用户工程中的 httpclient 库，运行时便会抛出 NoSuchMethodError 的异常。
解决方法：
方式一：将工程中引起 NoSuchMethodError 包的版本，改成和 SDK 中 pom.xml 里的对应库的版本一致
方式二：将 cos-java-sdk 换成 cos_api-bundle。此方案会将 cos-java-sdk 的所有依赖都独立安装，所以会占用更多的空间。

```
<groupId>com.qcloud</groupId>
<artifactId>cos_api-bundle</artifactId>
<version>5.6.35</version>
```

Java SDK 的默认超时时间是多少？

Java SDK 默认连接超时时间为 30000ms，默认读写超时时间为 30000ms，可以使用 SDK 中的 SetConnectionTimeoutMs 方法和 setSocketTimeout 来进行调整。

Java SDK 上传速度慢，日志频繁打印 IOException，该如何处理？

原因与解决办法如下：

1. 首先确认下是否是通过公网访问 COS，目前同地域 CVM 访问 COS 走内网（内网域名解析出的 IP 是10、100、169网段，有关 COS 域名请参见 地域和访问域名），如果是通过公网确认出口带宽是否较小，或者是否有其他程序占用带宽资源。
2. 确保在生产环境中的日志级别不是 DEBUG，推荐使用 INFO 日志。
3. 目前简单上传速度可达10MB，高级 API 在32并发的情况下速度可达60MB，如果速度远低于这两个值，请参考以上两点。
4. 如果 WARN 日志打印 IOException 可以忽略，SDK 会进行重试。IOException 的原因可能是网速过慢，解决办法可参考1和2。

上传文件请求路径中带有“+”，报错 Code: 403 SignatureDoesNotMatch，如何处理？

版权所有：腾讯云计算（北京）有限责任公司

可能原因：用户 Java 环境的 httpclient 版本引起的 urlencode 编码错误。
解决方法：建议您通过以下方式解决：
方式一：使用 httpclient 4.5.3 版本

```
<groupId>org.apache.httpcomponents</groupId>
<artifactId>httpclient</artifactId>
<version>4.5.3</version>
```

方式二：将 cos-java-sdk 换成 cos_api-bundle。此方案会将 cos-java-sdk 的所有依赖都独立安装，所以会占用更多的空间。

```
<groupId>com.qcloud</groupId>
<artifactId>cos_api-bundle</artifactId>
<version>5.6.35</version>
```

使用 Java SDK 报错提示某个依赖版本太低，如何处理？

原因：用户 Java 环境的依赖版本和 Java SDK 需要的依赖版本有冲突。
解决方法:
方式一：按照提示，升级对应依赖到要求的版本。
方式二：将 cos-java-sdk 换成 cos_api-bundle。此方案会将 cos-java-sdk 的所有依赖都独立安装，所以会占用更多的空间。

```
<groupId>com.qcloud</groupId>
<artifactId>cos_api-bundle</artifactId>
<version>5.6.35</version>
```

SDK 如何创建目录？

对象存储中文件和目录都是对象，目录只是以 `/` 结尾的对象。创建文件时，不需要创建目录。如创建一个对象键为 `xxx/yyy/zzz.txt` 的文件，只用把 key 设置为 `xxx/yyy/zzz.txt` 即可，不用建立 `xxx/yyy/` 这个对象。在控制台上展示时，也会以 `/` 作为分隔，展示出目录的层级效果。但这些目录对象是不存在的。如果想创建一个目录对象，可使用以下的示例代码：

```
String bucketName = "examplebucket-1250000000";
String key = "folder/images/";
// 目录对象即是一个/结尾的空文件，上传一个长度为 0 的 byte 流
InputStream input = new ByteArrayInputStream(new byte[0]);
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setContentLength(0);
PutObjectRequest putObjectRequest =
```

版权所有：腾讯云计算（北京）有限责任公司

```
new PutObjectRequest(bucketName, key, input, objectMetadata);
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
```

SDK 如何使用 HTTPS?

SDK 中相关的配置都统一放在 ClientConfig 类中，示例代码如下：

```
// 初始化用户身份信息(secretId, secretKey)
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 设置 bucket 的区域, COS 地域的简称请参照 https://intl.cloud.tencent.com/document/p
roduct/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 配置使用 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 生成 cos 客户端
COSClient cosClient = new COSClient(cred, clientConfig);
```

SDK 如何使用代理？

对于需要使用代理访问 COS 的客户，可在 ClientConfig 类中，配置使用代理 IP （或域名）以及端口，示例代码如下：

```
// 初始化用户身份信息(secretId, secretKey)
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 设置 bucket 的区域, COS 地域的简称请参照 https://intl.cloud.tencent.com/document/p
roduct/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 配置使用代理(IP 和端口需要同时设置)
// 设置代理 IP (也可传入域名)
clientConfig.setHttpProxyIp("192.168.2.3");
// 设置代理端口
clientConfig.setHttpProxyPort(8080);
// 生成 cos 客户端
COSClient cosClient = new COSClient(cred, clientConfig);
```

如何设置自定义 EndpointBuilder？

您的场景也许需要指定 API 请求的 Endpoint，此时，您需要实现 EndpointBuilder 接口中的 buildGeneralApiEndpoint 和 buildGetServiceApiEndpoint 中的两个函数，分别为普通 API 请求和 GETService 请求指定远端的 Endpoint。使用示例如下：

版权所有：腾讯云计算（北京）有限责任公司

```
// 步骤1：实现 EndpointBuilder 接口中的两个函数
class SelfDefinedEndpointBuilder implements EndpointBuilder {
@Override
public String buildGeneralApiEndpoint(String bucketName) {
return String.format("%s.%s", bucketName, "mytest.com");
}
@Override
public String buildGetServiceApiEndpoint() {
return "service.mytest.com";
}
}
// 步骤2：初始化客户端
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
SelfDefinedEndpointBuilder selfDefinedEndpointBuilder = new SelfDefinedEndpointBu
ilder();
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing"));
clientConfig .setEndpointBuilder(selfDefinedEndpointBuilder);
COSClient cosClient = new COSClient(cred, clientConfig);
```

SDK 的上传、下载、批量删除等操作中，使用的 key 值是否需要添加 / 前缀？

对象存储的 key 值无需携带 / 前缀。例如，您将对象 key 值设置为 exampleobject 上传的对象，可以通过 URL：`http://cos.ap-guangzhou.myqcloud.com/exampleobject` 进行访问。

> 注意：
> 在批删请求中，请勿传入 / 前缀的 key，这将导致对象删除失败。

COS 使用 Java SDK 上传报错 please make sure bucket name must contain legal appid when appid is missing. example: bucektname-1250000000，该如何处理？

COS Java SDK 旧版本仅支持例如125开头的 APPID，如果您使用的是 130 开头的 APPID，建议您将 SDK 更新至最新版本。Java SDK 下载安装请参见 Java SDK 文档。

使用 COS Java SDK 上传大文件报错，该如何处理？

使用 Java SDK 上传大于5G的文件时，建议使用 高级上传接口。高级上传接口根据用户文件的长度和数据类型，自动选择以简单上传或分块上传方式。

Java SDK 如何获取文件上传进度？

需获取上传进度，建议使用 高级上传 接口，通过调用 Upload 的方法 getProgress() 获取上传的进度 TransferProgress 类。

Java SDK 是否支持计算本地文件的 CRC64？

COS Java SDK 本身不支持计算本地文件的 crc64，需要您在业务侧自行实现，您可以参考 Java issues 寻求答案。

Java SDK 校验 COS 返回的 CRC64 是 21 位的怎么和本地计算的 CRC64 对比？

由于 Java 的 Long 只能支持 20 位数字，可以参考以下的对比方案：

```
CRC64 localCRC = new CRC64();
// 计算本地的 crc64
localCRC.update();
//...
// 把 COS 返回的 CRC 变成 Long
cosCRC = crc64ToLong(strCOSCRC);
// 比较
if (cosCRC == localCRC.getValue()) {
xxx
}
// 用来将 COS 返回的 CRC64 转化成 1 个 Java 里的 Long
long crc64ToLong(String crc64) {
if (crc64.charAt(0) == '-') {
return negativeCrc64ToLong(crc64);
} else {
return positiveCrc64ToLong(crc64);
}
}
long positiveCrc64ToLong(String strCrc64) {
BigInteger crc64 = new BigInteger(strCrc64);
BigInteger maxLong = new BigInteger(Long.toString(Long.MAX_VALUE));
int maxCnt = 0;
while (crc64.compareTo(maxLong) > 0) {
crc64 = crc64.subtract(maxLong);
maxCnt++;
}
return crc64.longValue() + Long.MAX_VALUE * maxCnt;
}
long negativeCrc64ToLong(String strCrc64) {
BigInteger crc64 = new BigInteger(strCrc64);
BigInteger minLong = new BigInteger(Long.toString(Long.MIN_VALUE));
int minCnt = 0;
while (crc64.compareTo(minLong) < 0) {
crc64 = crc64.subtract(minLong);
minCnt++;
}
```

版权所有：腾讯云计算（北京）有限责任公司

```
return crc64.longValue() + Long.MIN_VALUE * minCnt;
}
```

Java SDK 能否使用长连接?

Java SDK 默认使用长连接。

如何使用 Java SDK 获取指定目录所有路径的同时，判断路径是文件还是文件夹？

可以使用 Java SDK 中的 查询对象列表 功能来列出存储桶内的对象，**使用 prefix 参数可以指定目录前缀**。

Java SDK 报错连接池已关闭，抛出异常： java.lang.IllegalStateException: Connection pool shut down，该如何处理？

1. COSClient 是线程安全的类，允许多线程访问同一实例。
   因为实例内部维持了一个连接池，创建多个实例可能导致程序资源耗尽。请确保程序生命周期内实例只有一个，且在不再需要使用时，调用 COSClient.shutdown() 方法将其关闭。
   如果需要新建实例，请先将之前的实例关闭。**推荐一个进程里只使用一个 COSClient，在程序全部结束退出时才调用 COSClient.shutdown()**。
2. 高级接口封装类的 TransferManager 中也使用了 COSClient，建议在程序全部结束退出时再调用 TransferManager.shutdownNow(false)。
   此外，如果在创建 TransferManager 实例时复用了其他 COSClient，请在关闭时加入参数 false（即 TransferManager.shutdownNow(false)），避免关闭复用的 COSClient，导致其他使用 COSClient 的地方报错。

Java SDK 报错请求过期，抛出异常：com.qcloud.cos.exception.CosServiceException: Request has expired (Status Code: 403; Error Code: AccessDenied)，该如何处理？

由于签名过期导致，重新生成签名即可解决；若重新生成签名仍报相同的错误，可以再检查机器的本地时间是否与所在时区的时间一致。

版权所有：腾讯云计算（北京）有限责任公司

存储桶操作

最近更新时间：2023-06-16 15:59:14

简介

本文档提供关于存储桶的基本操作 API 概览以及 SDK 示例代码。

注意：

- 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄露目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

API	操作名	操作描述
GET Service (List Buckets)	查询存储桶列表	查询指定账号下所有的存储桶列表
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

查询存储桶列表

功能说明

查询指定账号下所有的存储桶列表。

方法原型

```
public List<Bucket> listBuckets() throws CosClientException, CosServiceException;
```

请求示例

```
// 如果只调用 listBuckets 方法，则创建 cosClient 时指定 region 为 new Region("") 即可
List<Bucket> buckets = cosClient.listBuckets();
for (Bucket bucketElement : buckets) {
```

```
String bucketName = bucketElement.getName();
String bucketLocation = bucketElement.getLocation();
}
```

参数说明

无

返回结果说明

- 成功：返回一个 所有 Bucket 类的列表，Bucket 类包含了 bucket 成员，location 等信息。
- 失败：发生错误（如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

创建存储桶

功能说明

在指定账号下创建一个存储桶。同一用户账号下，可以创建多个存储桶，数量上限是200个（不区分地域），存储桶中的对象数量没有限制。创建存储桶是低频操作，一般建议在控制台创建 Bucket，在 SDK 进行 Object 的操作。

方法原型

```
public Bucket createBucket(String bucketName) throws CosClientException, CosServi
ceException;
```

请求示例

```
String bucket = "examplebucket-1250000000"; //存储桶名称，格式：BucketName-APPID
CreateBucketRequest createBucketRequest = new CreateBucketRequest(bucket);
// 设置 bucket 的权限为 Private(私有读写)，其他可选有公有读私有写，公有读写
createBucketRequest.setCannedAcl(CannedAccessControlList.Private);
Bucket bucketResult = cosClient.createBucket(createBucketRequest);
```

参数说明

参数名称	描述	类型
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String

返回结果说明

- 成功： Bucket 类，包含有关 Bucket 的描述（Bucket 的名称，owner 和创建日期）。

- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

检索存储桶及其权限

功能说明

检索存储桶是否存在且是否有权限访问。

方法原型

```
public boolean doesBucketExist(String bucketName)
throws CosClientException, CosServiceException;
```

请求示例

```
// bucket的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
boolean bucketExistFlag = cosClient.doesBucketExist(bucketName);
```

参数说明

参数名称	描述	类型
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String

返回结果说明

- 成功：存在返回 true，否则 false。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

删除存储桶

功能说明

删除指定账号下的空存储桶。

方法原型

```
public void deleteBucket(String bucketName) throws CosClientException, CosService
Exception;
```

请求示例

```
// bucket的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
cosClient.deleteBucket(bucketName);
```

参数说明

参数名称	描述	类型
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

对象操作
上传对象

最近更新时间：2022-10-28 16:29:12

简介

本文档提供关于对象的上传操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object	简单上传对象	上传一个对象至存储桶
POST Object	表单上传对象	使用表单请求上传对象
APPEND Object	追加上传对象	使用分块追加的方式上传对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传操作
Upload Part	上传分块	分块上传对象
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

高级接口（推荐）

高级 API 由类 TransferManager 通过封装各个简单接口，来提供更便捷操作的接口。内部使用一个线程池，来并发接受和处理用户的请求，因此用户也可选择提交多个任务之后异步的执行任务。

版权所有：腾讯云计算（北京）有限责任公司

TransferManager 实例是并发安全的，这里推荐一个进程只创建一个 TransferManager 实例，当不会再通过这个实例调用高级接口的时候，再选择关闭这个实例。

功能说明

高级接口封装了简单上传、分块上传接口，根据文件大小智能的选择上传方式，同时支持续传、上传时显示进度等功能。

说明：

- 分块阈值用来区分使用简单上传还是分块上传，阈值支持用户自行配置，默认为5MB。
- 分块大小支持用户自行配置，默认为1MB。
- 对流类型的上传，小于分块阈值或未指定 Content-Length 头部的上传，高级接口会选择简单上传。
- 对流类型的上传，大于分块阈值且指定 Content-Length 头部的上传，高级接口会选择分块上传。
- 对文件类型的上传，小于分块阈值的文件，高级接口会选择简单上传，大于分块阈值的文件，高级接口会选择分块上传。
- 对于文件类型的分块上传，高级接口会多线程并发同时上传多个分块。
- 对于分块上传，高级上传接口提供获取进度的功能，具体参见下面的示例。

创建 TransferManager

使用高级接口的操作之前，必须先创建一个 TransferManager 的实例。

```
// 创建 TransferManager 实例，这个实例用来后续调用高级接口
TransferManager createTransferManager() {
// 创建一个 COSClient 实例，这是访问 COS 服务的基础实例。
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 自定义线程池大小，建议在客户端与 COS 网络充足（例如使用腾讯云的 CVM，同地域上传 COS）的情况下，设置成16或32即可，可较充分的利用网络资源
// 对于使用公网传输且网络带宽质量不高的情况，建议减小该值，避免因网速过慢，造成请求超时。
ExecutorService threadPool = Executors.newFixedThreadPool(32);
// 传入一个 threadpool，若不传入线程池，默认 TransferManager 中会生成一个单线程的线程池。
TransferManager transferManager = new TransferManager(cosClient, threadPool);
// 设置高级接口的配置项
// 分块上传阈值和分块大小分别为 5MB 和 1MB
TransferManagerConfiguration transferManagerConfiguration = new TransferManagerConfiguration();
transferManagerConfiguration.setMultipartUploadThreshold(5*1024*1024);
transferManagerConfiguration.setMinimumUploadPartSize(1*1024*1024);
transferManager.setConfiguration(transferManagerConfiguration);
return transferManager;
}
```

参数说明

TransferManagerConfiguration 类用于记录高级接口的配置信息，其主要成员说明如下：

成员名	设置方法	描述	类型
minimumUploadPartSize	set 方法	分块上传的块大小，单位：字节（Byte），默认为5MB	long
multipartUploadThreshold	set 方法	大于等于该值则并发的分块上传文件，单位：字节（Byte），默认为5MB	long
multipartCopyThreshold	set 方法	大于等于该值则并发的分块复制文件，单位：字节（Byte），默认为5GB	long
multipartCopyPartSize	set 方法	分块复制的块大小，单位：字节（Byte），默认为100MB	long

关闭 TransferManager

在确定不再通过 TransferManager 的实例调用高级接口之后，一定要关闭这个实例，防止资源泄露。

```
void shutdownTransferManager(TransferManager transferManager) {
// 指定参数为 true，则同时会关闭 transferManager 内部的 COSClient 实例。
// 指定参数为 false，则不会关闭 transferManager 内部的 COSClient 实例。
transferManager.shutdownNow(true);
}
```

上传本地文件

上传的源是本地的一个文件。

方法原型

```
// 上传对象
public Upload upload(final PutObjectRequest putObjectRequest)
throws CosServiceException, CosClientException;
```

请求示例

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 创建 TransferManager
TransferManager transferManager = createTransferManager();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。
```

```
String key = "exampleobject";
// 本地文件路径
String localFilePath = "/path/to/localFile";
File localFile = new File(localFilePath);
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFi
le);
// 设置存储类型（如有需要，不需要请忽略此行代码），默认是标准(Standard)，低频(standard_ia)
// 更多存储类型请参见 https://cloud.tencent.com/document/product/436/33417
putObjectRequest.setStorageClass(StorageClass.Standard_IA);
//若需要设置对象的自定义 Headers 可参照下列代码,若不需要可省略下面这几行,对象自定义 Headers
的详细信息可参考https://cloud.tencent.com/document/product/436/13361
ObjectMetadata objectMetadata = new ObjectMetadata();
//若设置Content-Type、Cache-Control、Content-Disposition、Content-Encoding、Expires
这五个字自定义 Headers，推荐采用objectMetadata.setHeader()
objectMetadata.setHeader(key, value);
//若要设置“x-cos-meta-[自定义后缀]”这样的自定义 Header，推荐采用
Map<String, String> userMeta = new HashMap<String, String>();
userMeta.put("x-cos-meta-[自定义后缀]", "value");
objectMetadata.setUserMetadata(userMeta);
putObjectRequest.withMetadata(objectMetadata);
try {
// 高级接口会返回一个异步结果Upload
// 可同步地调用 waitForUploadResult 方法等待上传完成，成功返回UploadResult，失败抛出异常
Upload upload = transferManager.upload(putObjectRequest);
UploadResult uploadResult = upload.waitForUploadResult();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

参数说明

参数名称	描述	类型
putObjectRequest	上传文件请求	PutObjectRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
file	构造函数或 set 方法	本地文件	File
input	构造函数或 set 方法	输入流	InputStream
metadata	构造函数或 set 方法	文件的元数据	ObjectMetadata
trafficLimit	set 方法	用于对上传对象进行流量控制，单位：bit/s，默认不进行流量控制	Int
storageClass	set 方法	存储类型	String

返回值

- 成功：返回 Upload，可以查询上传是否结束，也可同步的等待上传结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

返回参数说明

通过调用 Upload 的 waitForUploadResult() 方法获取的上传对象信息记录在类 UploadResult 中，类 UploadResult 主要成员说明如下：

成员名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String

版权所有：腾讯云计算（北京）有限责任公司

成员名称	描述	类型
key	对象键（Key）是对象在存储桶中的唯一标识。 例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
requestId	请求 Id	String
dateStr	当前服务端时间	String
versionId	当存储桶开启了版本控制功能，返回对象的版本号 Id	String
crc64Ecma	服务端根据对象内容计算出来的 CRC64	String

上传流类型

上传的源是一个 InputStream 类型（和其子类型）的流实例。

方法原型

```
// 上传对象
public Upload upload(final PutObjectRequest putObjectRequest)
throws CosServiceException, CosClientException;
```

请求示例

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 创建 TransferManager
TransferManager transferManager = createTransferManager();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。
String key = "exampleobject";
// 这里创建一个 ByteArrayInputStream 来作为示例，实际中这里应该是您要上传的 InputStream 类型的流
long inputStreamLength = 1024 * 1024;
byte data[] = new byte[inputStreamLength];
InputStream inputStream = new ByteArrayInputStream(data);
ObjectMetadata objectMetadata = new ObjectMetadata();
// 上传的流如果能够获取准确的流长度，则推荐一定填写 content-length
// 如果确实没办法获取到，则下面这行可以省略，但同时高级接口也没办法使用分块上传了
objectMetadata.setContentLength(inputStreamLength);
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, inputStream, objectMetadata);
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 设置存储类型（如有需要，不需要请忽略此行代码），默认是标准(Standard)，低频(standard_ia)
// 更多存储类型请参见 https://cloud.tencent.com/document/product/436/33417
putObjectRequest.setStorageClass(StorageClass.Standard_IA);
try {
// 高级接口会返回一个异步结果Upload
// 可同步地调用 waitForUploadResult 方法等待上传完成，成功返回UploadResult，失败抛出异常
Upload upload = transferManager.upload(putObjectRequest);
UploadResult uploadResult = upload.waitForUploadResult();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

参数说明

参数名称	描述	类型
putObjectRequest	上传文件请求	PutObjectRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
file	构造函数或 set 方法	本地文件	File

Request 成员	设置方法	描述	类型
input	构造函数或 set 方法	输入流	InputStream
metadata	构造函数或 set 方法	文件的元数据	ObjectMetadata
trafficLimit	set 方法	用于对上传对象进行流量控制，单位：bit/s，默认不进行流量控制	Int
storageClass	set 方法	存储类型	String

返回值

- 成功：返回 Upload，可以查询上传是否结束，也可同步的等待上传结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

返回参数说明

通过调用 Upload 的 waitForUploadResult() 方法获取的上传对象信息记录在类 UploadResult 中，类 UploadResult 主要成员说明如下：

成员名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	对象键（Key）是对象在存储桶中的唯一标识。 例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
requestId	请求 Id	String
dateStr	当前服务端时间	String
versionId	当存储桶开启了版本控制功能，返回对象的版本号 Id	String
crc64Ecma	服务端根据对象内容计算出来的 CRC64	String

显示上传进度

说明：
只有使用了分块上传的接口才能显示上传进度。

要显示上传进度首先需要一个自己的打印上传进度的函数，在这个函数里通过调用接口来获取已经成功上传的大小从而计算出当前进度。

方法原型

```
// 上传对象
public Upload upload(final PutObjectRequest putObjectRequest)
throws CosServiceException, CosClientException;
```

请求示例

```
// 可以参考下面的例子，结合实际情况做调整
void showTransferProgress(Transfer transfer) {
// 这里的 Transfer 是异步上传结果 Upload 的父类
System.out.println(transfer.getDescription());
// transfer.isDone() 查询上传是否已经完成
while (transfer.isDone() == false) {
try {
// 每 2 秒获取一次进度
Thread.sleep(2000);
} catch (InterruptedException e) {
return;
}
TransferProgress progress = transfer.getProgress();
long sofar = progress.getBytesTransferred();
long total = progress.getTotalBytesToTransfer();
double pct = progress.getPercentTransferred();
System.out.printf("upload progress: [%d / %d] = %.02f%%\n", sofar, total, pct);
}
// 完成了 Completed，或者失败了 Failed
System.out.println(transfer.getState());
}
```

结合上传文件的代码示例如下：

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 创建 TransferManager
TransferManager transferManager = createTransferManager();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
```

```
// 对象键(Key)是对象在存储桶中的唯一标识。
String key = "exampleobject";
// 本地文件路径
String localFilePath = "/path/to/localFile";
File localFile = new File(localFilePath);
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFi
le);
try {
// 高级接口会返回一个异步结果Upload
Upload upload = transferManager.upload(putObjectRequest);
// 打印上传进度，直到上传结束
showTransferProgress(upload);
// 捕获可能出现的异常
UploadResult uploadResult = upload.waitForUploadResult();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

获取进度说明

通过 upload 这个类的 getProgress 可以获得一个 TransferProgress 类，这个类的下面三个方法用来获取上传进度，说明如下：

方法名称	描述	类型
getBytesTransferred	获取已上传的字节数	long
getTotalBytesToTransfer	获取总文件的字节数	long
getPercentTransferred	获取已上传的字节百分比	double

参数说明

参数名称	描述	类型
putObjectRequest	上传文件请求	PutObjectRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
file	构造函数或 set 方法	本地文件	File
input	构造函数或 set 方法	输入流	InputStream
metadata	构造函数或 set 方法	文件的元数据	ObjectMetadata
trafficLimit	set 方法	用于对上传对象进行流量控制，单位：bit/s，默认不进行流量控制	Int

返回值

- 成功：返回 Upload，可以查询上传是否结束，也可同步的等待上传结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

返回参数说明

通过调用 Upload 的 waitForUploadResult() 方法获取的上传对象信息记录在类 UploadResult 中，类 UploadResult 主要成员说明如下：

成员名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String

成员名称	描述	类型
key	对象键（Key）是对象在存储桶中的唯一标识。 例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
requestId	请求 ID	String
dateStr	当前服务端时间	String
versionId	当存储桶开启了版本控制功能，返回对象的版本号 ID	String
crc64Ecma	服务端根据对象内容计算出来的 CRC64	String

上传时暂停、继续与取消

使用高级接口可以暂停上传任务、随后继续上传，或者直接取消一个上传任务。

> 说明：
>
> - 流式上传不可以暂停、继续与取消。
> - 简单上传不可以暂停、继续与取消。
> - 加密上传不可以暂停、继续与取消。

方法原型

```
// 上传对象
public Upload upload(final PutObjectRequest putObjectRequest)
throws CosServiceException, CosClientException;
```

请求示例

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 示例代码：创建 TransferManager
TransferManager transferManager = createTransferManager();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。
String key = "exampleobject";
// 本地文件路径
String localFilePath = "/path/to/localFile";
File localFile = new File(localFilePath);
```

版权所有：腾讯云计算（北京）有限责任公司

```
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFi
le);
try {
// 高级接口会返回一个异步结果Upload
Upload upload = transferManager.upload(putObjectRequest);
// 等待 3 秒，上传一部分
Thread.sleep(3000);
// 暂停上传，获取一个 PersistableUpload 实例，用来随后的恢复
PersistableUpload persistableUpload = upload.pause();
// 复杂的暂停与继续用法：
// PersistableUpload 实例也可以通过 serialize 序列化后存储起来，之后再通过 deserialize
反序列化来恢复继续上传
// persistableUpload.serialize(out);
// 继续上传
upload = transferManager.resumeUpload(persistableUpload);
// 捕获可能出现的异常
UploadResult uploadResult = upload.waitForUploadResult();
// 或者直接取消这次上传
// upload.abort();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 示例代码：关闭 TransferManager
shutdownTransferManager(transferManager);
```

参数说明

参数名称	描述	类型
putObjectRequest	上传文件请求	PutObjectRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String

Request 成员	设置方法	描述	类型
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。 例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
file	构造函数或 set 方法	本地文件	File
input	构造函数或 set 方法	输入流	InputStream
metadata	构造函数或 set 方法	文件的元数据	ObjectMetadata
trafficLimit	set 方法	用于对上传对象进行流量控制，单位：bit/s，默认不进行流量控制	Int

返回值

- 成功：返回 Upload，可以查询上传是否结束，也可同步的等待上传结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

返回参数说明

通过调用 Upload 的 waitForUploadResult() 方法获取的上传对象信息记录在类 UploadResult 中，类 UploadResult 主要成员说明如下：

成员名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	对象键（Key）是对象在存储桶中的唯一标识。 例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
requestId	请求 ID	String

版权所有：腾讯云计算（北京）有限责任公司

成员名称	描述	类型
dateStr	当前服务端时间	String
versionId	当存储桶开启了版本控制功能，返回对象的版本号 ID	String
crc64Ecma	服务端根据对象内容计算出来的 CRC64	String

上传本地目录

TransferManager 实例封装了从本地一个目录来读取文件并且上传到 COS 的功能，这个功能可以在不破坏目录结构的情况下，将文件上传。同时，也可以指定将目录下的文件上传到另一个目录。

> 说明：
> 支持递归上传目录，如果上传目录太大，可能会存在上传较慢或阻塞太久的情况。如果需要递归上传目录，则建议上传小容量的目录（例如有 1024 以上个文件）。如果需要上传大容量目录，建议分成多个小目录进行分次调用。

方法原型

```
public MultipleFileUpload uploadDirectory(String bucketName, String virtualDirectoryKeyPrefix,
File directory, boolean includeSubdirectories);
```

请求示例

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 示例代码：创建 TransferManager
TransferManager transferManager = createTransferManager();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 设置文件上传到 bucket 之后的前缀目录，设置为 “”，表示上传到 bucket 的根目录
String cos_path = "/prefix";
// 要上传的文件夹的绝对路径
String dir_path = "/path/to/localdir";
// 是否递归上传目录下的子目录，如果是 true，子目录下的文件也会上传，且cos上会保持目录结构
Boolean recursive = false;
try {
// 返回一个异步结果Upload，可同步的调用waitForUploadResult等待upload结束，成功返回UploadResult，失败抛出异常.
MultipleFileUpload upload = transferManager.uploadDirectory(bucketName, cos_path,
new File(dir_path), recursive);
```

```
// 可以选择查看上传进度，这个函数参见 高级接口 -> 上传文件 -> 显示上传进度
showTransferProgress(upload);
// 或者阻塞等待完成
upload.waitForCompletion();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 示例代码：关闭 TransferManager
shutdownTransferManager(transferManager);
```

参数说明

参数名称	描述	类型
bucketName	cos 上的 bucket	GetObjectRequest
virtualDirectoryKeyPrefix	cos 上 object 的前缀	String
directory	要上传的文件夹的绝对路径	File
includeSubDirectory	是否递归上传子目录	Boolean

返回值

- 成功：返回 MultipleFileUpload，可以查询上传是否结束，也可同步的等待上传结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

简单操作

简单操作由 COSClient 类型发起请求，使用简单操作之前必须先创建一个 COSClient 实例。

COSClient 实例是并发安全的，这里推荐一个进程只创建一个 COSClient 实例，当不会再通过这个实例发起请求的时候，再选择关闭这个实例。

创建 COSClient

调用 COS 的接口之前，必须先创建一个 COSClient 的实例。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 设置用户身份信息。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.co
m/cam/capi 进行查看和管理
String secretId = "SECRETID";
String secretKey = "SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

使用临时密钥创建 COSClient

如果要使用临时密钥请求 COS，则需要用临时密钥创建 COSClient。

本 SDK 并不能生成临时密钥，而需要使用额外的操作来生成，参考 临时密钥生成。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 这里需要已经获取到临时密钥的结果。
// 临时密钥的生成参考 https://intl.cloud.tencent.com/document/product/436/14048
String tmpSecretId = "TMPSECRETID";
String tmpSecretKey = "TMPSECRETKEY";
String sessionToken = "SESSIONTOKEN";
COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sess
ionToken);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
```

```
// 设置 bucket 的地域
// COS_REGION 请参照 https://cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议, http 或者 https
// 5.6.53 及更低的版本, 建议设置使用 https 协议
// 5.6.54 及更高版本, 默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置, 是可选的:
// 设置 socket 读取超时, 默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时, 默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话, 设置 http 代理, ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

上传本地文件

上传的源是本地的一个文件。

方法原型

```
// 方法1 将本地文件上传到 COS
public PutObjectResult putObject(String bucketName, String key, File file)
throws CosClientException, CosServiceException;
// 方法2 输入流上传到 COS
public PutObjectResult putObject(String bucketName, String key, InputStream inpu
t, ObjectMetadata metadata)
throws CosClientException, CosServiceException;
// 方法3 对以上两个方法的包装, 支持更细粒度的参数控制, 如 content-type, content-dispositi
on 等
public PutObjectResult putObject(PutObjectRequest putObjectRequest)
throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例, 如果没有则创建
// 详细代码参见本页:简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID, 此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。
String key = "exampleobject";
```

```
// 本地文件路径
String localFilePath = "/path/to/localFile";
File localFile = new File(localFilePath);
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFi
le);
// 设置存储类型（如有需要，不需要请忽略此行代码），默认是标准(Standard)，低频(standard_ia)
// 更多存储类型请参见 https://cloud.tencent.com/document/product/436/33417
putObjectRequest.setStorageClass(StorageClass.Standard_IA);
try {
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
System.out.println(putObjectResult.getRequestId());
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
putObjectRequest	上传文件请求	PutObjectRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
file	构造函数或 set 方法	本地文件	File

Request 成员	设置方法	描述	类型
input	构造函数或 set 方法	输入流	InputStream
metadata	构造函数或 set 方法	对象的元数据	ObjectMetad
trafficLimit	set 方法	用于对上传对象进行流量控制，单位：bit/s，默认不进行流量控制	Int
storageClass	set 方法	存储类型	String

ObjectMetadata 类用于记录对象的元信息，其主要成员说明如下：

成员名称	描述	类型
httpExpiresDate	缓存的超时时间，为 HTTP 响应头部中 Expires 字段的值	Date
ongoingRestore	正在从归档存储类型恢复该对象	Boolean
userMetadata	前缀为 x-cos-meta- 的用户自定义元信息	Map<String, String>
metadata	除用户自定义元信息以外的其他头部	Map<String, String>
restoreExpirationTime	归档对象恢复副本的过期时间	Date

返回结果说明

- 成功：PutObjectResult，包含文件的 eTag 等信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

返回参数说明

PutObjectResult 类用于返回结果信息，其主要成员说明如下：

成员名称	描述	类型
requestId	请求 ID	String
dateStr	当前服务端时间	String
versionId	开启版本控制的存储桶，返回对象的版本号 ID	String

版权所有：腾讯云计算（北京）有限责任公司

成员名称	描述	类型
eTag	简单上传接口返回对象的 MD5 值，例如：09cba091df696af91549de27b8e7d0f6，**注意：虽然在请求响应头域 ETag 的值是有双引号的，但是这里解析后的字符串 eTag 值是没有双引号的**	String
crc64Ecma	服务端根据对象内容计算出来的 CRC64	String

上传流类型

上传的源是一个 InputStream 类型（和其子类型）的流实例。

方法原型

```
// 方法1 将本地文件上传到 COS
public PutObjectResult putObject(String bucketName, String key, File file)
throws CosClientException, CosServiceException;
// 方法2 输入流上传到 COS
public PutObjectResult putObject(String bucketName, String key, InputStream inpu
t, ObjectMetadata metadata)
throws CosClientException, CosServiceException;
// 方法3 对以上两个方法的包装，支持更细粒度的参数控制，如 content-type, content-dispositi
on 等
public PutObjectResult putObject(PutObjectRequest putObjectRequest)
throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。
String key = "exampleobject";
// 这里创建一个 ByteArrayInputStream 来作为示例，实际中这里应该是您要上传的 InputStream
类型的流
long inputStreamLength = 1024 * 1024;
byte data[] = new byte[inputStreamLength];
InputStream inputStream = new ByteArrayInputStream(data);
ObjectMetadata objectMetadata = new ObjectMetadata();
// 上传的流如果能够获取准确的流长度，则推荐一定填写 content-length
// 如果确实没办法获取到，则下面这行可以省略，但同时高级接口也没办法使用分块上传了
objectMetadata.setContentLength(inputStreamLength);
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, inputSt
```

版权所有：腾讯云计算（北京）有限责任公司

```
ream, objectMetadata);
// 设置存储类型（如有需要，不需要请忽略此行代码），默认是标准(Standard)，低频(standard_ia)
// 更多存储类型请参见 https://cloud.tencent.com/document/product/436/33417
putObjectRequest.setStorageClass(StorageClass.Standard_IA);
try {
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
System.out.println(putObjectResult.getRequestId());
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
putObjectRequest	上传文件请求	PutObjectRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
file	构造函数或 set 方法	本地文件	File
input	构造函数或 set 方法	输入流	InputStream
metadata	构造函数或 set 方法	对象的元数据	ObjectMetad

版权所有：腾讯云计算（北京）有限责任公司

Request 成员	设置方法	描述	类型
trafficLimit	set 方法	用于对上传对象进行流量控制，单位：bit/s，默认不进行流量控制	Int
storageClass	set 方法	存储类型	String

ObjectMetadata 类用于记录对象的元信息，其主要成员说明如下：

成员名称	描述	类型
httpExpiresDate	缓存的超时时间，为 HTTP 响应头部中 Expires 字段的值	Date
ongoingRestore	正在从归档存储类型恢复该对象	Boolean
userMetadata	前缀为 x-cos-meta- 的用户自定义元信息	Map<String, String>
metadata	除用户自定义元信息以外的其他头部	Map<String, String>
restoreExpirationTime	归档对象恢复副本的过期时间	Date

返回结果说明

- 成功：PutObjectResult，包含文件的 eTag 等信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

返回参数说明

PutObjectResult 类用于返回结果信息，其主要成员说明如下：

成员名称	描述	类型
requestId	请求 ID	String
dateStr	当前服务端时间	String
versionId	开启版本控制的存储桶，返回对象的版本号 ID	String
eTag	简单上传接口返回对象的 MD5 值，例如：09cba091df696af91549de27b8e7d0f6，**注意：虽然在请求响应头域 ETag 的值是有双引号的，但是这里解析后的字符串 eTag 值是没有双引号的**	String
crc64Ecma	服务端根据对象内容计算出来的 CRC64	String

版权所有：腾讯云计算（北京）有限责任公司

创建目录

COS 本身没有目录的概念，但是可以将以 '/' 分隔的对象路径看做虚拟文件夹。

> 说明：
>
> - 如果需要一个目录，可以直接往需要的目录上传一个文件，则目录会自动创建。例如：上传一个 /dir/example.txt 这样的文件，则 /dir 这个目录就自动生成了。这种情况请参考本页的上传本地文件。

如果需要一个下面没有文件的目录，则使用这里的办法：上传一个空流到一个以 '/' 结尾的路径，这样就有了一个虚拟的目录。

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 在这里指定要创建的目录的路径
String key = "/example/dir/";
// 这里创建一个空的 ByteArrayInputStream 来作为示例
byte data[] = new byte[0];
InputStream inputStream = new ByteArrayInputStream(data);
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setContentLength(0);
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, inputSt
ream, objectMetadata);
try {
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
System.out.println(putObjectResult.getRequestId());
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

追加上传对象

以分段追加的方式上传对象。

方法原型

```
public AppendObjectResult appendObject(AppendObjectRequest appendObjectRequest)
throws CosServiceException, CosClientException
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。
String key = "exampleobject";
// 这里假定要追加上传的是这两个文件里的内容
File part1File = new File("/path/to/part1File");
File part2File = new File("/path/to/part2File");
try {
AppendObjectRequest appendObjectRequest = new AppendObjectRequest(bucketName, ke
y, part1File);
appendObjectRequest.setPosition(0L);
AppendObjectResult appendObjectResult = cosClient.appendObject(appendObjectReques
t);
long nextAppendPosition = appendObjectResult.getNextAppendPosition();
System.out.println(nextAppendPosition);
appendObjectRequest = new AppendObjectRequest(bucketName, key, part2File);
appendObjectRequest.setPosition(nextAppendPosition);
appendObjectResult = cosClient.appendObject(appendObjectRequest);
nextAppendPosition = appendObjectResult.getNextAppendPosition();
System.out.println(nextAppendPosition);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
// 关闭客户端
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
appendObjectRequest	追加上传文件请求	AppendObjectRequest

AppendObjectRequest 成员说明：

Request 成员	设置方法	描述	类型	
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String	
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String	
localfile	构造函数或 set 方法	本地文件	File	
input	构造函数或 set 方法	输入流	InputStream	
metadata	构造函数或 set 方法	对象的元数据	ObjectMetadata	
trafficLimit	set 方法	用于对上传对象进行流量控制，单位：bit/s，默认不进行流量控制	Int	
position	set 方法	追加操作的起始位置，单位：byte	Long	

返回结果说明

- 成功：AppendObjectResult。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

返回参数说明

AppendObjectResult 类用于返回结果信息，其主要成员说明如下：

成员名称	描述	类型
metadata	返回头部	ObjectMetadata
nextAppendPosition	下一次追加的位置	Long

版权所有：腾讯云计算（北京）有限责任公司

表单上传对象

使用表单请求上传对象。

构造表单上传的 form body

根据表单参数，构造上传最开始的 form body 部分

```
String buildPostObjectBody(String boundary, Map<String, String> formFields, Strin
g filename, String contentType) {
StringBuffer stringBuffer = new StringBuffer();
for(Map.Entry entry: formFields.entrySet()) {
// 添加boundary行,行首以--开头
stringBuffer.append("--").append(boundary).append("\r\n");
// 字段名
stringBuffer.append("Content-Disposition: form-data; name=\""
+ entry.getKey() + "\"\r\n\r\n");
// 字段值
stringBuffer.append(entry.getValue() + "\r\n");
}
// 添加boundary行,行首以--开头
stringBuffer.append("--").append(boundary).append("\r\n");
// 文件名
stringBuffer.append("Content-Disposition: form-data; name=\"file\"; "
+ "filename=\"" + filename + "\"\r\n");
// 文件类型
stringBuffer.append("Content-Type: " + contentType + "\r\n\r\n");
return stringBuffer.toString();
}
```

上传本地文件

```
// 存储桶的命名格式为 BucketName-APPID, 此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// COS_REGION 请参照 https://cloud.tencent.com/document/product/436/6224
String endpoint = "cos.{COS_REGION}.myqcloud.com";
// 对象键(Key)是对象在存储桶中的唯一标识。
String key = "exampleobject";
// SECRETID 和 SECRETKEY 请登录访问管理控制台进行查看和管理
String secretId = "AKIDXXXXXXXX";
String seretKey = "1A2Z3YYYYYYYYY";
String localFilePath = "/path/to/localFile";
String contentType = "image/jpeg";
long startTimestamp = System.currentTimeMillis() / 1000;
long endTimestamp = startTimestamp + 30 * 60;
String endTimestampStr = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'").for
```

版权所有：腾讯云计算（北京）有限责任公司

```
mat(endTimestamp * 1000);
String keyTime = startTimestamp + ";" + endTimestamp;
String boundary = "----WebKitFormBoundaryZBPbaoYE2gqeB21N";
// 设置表单的body字段值
Map<String, String> formFields = new HashMap<>();
formFields.put("q-sign-algorithm", "sha1");
formFields.put("key", key);
formFields.put("q-ak", secretId);
formFields.put("q-key-time", keyTime);
// 构造policy, 参考文档: https://intl.cloud.tencent.com/document/product/436/14690
String policy = "{\n" +
" \"expiration\": \"" + endTimestampStr + "\",\n" +
" \"conditions\": [\n" +
" { \"bucket\": \"" + bucketName + "\" },\n" +
" { \"q-sign-algorithm\": \"sha1\" },\n" +
" { \"q-ak\": \"" + secretId + "\" },\n" +
" { \"q-sign-time\":\"" + keyTime + "\" }\n" +
" ]\n" +
"}";
// policy需要base64后算放入表单中
String encodedPolicy = new String(Base64.encodeBase64(policy.getBytes()));
// 设置policy
formFields.put("policy", encodedPolicy);
// 根据编码后的policy和secretKey计算签名
COSSigner cosSigner = new COSSigner();
String signature = cosSigner.buildPostObjectSignature(seretKey,
keyTime, policy);
// 设置签名
formFields.put("q-signature", signature);
// 根据以上表单参数, 构造最开始的body部分
String formBody = buildPostObjectBody(boundary, formFields,
localFilePath, contentType);
HttpURLConnection conn = null;
try {
String urlStr = "http://" + bucketName + "." + endpoint;
URL url = new URL(urlStr);
conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("POST");
conn.setRequestProperty("User-Agent", VersionInfoUtils.getUserAgent());
conn.setRequestProperty("Content-Type", "multipart/form-data; boundary=" + bounda
ry);
conn.setDoOutput(true);
conn.setDoInput(true);
OutputStream out = new DataOutputStream(conn.getOutputStream());
// 写入表单的最开始部分
out.write(formBody.getBytes());
// 将文件内容写入到输出流中
```

```
File file = new File(localFilePath);
DataInputStream in = new DataInputStream(new FileInputStream(file));
int readBytes;
byte[] bytes = new byte[4096];
while ((readBytes = in.read(bytes)) != -1) {
out.write(bytes, 0, readBytes);
}
in.close();
// 添加最后一个分割符，行首和行尾都是--
byte[] endData = ("\r\n--" + boundary + "--\r\n").getBytes();
out.write(endData);
out.flush();
out.close();
// 读取响应头部
for (Map.Entry<String, List<String>> entries : conn.getHeaderFields().entrySet())
{
String values = "";
for (String value : entries.getValue()) {
values += value + ",";
}
if(entries.getKey() == null) {
System.out.println("reponse line:" + values );
} else {
System.out.println(entries.getKey() + ":" + values );
}
}
} catch (Exception e) {
e.printStackTrace();
throw e;
} finally {
if (conn != null) {
conn.disconnect();
}
}
```

分块操作

分块操作是大文件上传时，使用一系列分块，改善大文件上传耗时太久可能导致的各种中断问题。

> 说明：
> 如果要上传的文件或者流可以完整获取，推荐使用高级接口。

操作流程

分块上传的流程

1. 初始化分块上传（Initiate Multipart Upload），得到 UploadId
2. 使用 UploadId 上传分块（Upload Part）
3. 完成分块上传（Complete Multipart Upload）

分块继续上传的流程

1. 如果没有记录 UploadId，则要先查询分块上传任务（List Multipart Uploads），得到对应文件的 UploadId
2. 使用 UploadId 列出已上传的分块（List Parts）
3. 使用 UploadId 上传剩余的分块（Upload Part）
4. 完成分块上传（Complete Multipart Upload）

终止分块上传的流程

1. 如果没有记录 UploadId，则查询分块上传任务（List Multipart Uploads），得到对应文件的 UploadId
2. 终止分块上传并删除已上传分块（Abort Multipart Upload）

初始化分块上传

初始化分块上传操作，获取对应的 uploadId 用于后续的操作。

方法原型

```
public InitiateMultipartUploadResult initiateMultipartUpload(
InitiateMultipartUploadRequest request) throws CosClientException, CosServiceExce
ption;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。
String key = "exampleobject";
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(bucke
tName, key);
// 分块上传的过程中，仅能通过初始化分块指定文件上传之后的metadata
// 需要的头部可以在这里指定
ObjectMetadata objectMetadata = new ObjectMetadata();
request.setObjectMetadata(objectMetadata);
```

版权所有：腾讯云计算（北京）有限责任公司

```
try {
InitiateMultipartUploadResult initResult = cosClient.initiateMultipartUpload(requ
est);
// 获取uploadid
String uploadId = initResult.getUploadId();
System.out.println(uploadId);
} catch (CosServiceException e) {
throw e;
} catch (CosClientException e) {
throw e;
}
```

参数说明

参数名称	描述	类型
initiateMultipartUploadRequest	请求	InitiateMultipartUploadRequest

Request 成员说明：

参数名称	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	指定分块上传到 COS 上的路径，即 对象键。例如对象键为 folder/picture.jpg	String

返回结果说明

- 成功：返回 InitiateMultipartUploadResult，包含标志本次分块上传的 uploadId。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询分块上传任务

查询分块上传任务（List Multipart Uploads），获取当前正在进行的所有上传任务，从中找到需要处理的 uploadId。

方法原型

```
public MultipartUploadListing listMultipartUploads(
ListMultipartUploadsRequest listMultipartUploadsRequest)
throws CosClientException, CosServiceException
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 需要查询的分块上传任务对应的 key
String targetKey = "targetKey";
ListMultipartUploadsRequest listMultipartUploadsRequest = new ListMultipartUpload
sRequest(bucketName);
// 每次请求最多列出多少个
listMultipartUploadsRequest.setMaxUploads(100);
// 设置要查询的分块上传任务的目标前缀，直接设置要查询的 key
listMultipartUploadsRequest.setPrefix("targetKey");
MultipartUploadListing multipartUploadListing = null;
boolean found = false;
do {
multipartUploadListing = cosClient.listMultipartUploads(listMultipartUploadsReque
st);
List<MultipartUpload> multipartUploads = multipartUploadListing.getMultipartUploa
ds();
for (MultipartUpload mUpload : multipartUploads) {
if (mUpload.getKey().equals(targetKey)) {
System.out.println(mUpload.getUploadId());
found = true;
break;
}
}
if (found) {
break;
}
} while (multipartUploadListing.isTruncated());
if (!found) {
System.out.printf("do not found upload task with key: %s\n", targetKey);
}
```

参数说明

参数名称	描述	类型
listMultipartUploadsRequest	请求	ListMultipartUploadsRequest

Request 成员说明：

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
keyMarker	列出条目从该 Key 值开始	String
delimiter	定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始	String
prefix	限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	String
uploadIdMarker	列出条目从该 UploadId 值开始	String
maxUploads	设置最大返回的 multipart 数量，合法值1到1000	String
encodingType	规定返回值的编码方式，可选值：url	String

返回结果说明

- 成功：返回 MultipartUploadListing，包含正在进行分块上传的信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

上传分块

分块上传对象（Upload Part）。

方法原型

```
public UploadPartResult uploadPart(UploadPartRequest uploadPartRequest) throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。
String key = "exampleobject";
// uploadId 是一次分块上传任务的唯一标识，通过初始化分块上传或者查询分块上传任务获得
String uploadId = "exampleuploadid";
// 每个分块上传之后都会得到一个返回值 etag，保存起来用于最后合并分块时使用
```

```
List<PartETag> partETags = new LinkedList<>();
// 上传数据，这里上传 10 个 1M 的分块数据
for (int i = 1; i <= 10; i++) {
byte data[] = new byte[1024 * 1024];
// 这里创建一个 ByteArrayInputStream 来作为示例，实际中这里应该是您要上传的 InputStream 类型的流
long inputStreamLength = 1024 * 1024;
byte data[] = new byte[inputStreamLength];
InputStream inputStream = new ByteArrayInputStream(data);
UploadPartRequest uploadPartRequest = new UploadPartRequest();
uploadPartRequest.setBucketName(bucketName);
uploadPartRequest.setKey(key);
uploadPartRequest.setUploadId(uploadId);
uploadPartRequest.setInputStream(inputStream(data));
// 设置分块的长度
uploadPartRequest.setPartSize(data.length);
// 设置要上传的分块编号，从 1 开始
uploadPartRequest.setPartNumber(i);
try {
UploadPartResult uploadPartResult = cosClient.uploadPart(uploadPartRequest);
PartETag partETag = uploadPartResult.getPartETag();
partETags.add(partETag);
} catch (CosServiceException e) {
throw e;
} catch (CosClientException e) {
throw e;
}
}
System.out.println(partETags);
```

参数说明

参数名称	描述	类型
uploadPartRequest	请求	UploadPartRequest

Request 成员说明：

参数名称	设置方法	描述	类型
bucketName	set 方法	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
key	set 方法	指定分块上传到 COS 上的路径，即 对象键。例如对象键为 folder/picture.jpg	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	设置方法	描述	类型
uploadId	set 方法	标识指定分块上传的 uploadId	String
partNumber	set 方法	标识指定分块的编号，必须 >= 1	int
inputStream	set 方法	待上传分块的输入流	InputStream
trafficLimit	set 方法	用于对上传分块进行流量控制，单位：bit/s，默认不进行流量控制	Int

返回结果说明

- 成功：返回 UploadPartResult，包含上传分块的 eTag 信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

返回参数说明

UploadPartResult 类用于返回结果信息，其主要成员说明如下：

成员名称	描述	类型
partNumber	标识指定分块的编号	String
eTag	返回上传分块的 MD5 值	String
crc64Ecma	服务端根据分块内容计算出来的 CRC64	String

查询已上传的分块

查询特定分块上传操作中的已上传的块（List Parts）。

方法原型

```
public PartListing listParts(ListPartsRequest request)
throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
```

```
// 对象键(Key)是对象在存储桶中的唯一标识。
String key = "exampleobject";
// uploadId 是一次分块上传任务的唯一标识，通过初始化分块上传或者查询分块上传任务获得
String uploadId = "exampleuploadid";
// 用于保存已上传的分片信息
List<PartETag> partETags = new LinkedList<>();
PartListing partListing = null;
ListPartsRequest listPartsRequest = new ListPartsRequest(bucketName, key, uploadId);
do {
try {
partListing = cosClient.listParts(listPartsRequest);
} catch (CosServiceException e) {
throw e;
} catch (CosClientException e) {
throw e;
}
for (PartSummary partSummary : partListing.getParts()) {
partETags.add(new PartETag(partSummary.getPartNumber(), partSummary.getETag()));
}
listPartsRequest.setPartNumberMarker(partListing.getNextPartNumberMarker());
} while (partListing.isTruncated());
```

参数说明

参数名称	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID，详情请参见命名规范	String
key	构造函数或 set 方法	对象的名称	String
uploadId	构造函数或 set 方法	本次要查询的分块上传的 uploadId	String
maxParts	set 方法	单次返回最大的条目数量，默认1000	String
partNumberMarker	set 方法	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
encodingType	set 方法	规定返回值的编码方式	String

返回结果说明

- 成功：返回 PartListing，包含每一分块的 ETag 和编号，以及下一次 list 的起点 marker。

- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

完成分块上传

完成整个文件的分块上传（Complete Multipart Upload）。

> 说明：
> 完成分块上传之后，分块上传任务将删除，任务对应的 uploadId 不再有效。

方法原型

```
public CompleteMultipartUploadResult completeMultipartUpload(CompleteMultipartUploadRequest request) throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。
String key = "exampleobject";
// uploadId 是一次分块上传任务的唯一标识，通过初始化分块上传或者查询分块上传任务获得
String uploadId = "exampleuploadid";
// 保存已上传的分片信息，实际情况里，这里的内容是从上传分块接口中获取到的
List<PartETag> partETags = new LinkedList<>();
// 分片上传结束后，调用complete完成分片上传
CompleteMultipartUploadRequest completeMultipartUploadRequest =
new CompleteMultipartUploadRequest(bucketName, key, uploadId, partETags);
try {
CompleteMultipartUploadResult completeResult =
cosClient.completeMultipartUpload(completeMultipartUploadRequest);
System.out.println(completeResult.getRequestId());
} catch (CosServiceException e) {
throw e;
} catch (CosClientException e) {
throw e;
}
// 确保不再使用 cosClient 之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	指定分块上传到 COS 上的路径，即 对象键。例如对象键为 folder/picture.jpg	String
uploadId	构造函数或 set 方法	标识指定分块上传的 uploadId	String
partETags	构造函数或 set 方法	标识分块块的编号和上传返回的 eTag	List<PartETag>

返回结果说明

- 成功：返回 CompleteMultipartUploadResult，包含完成对象的 eTag 信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

终止分块上传

终止一个分块上传操作并删除已上传的分块（Abort Multipart Upload）。

> 说明：
> 终止分块上传之后，分块上传任务和已经上传的分块都会被删除，对应的 uploadId 不再有效。

方法原型

```
public void abortMultipartUpload(AbortMultipartUploadRequest request) throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。
String key = "exampleobject";
```

```
// uploadId 是一次分块上传任务的唯一标识，通过初始化分块上传或者查询分块上传任务获得
String uploadId = "exampleuploadid";
AbortMultipartUploadRequest abortMultipartUploadRequest = new AbortMultipartUploa
dRequest(bucketName, key, uploadId);
try {
cosClient.abortMultipartUpload(abortMultipartUploadRequest);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
// 确保不再使用 cosClient 之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	分块上传到 COS 上的路径，即 对象键。例如对象键为 folder/picture.jpg	String
uploadId	构造函数或 set 方法	标识指定分块上传的 uploadId	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（例如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。 详情请参见 异常处理。

下载对象

最近更新时间：2023-04-25 15:26:18

简介

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地

高级接口（推荐）

高级接口指定暂停、恢复以及取消下载请求，同时支持断点下载功能。

> 说明：
> 高级接口的下载是直接把对象写进了指定的本地文件中，如果需要一个下载流，参考本页的简单操作。

创建 TransferManager

使用高级接口的操作之前，必须先创建一个 TransferManager 的实例。

```
// 创建 TransferManager 实例，这个实例用来后续调用高级接口
TransferManager createTransferManager() {
// 创建一个 COSClient 实例，这是访问 COS 服务的基础实例。
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();

// 自定义线程池大小，建议在客户端与 COS 网络充足（例如使用腾讯云的 CVM，同地域上传 COS）的情况下，设置成16或32即可，可较充分的利用网络资源
// 对于使用公网传输且网络带宽质量不高的情况，建议减小该值，避免因网速过慢，造成请求超时。
ExecutorService threadPool = Executors.newFixedThreadPool(32);

// 传入一个 threadpool，若不传入线程池，默认 TransferManager 中会生成一个单线程的线程池。
TransferManager transferManager = new TransferManager(cosClient, threadPool);

return transferManager;
}
```

参数说明

TransferManagerConfiguration 类用于记录高级接口的配置信息，其主要成员说明如下：

成员名	设置方法	描述	类型
minimumUploadPartSize	set 方法	分块上传的块大小，单位：字节（Byte），默认为5MB	long
multipartUploadThreshold	set 方法	大于等于该值则并发的分块上传文件，单位：字节（Byte），默认为5MB	long
multipartCopyThreshold	set 方法	大于等于该值则并发的分块复制文件，单位：字节（Byte），默认为5GB	long
multipartCopyPartSize	set 方法	分块复制的块大小，单位：字节（Byte），默认为100MB	long

关闭 TransferManager

在确定不再通过 TransferManager 的实例调用高级接口之后，一定要关闭这个实例，防止资源泄露。

```
void shutdownTransferManager(TransferManager transferManager) {
// 指定参数为 true，则同时会关闭 transferManager 内部的 COSClient 实例。
// 指定参数为 false，则不会关闭 transferManager 内部的 COSClient 实例。
transferManager.shutdownNow(true);
}
```

下载对象

下载对象将 COS 上的对象下载存储到本地的一个文件中。

方法原型

```
public Download download(final GetObjectRequest getObjectRequest, final File fil
e);
```

请求示例

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 创建 TransferManager
TransferManager transferManager = createTransferManager();

// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
```

```
String key = "exampleobject";
// 本地文件路径
String localFilePath = "/path/to/localFile";
File downloadFile = new File(localFilePath);

GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
try {
// 返回一个异步结果 Download, 可同步的调用 waitForCompletion 等待下载结束, 成功返回 void, 失败抛出异常
Download download = transferManager.download(getObjectRequest, downloadFile);
download.waitForCompletion();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}

// 确定本进程不再使用 transferManager 实例之后, 关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

参数说明

参数名称	描述	类型	默认值
getObjectRequest	下载对象请求	GetObjectRequest	无
file	要下载到的本地文件	File	无

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String

Request 成员	设置方法	描述	类型
range	set 方法	下载的 range 范围	Long[]
trafficLimit	set 方法	用于对下载对象进行流量控制，单位：bit/s，默认不进行流量控制	int

返回值

- 成功：返回 Download，可以查询下载是否结束，也可同步的等待下载结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

断点下载对象

断点下载对象利用了分 range 多线程同时下载的方式下载对象，并在完成之后做完整性校验。如果下载过程中出现异常中断，重新下载时不会下载已经下载过的 range (源文件如果在重启前被修改，则会从头下载)。
适合大文件下载。

方法原型

```
public Download download(final GetObjectRequest getObjectRequest, final File fil
e,
boolean resumableDownload);

public Download download(final GetObjectRequest getObjectRequest, final File fil
e,
boolean resumableDownload, String resumableTaskFile,
int multiThreadThreshold, int partSize);
```

请求示例

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 创建 TransferManager
TransferManager transferManager = createTransferManager();

// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
// 本地文件路径
String localFilePath = "/path/to/localFile";
File downloadFile = new File(localFilePath);
```

```
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
try {
// 返回一个异步结果 Download, 可同步的调用 waitForCompletion 等待下载结束, 成功返回 void, 失败抛出异常
Download download = transferManager.download(getObjectRequest, downloadFile, true);
download.waitForCompletion();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}

// 确定本进程不再使用 transferManager 实例之后, 关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

参数说明

参数名称	描述	类型	默认值
getObjectRequest	下载对象请求	GetObjectRequest	无
file	要下载到的本地文件	File	无
resumableDownload	是否启用分块断点续传下载	boolean	false
resumableTaskFile	断点续传下载时记录信息文件名	String	file.cosresumabletask
multiThreadThreshold	断点续传下载使用多线程下载的最小文件大小	int	20 * 1024 * 1024
partSize	断点续传下载使用的分块大小	int	8 * 1024 * 1024

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String

Request 成员	设置方法	描述	类型
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
range	set 方法	下载的 range 范围	Long[]
trafficLimit	set 方法	用于对下载对象进行流量控制，单位：bit/s，默认不进行流量控制	int

返回值

- 成功：返回 Download，可以查询下载是否结束，也可同步的等待下载结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

显示下载进度

要显示下载进度首先需要一个自己的打印下载进度的函数，在这个函数里通过调用接口来获取已经成功下载的大小从而计算出当前进度。

方法原型

```
public Download download(final GetObjectRequest getObjectRequest, final File file);
```

请求示例

```
// 可以参考下面的例子，结合实际情况做调整
void showTransferProgress(Transfer transfer) {
System.out.println(transfer.getDescription());

// transfer.isDone() 查询下载是否已经完成
while (transfer.isDone() == false) {
try {
// 每 2 秒获取一次进度
Thread.sleep(2000);
} catch (InterruptedException e) {
return;
}

TransferProgress progress = transfer.getProgress();
long sofar = progress.getBytesTransferred();
```

```
long total = progress.getTotalBytesToTransfer();
double pct = progress.getPercentTransferred();
System.out.printf("upload progress: [%d / %d] = %.02f%%\n", sofar, total, pct);
}

// 完成了 Completed，或者失败了 Failed
System.out.println(transfer.getState());
}
```

结合下载文件的代码示例如下：

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 创建 TransferManager
TransferManager transferManager = createTransferManager();

// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
// 本地文件路径
String localFilePath = "/path/to/localFile";
File downloadFile = new File(localFilePath);

GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
try {
// 返回一个异步结果 Download，可同步的调用 waitForCompletion 等待下载结束，成功返回 void，
失败抛出异常
Download download = transferManager.download(getObjectRequest, downloadFile);
// 打印下载进度，直到下载结束
showTransferProgress(download);
// 这里可以捕获可能出现的异常
download.waitForCompletion();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}

// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

获取进度说明

通过 download 这个类的 getProgress 可以获得一个 TransferProgress 类，这个类的下面三个方法用来获取下载进度，说明如下：

方法名称	描述	类型
getBytesTransferred	获取已下载的字节数	long
getTotalBytesToTransfer	获取总文件的字节数	long
getPercentTransferred	获取已下载的字节百分比	double

参数说明

参数名称	描述	类型	默认值
getObjectRequest	下载对象请求	GetObjectRequest	无
file	要下载到的本地文件	File	无

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
range	set 方法	下载的 range 范围	Long[]
trafficLimit	set 方法	用于对下载对象进行流量控制，单位：bit/s，默认不进行流量控制	int

返回值

- 成功：返回 Download，可以查询下载是否结束，也可同步的等待下载结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

下载时暂停、继续与取消

使用高级接口可以暂停下载任务、随后继续下载，或者直接取消一个下载任务。

方法原型

```
public Download download(final GetObjectRequest getObjectRequest, final File file);
```

请求示例

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 示例代码：创建 TransferManager
TransferManager transferManager = createTransferManager();

// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.com/document/product/436/13324)
String key = "exampleobject";
// 本地文件路径
String localFilePath = "/path/to/localFile";
File downloadFile = new File(localFilePath);
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);

try {
// 返回一个异步结果 copy，可同步的调用 waitForCompletion 等待 download 结束，成功返回 void，失败抛出异常.
Download download = transferManager.download(getObjectRequest, downloadFile);
// 等待 3 秒，下载一部分
Thread.sleep(3000L);
// 暂停下载，获取一个 PersistableUpload 实例，用来随后的恢复
PersistableDownload persistableDownload = download.pause();
// 复杂的暂停与继续用法：
// PersistableDownload 实例也可以通过 serialize 序列化后存储起来，之后再通过 deserialize 反序列化来恢复继续下载
// persistableDownload.serialize(out);
// 继续下载
download = transferManager.resumeDownload(persistableDownload);
// 捕获可能出现的异常
download.waitForCompletion();

// 或者直接取消这次下载
// upload.abort();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
```

版权所有：腾讯云计算（北京）有限责任公司

```
} catch (InterruptedException e) {
e.printStackTrace();
}

// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 示例代码：关闭 TransferManager
shutdownTransferManager(transferManager);
```

参数说明

参数名称	描述	类型	默认值
getObjectRequest	下载对象请求	GetObjectRequest	无
file	要下载到的本地文件	File	无

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
range	set 方法	下载的 range 范围	Long[]
trafficLimit	set 方法	用于对下载对象进行流量控制，单位：bit/s，默认不进行流量控制	int

返回值

- 成功：返回 Download，可以查询下载是否结束，也可同步的等待下载结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

下载目录

下载目录可以下载 COS 上有相同前缀的一批对象（一个虚拟目录），到指定的本地目录中。下载到本地的文件，保持和在 COS 相同的目录结构。

版权所有：腾讯云计算（北京）有限责任公司

方法原型

```
public MultipleFileDownload downloadDirectory(String bucketName, String keyPrefi
x,
File destinationDirectory) {
```

请求示例

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 示例代码：创建 TransferManager
TransferManager transferManager = createTransferManager();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";

// 设置要下载的对象的前缀（相当于 cos 上的一个目录），如果设置成 ""，则下载整个 bucket。
String cos_path = "/prefix";
// 要保存下载的文件的文件夹的绝对路径
String dir_path = "/to/mydir";

try {
// 返回一个异步结果 download，可同步的调用 waitForUploadResult 等待 download 结束.
MultipleFileDownload download = transferManager.downloadDirectory(bucketName, cos
_path, new File(dir_path));

// 可以选择查看下载进度
showTransferProgress(download);

// 或者阻塞等待完成
download.waitForCompletion();

System.out.println("download directory done.");
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}

// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 示例代码：关闭 TransferManager
shutdownTransferManager(transferManager);
```

参数说明

参数名称	描述	类型
bucketName	cos 上的 bucket	GetObjectRequest
keyPrefix	cos 上 object 的前缀	String
destinationDirectory	本地的文件夹的绝对路径	File

返回值

- 成功：返回 MultipleFileUpload，可以查询下载是否结束，也可同步的等待下载结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

简单操作

简单操作由 COSClient 类型发起请求，使用简单操作之前必须先创建一个 COSClient 实例。

COSClient 实例是并发安全的，这里推荐一个进程只创建一个 COSClient 实例，当不会再通过这个实例发起请求的时候，再选择关闭这个实例。

创建 COSClient

调用 COS 的接口之前，必须先创建一个 COSClient 的实例。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 设置用户身份信息。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
String secretId = System.getenv("secretId");//用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
String secretKey = System.getenv("secretKey");//用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);

// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();

// 设置 bucket 的地域
// COS_REGION 请参见 https://cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
```

```
// 设置请求协议, http 或者 https
// 5.6.53 及更低的版本, 建议设置使用 https 协议
// 5.6.54 及更高版本, 默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);

// 以下的设置, 是可选的:

// 设置 socket 读取超时, 默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时, 默认 30s
clientConfig.setConnectionTimeout(30*1000);

// 如果需要的话, 设置 http 代理, ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);

// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

使用临时密钥创建 COSClient

如果要使用临时密钥请求 COS，则需要用临时密钥创建 COSClient。

本 SDK 并不能生成临时密钥，而需要使用额外的操作来生成，详情请参见 临时密钥生成。

```
// 创建 COSClient 实例, 这个实例用来后续调用请求
COSClient createCOSClient() {
// 这里需要已经获取到临时密钥的结果。
// 临时密钥的生成参见 https://intl.cloud.tencent.com/document/product/436/14048
String tmpSecretId = "TMPSECRETID";
String tmpSecretKey = "TMPSECRETKEY";
String sessionToken = "SESSIONTOKEN";

COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sess
ionToken);

// ClientConfig 中包含了后续请求 COS 的客户端设置:
ClientConfig clientConfig = new ClientConfig();

// 设置 bucket 的地域
// COS_REGION 请参见 https://cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));

// 设置请求协议, http 或者 https
```

```
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);

// 以下的设置，是可选的：

// 设置 socket 读取超时，默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);

// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);

// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

下载对象

下载一个 Object 到本地（GET Object）。

> 说明：
> 以下只有下载并使用流的示例，如果需要下载到文件中，直接使用本页的高级接口。

方法原型

```
public COSObject getObject(GetObjectRequest getObjectRequest)
throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();

// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
```

```
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
COSObjectInputStream cosObjectInput = null;

try {
COSObject cosObject = cosClient.getObject(getObjectRequest);
cosObjectInput = cosObject.getObjectContent();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}

// 处理下载到的流
// 这里是直接读取，按实际情况来处理
byte[] bytes = null;
try {
bytes = IOUtils.toByteArray(cosObjectInput);
} catch (IOException e) {
e.printStackTrace();
} finally {
// 用完流之后一定要调用 close()
cosObjectInput.close();
}

// 在流没有处理完之前，不能关闭 cosClient
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
getObjectRequest	下载文件请求	GetObjectRequest
destinationFile	本地的保存文件	File

Request 成员说明：

Request 成员	设置方法	描述	类型

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
range	set 方法	下载的 range 范围	Long[]
trafficLimit	set 方法	用于对下载对象进行流量控制，单位：bit/s，默认不进行流量控制	Int

返回结果说明

- 成功：返回 COSObject 类，包含输入流以及对象属性。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

返回参数说明

COSObject 类用于返回结果信息，其主要成员说明如下：

成员名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
metadata	对象的元数据	ObjectMetadata
objectContent	包含 COS 对象内容的数据流	COSObjectInputStream

版权所有：腾讯云计算（北京）有限责任公司

复制与移动对象

最近更新时间：2022-06-01 15:20:53

简介

本文档提供关于对象的复制、移动操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object - Copy	对象复制	复制文件到目标路径

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传/复制	查询正在进行中的分块上传/复制信息
Initiate Multipart Upload	初始化分块上传/复制	初始化分块上传/复制操作
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传/复制块	查询特定分块操作中的已上传/复制的块
Complete Multipart Upload	完成分块上传/复制	完成整个文件的分块上传/复制
Abort Multipart Upload	终止分块上传/复制	终止一个分块操作并删除已上传/复制的块

高级接口（推荐）

高级 API 由类 TransferManager 通过封装各个简单接口，来提供更便捷操作的接口。内部使用一个线程池，来并发接受和处理用户的请求，因此用户也可选择提交多个任务之后异步的执行任务。

TransferManager 实例是并发安全的，这里推荐一个进程只创建一个 TransferManager 实例，当不会再通过这个实例调用高级接口的时候，再选择关闭这个实例。

功能说明

高级接口封装了简单复制、分块复制接口，根据文件大小选择复制方式。

创建 TransferManager

使用高级接口的操作之前，必须先创建一个 TransferManager 的实例。

```
// 创建 TransferManager 实例，这个实例用来后续调用高级接口
TransferManager createTransferManager() {
// 创建一个 COSClient 实例，这是访问 COS 服务的基础实例。
// 这里创建的 cosClient 是以复制的目的端信息为基础的
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 自定义线程池大小，建议在客户端与 COS 网络充足（例如使用腾讯云的 CVM，同地域上传 COS）的情况下，设置成16或32即可，可较充分的利用网络资源
// 对于使用公网传输且网络带宽质量不高的情况，建议减小该值，避免因网速过慢，造成请求超时。
ExecutorService threadPool = Executors.newFixedThreadPool(32);
// 传入一个 threadpool，若不传入线程池，默认 TransferManager 中会生成一个单线程的线程池。
TransferManager transferManager = new TransferManager(cosClient, threadPool);
// 设置高级接口的配置项
// 分块复制阈值和分块大小分别为 5MB 和 1MB
TransferManagerConfiguration transferManagerConfiguration = new TransferManagerConfiguration();
transferManagerConfiguration.setMultipartCopyThreshold(5*1024*1024);
transferManagerConfiguration.setMultipartCopyPartSize(1*1024*1024);
transferManager.setConfiguration(transferManagerConfiguration);
return transferManager;
}
```

参数说明

TransferManagerConfiguration 类用于记录高级接口的配置信息，其主要成员说明如下：

成员名	设置方法	描述	类型
minimumUploadPartSize	set 方法	分块上传的块大小，单位：字节（Byte），默认为5MB	long
multipartUploadThreshold	set 方法	大于等于该值则并发的分块上传文件，单位：字节（Byte），默认为5MB	long
multipartCopyThreshold	set 方法	大于等于该值则并发的分块复制文件，单位：字节（Byte），默认为5GB	long
multipartCopyPartSize	set 方法	分块复制的块大小，单位：字节（Byte），默认为100MB	long

关闭 TransferManager

在确定不再通过 TransferManager 的实例调用高级接口之后，一定要关闭这个实例，防止资源泄露。

版权所有：腾讯云计算（北京）有限责任公司

```
void shutdownTransferManager(TransferManager transferManager) {
// 指定参数为 true, 则同时会关闭 transferManager 内部的 COSClient 实例。
// 指定参数为 false, 则不会关闭 transferManager 内部的 COSClient 实例。
transferManager.shutdownNow(true);
}
```

同地域复制

复制的源对象和目的对象属于同一个 Region。

方法原型

```
// 上传对象
public Copy copy(final CopyObjectRequest copyObjectRequest);
```

请求示例

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 创建 TransferManager
TransferManager transferManager = createTransferManager();
// 复制的源桶所在的地域
Region srcBucketRegion = new Region("ap-beijing");
// 复制的源桶，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String srcBucketName = "srcbucket-1250000000";
// 复制的源文件路径
String srcKey = "path/srckey";
// 复制的目的桶名，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String destBucketName = "destbucket-1250000000";
// 复制的目的文件路径
String destKey = "path/destkey";
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketRegion, srcB
ucketName,
srcKey, destBucketName, destKey);
try {
Copy copy = transferManager.copy(copyObjectRequest);
// 高级接口会返回一个异步结果 Copy
// 可同步的调用 waitForCopyResult 等待复制结束，成功返回 CopyResult，失败抛出异常
CopyResult copyResult = copy.waitForCopyResult();
System.out.println(copyResult.getRequestId());
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
```

```
}
// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

参数说明

参数名称	描述	类型
copyObjectRequest	复制文件请求	CopyObjectRequest

Request 成员说明：

参数名称	描述	类型
sourceBucketRegion	源 Bucket Region 。默认值：与当前 clientconfig 的 region 一致，表示统一地域复制	String
sourceBucketName	源存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
sourceKey	源对象键，对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
sourceVersionId	源文件 version id（适用于开启了版本控制的源 Bucket）。默认值：源文件当前最新版本	String
destinationBucketName	目标存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
destinationKey	目的对象键，对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
storageClass	复制的目标文件的存储类型。枚举值：Standard，Standard_IA。默认值：Standard。更多存储类型请参见 存储类型概述	String

返回值

- 成功：返回 Copy，可以查询 Copy 是否结束，也可同步的等待上传结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

版权所有：腾讯云计算（北京）有限责任公司

跨地域复制

复制的源对象和目的对象属于不同的 Region。例如：源对象在北京的桶，需要复制到上海的桶中。

> 说明：
> 金融云地域和公有云地域不互通，所以不能够进行跨地域复制操作。

方法原型

```
// 上传对象
public Copy copy(final CopyObjectRequest copyObjectRequest);
```

请求示例

```
// 这里创建以复制的目的端信息为基础
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 创建 TransferManager
TransferManager transferManager = createTransferManager();
// 复制的源桶所在的地域
Region srcBucketRegion = new Region("ap-beijing");
// 复制的源桶，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String srcBucketName = "srcbucket-1250000000";
// 复制的源文件路径
String srcKey = "path/srckey";
// 注意：复制的目的桶所在地域是在创建 TransferManager/CosClient 时通过函数：createTransf
erManager() -> createCOSClient() -> clientConfig.setRegion(dstRegion) 设置的
// 复制的目的桶名，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String destBucketName = "destbucket-1250000000";
// 复制的目的文件路径
String destKey = "path/destkey";
// 这里创建的 cosClient 是以复制的源端信息为基础的
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient srcCOSClient = createCOSClient();
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketRegion, srcB
ucketName,
srcKey, destBucketName, destKey);
try {
Copy copy = transferManager.copy(copyObjectRequest, srcCOSClient, null);
// 高级接口会返回一个异步结果 Copy
// 可同步的调用 waitForCopyResult 等待复制结束，成功返回 CopyResult，失败抛出异常
CopyResult copyResult = copy.waitForCopyResult();
} catch (CosServiceException e) {
e.printStackTrace();
```

```
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

参数说明

参数名称	描述	类型
copyObjectRequest	复制文件请求	CopyObjectRequest

Request 成员说明：

参数名称	描述	类型
sourceBucketRegion	源 Bucket Region 。默认值：与当前 clientconfig 的 region 一致，表示统一地域复制	String
sourceBucketName	源存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
sourceKey	源对象键，对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
sourceVersionId	源文件 version id（适用于开启了版本控制的源 Bucket）。默认值：源文件当前最新版本	String
destinationBucketName	目标存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
destinationKey	目的对象键，对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
storageClass	复制的目标文件的存储类型。枚举值：Standard，Standard_IA。默认值：Standard。更多存储类型请参见 存储类型概述	String

返回值

- 成功：返回 Copy，可以查询 Copy 是否结束，也可同步的等待上传结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

移动对象

移动对象操作包括两个步骤：复制源对象到目标位置，删除源对象。

由于 COS 通过存储桶名称（Bucket）和对象键（ObjectKey）来标识对象。移动对象也就意味着修改这个对象的标识，COS Java SDK 目前没有提供修改对象唯一标识名的单独接口，但是可以通过组合 **复制对象** 加上 **删除对象** 的基本操作，来达到修改对象标识的目的，从而实现移动对象。

例如将 mybucket-1250000000 这个存储桶中的 picture.jpg 这个对象移动到同个存储桶的 doc 路径下。首先可以复制 picture.jpg 对象到存储桶的 doc 路径下，即对象键设定为 doc/picture.jpg，复制完成后删除 picture.jpg 这个对象，来实现“移动”的效果。

同样的，如果想将 mybucket-1250000000 这个存储桶里的 picture.jpg 这个对象移动到 myanothorbucket-1250000000 这个存储桶里，可以先将对象复制到 myanothorbucket-1250000000 存储桶，然后删除原来的对象。

请求示例

```
// 复制的源桶所在的地域
Region srcBucketRegion = new Region("ap-beijing");
// 复制的源桶，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String srcBucketName = "srcbucket-1250000000";
// 复制的源文件路径
String srcKey = "path/srckey";
// 复制的目的桶所在的地域
Region destBucketRegion = new Region("ap-shanghai");
// 复制的目的桶名，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String destBucketName = "destbucket-1250000000";
// 复制的目的文件路径
String destKey = "path/destkey";
// 实现参考本页：高级接口 -> 复制对象
copyObject();
// 实现参考：删除对象
deleteObject();
```

简单操作

简单操作由 COSClient 类型发起请求，使用简单操作之前必须先创建一个 COSClient 实例。

COSClient 实例是并发安全的，这里推荐一个进程只创建一个 COSClient 实例，当不会再通过这个实例发起请求的时候，再选择关闭这个实例。

创建 COSClient

调用 COS 的接口之前，必须先创建一个 COSClient 的实例。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 设置用户身份信息。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
String secretId = "SECRETID";
String secretKey = "SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

使用临时密钥创建 COSClient

如果要使用临时密钥请求 COS，则需要用临时密钥创建 COSClient。
本 SDK 并不能生成临时密钥，而需要使用额外的操作来生成，参考 临时密钥生成。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 这里需要已经获取到临时密钥的结果。
// 临时密钥的生成参考 https://intl.cloud.tencent.com/document/product/436/14048
String tmpSecretId = "TMPSECRETID";
```

```
String tmpSecretKey = "TMPSECRETKEY";
String sessionToken = "SESSIONTOKEN";
COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sess
ionToken);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议, http 或者 https
// 5.6.53 及更低的版本, 建议设置使用 https 协议
// 5.6.54 及更高版本, 默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置, 是可选的：
// 设置 socket 读取超时, 默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时, 默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话, 设置 http 代理, ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

复制对象

复制文件到目标路径（PUT Object-Copy）。

> 说明：
> 推荐直接使用高级接口的复制。

方法原型

```
public CopyObjectResult copyObject(CopyObjectRequest copyObjectRequest)
throws CosClientException, CosServiceException
```

请求示例

```
// 复制的源桶所在的地域
Region srcBucketRegion = new Region("ap-beijing");
// 复制的源桶, 命名格式为 BucketName-APPID, 此处填写的存储桶名称必须为此格式
```

```
String srcBucketName = "srcbucket-1250000000";
// 复制的源文件路径
String srcKey = "path/srckey";
// 复制的目的桶所在的地域
Region destBucketRegion = new Region("ap-beijing");
// 复制的目的桶名，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String destBucketName = "destbucket-1250000000";
// 复制的目的文件路径
String destKey = "path/destkey";
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 这里创建的 cosClient 是以复制的目的端信息为基础的
// 详细代码参见本页：简单上传 -> 创建 COSClient
COSClient cosClient = createCOSClient();
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketRegion, srcB
ucketName,
srcKey, destBucketName, destKey);
try {
CopyObjectResult copyObjectResult = cosClient.copyObject(copyObjectRequest);
System.out.println(copyObjectResult.getRequestId());
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
copyObjectRequest	拷贝文件请求	CopyObjectRequest

Request 成员说明：

参数名称	描述	类型
sourceBucketRegion	源 Bucket region。默认值：与当前 clientConfig 的 region 一致，表示同地域拷贝	String
sourceBucketName	源存储桶名称，命名格式为 BucketName-APPID，详情请参见 命名规范	String

参数名称	描述	类型
sourceKey	源对象键，对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
sourceVersionId	源文件 version id（适用于开启了版本控制的源 Bucket）。默认值：源文件当前最新版本	String
destinationBucketName	目标存储桶名称，Bucket 的命名格式为 BucketName-APPID ，name 由字母数字和中划线构成	String
destinationKey	目的对象键，对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
storageClass	拷贝的目的文件的存储类型。枚举值：Standard，Standard_IA。默认值：Standard。更多存储类型请参见 存储类型概述	String

返回结果说明

- 成功：返回 CopyObjectResult，包含新文件的 Etag 等信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

分块操作

大文件复制时，使用一系列小分块复制，改善大文件上传耗时太久可能导致的各种中断问题。

> 说明：
> 推荐直接使用高级接口的复制。

操作流程

分块复制的流程

1. 初始化分块复制（Initiate Multipart Upload），得到 UploadId
2. 使用 UploadId 复制分块（Upload Part）
3. 完成分块复制（Complete Multipart Upload）

分块继续复制的流程

1. 如果没有记录 UploadId，则要先查询分块复制任务（List Multipart Uploads），得到对应文件的 UploadId
2. 使用 UploadId 列出已复制的分块（List Parts）
3. 使用 UploadId 复制剩余的分块（Upload Part）
4. 完成分块复制（Complete Multipart Upload）

终止分块复制的流程

1. 如果没有记录 UploadId，则查询分块复制任务（List Multipart Uploads），得到对应文件的 UploadId
2. 终止分块复制并删除已复制分块（Abort Multipart Upload）

初始化分块复制

初始化分块复制操作，获取对应的 uploadId 用于后续的操作。

方法原型

```
public InitiateMultipartUploadResult initiateMultipartUpload(
InitiateMultipartUploadRequest request) throws CosClientException, CosServiceExce
ption;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 这里创建的 cosClient 是以复制的目的端信息为基础的
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(bucke
tName, key);
// 分块复制的过程中，仅能通过初始化分块指定文件复制之后的metadata
// 需要的头部可以在这里指定
ObjectMetadata objectMetadata = new ObjectMetadata();
request.setObjectMetadata(objectMetadata);
try {
InitiateMultipartUploadResult initResult = cosClient.initiateMultipartUpload(requ
est);
// 获取uploadid
String uploadId = initResult.getUploadId();
System.out.println(uploadId);
} catch (CosServiceException e) {
```

```
throw e;
} catch (CosClientException e) {
throw e;
}
```

参数说明

参数名称	描述	类型
initiateMultipartUploadRequest	请求	InitiateMultipartUploadRequest

Request 成员说明：

参数名称	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	指定分块上传到 COS 上的路径，即 对象键。例如对象键为 folder/picture.jpg	String

返回结果说明

- 成功：返回 InitiateMultipartUploadResult，包含标志本次分块上传的 uploadId。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询分块复制任务

查询分块复制任务（List Multipart Uploads），获取当前正在进行的所有复制任务，从中找到需要处理的 uploadId。

方法原型

```
public MultipartUploadListing listMultipartUploads(
ListMultipartUploadsRequest listMultipartUploadsRequest)
throws CosClientException, CosServiceException
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 这里创建的 cosClient 是以复制的目的端信息为基础的
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
```

```
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 需要查询的分块复制任务对应的 key
String targetKey = "targetKey";
ListMultipartUploadsRequest listMultipartUploadsRequest = new ListMultipartUpload
sRequest(bucketName);
// 每次请求最多列出多少个
listMultipartUploadsRequest.setMaxUploads(100);
// 设置要查询的分块复制任务的目标前缀，直接设置要查询的 key
listMultipartUploadsRequest.setPrefix("targetKey");
MultipartUploadListing multipartUploadListing = null;
boolean found = false;
do {
multipartUploadListing = cosClient.listMultipartUploads(listMultipartUploadsReque
st);
List<MultipartUpload> multipartUploads = multipartUploadListing.getMultipartUploa
ds();
for (MultipartUpload mUpload : multipartUploads) {
if (mUpload.getKey().equals(targetKey)) {
System.out.println(mUpload.getUploadId());
found = true;
break;
}
}
if (found) {
break;
}
} while (multipartUploadListing.isTruncated());
if (!found) {
System.out.printf("do not found upload task with key: %s\n", targetKey);
}
```

参数说明

参数名称	描述	类型
listMultipartUploadsRequest	请求	ListMultipartUploadsRequest

Request 成员说明：

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
keyMarker	列出条目从该 Key 值开始	String

参数名称	描述	类型
delimiter	定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始	String
prefix	限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	String
uploadIdMarker	列出条目从该 UploadId 值开始	String
maxUploads	设置最大返回的 multipart 数量，合法值1到1000	String
encodingType	规定返回值的编码方式，可选值：url	String

返回结果说明

- 成功：返回 MultipartUploadListing，包含正在进行分块上传的信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

复制分块

分块复制对象（Upload Part - Copy）。

方法原型

```
public CopyPartResult copyPart(CopyPartRequest copyPartRequest) throws CosClientException, CosServiceException
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 这里创建的 cosClient 是以复制的目的端信息为基础的
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.com/document/product/436/13324)
String key = "exampleobject";
// uploadId 是一次分块复制任务的唯一标识，通过初始化分块复制或者查询分块复制任务获得
String uploadId = "exampleuploadid";
// 复制的源桶所在的地域
```

```
Region srcBucketRegion = new Region("ap-beijing");
// 复制的源桶，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String srcBucketName = "srcbucket-1250000000";
// 复制的源文件路径
String srcKey = "path/srckey";
// 复制的目的桶所在的地域
Region destBucketRegion = new Region("ap-beijing");
// 复制的目的桶名，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String destBucketName = "destbucket-1250000000";
// 复制的目的文件路径
String destKey = "path/destkey";
CopyPartRequest copyPartRequest = new CopyPartRequest();
copyPartRequest.setSourceBucketRegion(srcBucketRegion);
copyPartRequest.setSourceBucketName(srcBucketName);
copyPartRequest.setSourceKey(srcKey);
// 指定要拷贝的源文件的数据范围(类似content-range)
copyPartRequest.setFirstByte(0L);
copyPartRequest.setLastByte(1048575L);
copyPartRequest.setDestinationBucketName(destBucketName);
copyPartRequest.setDestinationKey(destKey);
// 设置本次复制的分块的序号，从 1 开始
copyPartRequest.setPartNumber(1);
copyPartRequest.setUploadId(uploadId);
try {
CopyPartResult copyPartResult = cosClient.copyPart(copyPartRequest);
PartETag partETag = copyPartResult.getPartETag();
System.out.println(partETag);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
```

参数说明

参数名称	描述	类型
copyPartRequest	请求	CopyPartRequest

Request 成员说明：

参数名称	设置方法	描述	类型
destinationBucketName	set 方法	目标存储桶名称，Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String

参数名称	设置方法	描述	类型
destinationKey	set 方法	目标对象名称，指定分块被复制后所存放的 COS 路径，即 对象键。例如对象键为 folder/picture.jpg	String
uploadId	set 方法	标识指定分块上传的 uploadId	String
partNumber	set 方法	标识指定分块的编号，必须 >= 1	int
sourceBucketRegion	set 方法	源存储桶的地域	Region
sourceBucketName	set 方法	源存储桶的名称	String
sourceKey	set 方法	源对象名称，分块在复制前所在的 COS 上的路径，即 对象键。例如对象键为 folder/picture.jpg	String
firstByte	set 方法	源对象的首字节偏移	Long
lastByte	set 方法	源对象的最后一字节偏移	Long

返回结果说明

- 成功：返回 CopyPartResult，包含分块的 ETag 信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询已复制的分块

查询特定分块复制操作中的已复制的块（List Parts）。

方法原型

```
public PartListing listParts(ListPartsRequest request)
throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 这里创建的 cosClient 是以复制的目的端信息为基础的
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
```

```
// uploadId 是一次分块复制任务的唯一标识，通过初始化分块复制或者查询分块复制任务获得
String uploadId = "exampleuploadid";
// 用于保存已复制的分片信息
List<PartETag> partETags = new LinkedList<>();
PartListing partListing = null;
ListPartsRequest listPartsRequest = new ListPartsRequest(bucketName, key, uploadI
d);
do {
try {
partListing = cosClient.listParts(listPartsRequest);
} catch (CosServiceException e) {
throw e;
} catch (CosClientException e) {
throw e;
}
for (PartSummary partSummary : partListing.getParts()) {
partETags.add(new PartETag(partSummary.getPartNumber(), partSummary.getETag()));
}
listPartsRequest.setPartNumberMarker(partListing.getNextPartNumberMarker());
} while (partListing.isTruncated());
```

参数说明

参数名称	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID ，详情请参见命名规范	String
key	构造函数或 set 方法	对象的名称	String
uploadId	构造函数或 set 方法	本次要查询的分块上传的 uploadId	String
maxParts	set 方法	单次返回最大的条目数量，默认1000	String
partNumberMarker	set 方法	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
encodingType	set 方法	规定返回值的编码方式	String

返回结果说明

- 成功：返回 PartListing，包含每一分块的 ETag 和编号，以及下一次 list 的起点 marker。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

版权所有：腾讯云计算（北京）有限责任公司

完成分块复制

完成整个文件的分块复制（Complete Multipart Upload）。

> 说明：
> 完成分块复制之后，分块复制任务将删除，任务对应的 uploadId 不再有效。

方法原型

```
public CompleteMultipartUploadResult completeMultipartUpload(CompleteMultipartUploadRequest request) throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 这里创建的 cosClient 是以复制的目的端信息为基础的
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.com/document/product/436/13324)
String key = "exampleobject";
// uploadId 是一次分块复制任务的唯一标识，通过初始化分块复制或者查询分块复制任务获得
String uploadId = "exampleuploadid";
// 保存已上传的分片信息，实际情况里，这里的内容是从上传分块接口中获取到的
List<PartETag> partETags = new LinkedList<>();
// 分片上传结束后，调用complete完成分片复制
CompleteMultipartUploadRequest completeMultipartUploadRequest =
new CompleteMultipartUploadRequest(bucketName, key, uploadId, partETags);
try {
CompleteMultipartUploadResult completeResult =
cosClient.completeMultipartUpload(completeMultipartUploadRequest);
System.out.println(completeResult.getRequestId());
} catch (CosServiceException e) {
throw e;
} catch (CosClientException e) {
throw e;
}
// 确保不再使用 cosClient 之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	指定分块上传到 COS 上的路径，即 对象键。例如对象键为 folder/picture.jpg	String
uploadId	构造函数或 set 方法	标识指定分块上传的 uploadId	String
partETags	构造函数或 set 方法	标识分块块的编号和上传返回的 eTag	List<PartETag>

返回结果说明

- 成功：返回 CompleteMultipartUploadResult，包含完成对象的 eTag 信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

终止分块复制

终止一个分块复制操作并删除已复制的分块（Abort Multipart Upload）。

> 说明：
> 终止分块复制之后，分块复制任务和已经复制的分块都会被删除，对应的 uploadId 不再有效。

方法原型

```
public void abortMultipartUpload(AbortMultipartUploadRequest request) throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 这里创建的 cosClient 是以复制的目的端信息为基础的
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.com/document/product/436/13324)
```

```
String key = "exampleobject";
// uploadId 是一次分块复制任务的唯一标识，通过初始化分块复制或者查询分块复制任务获得
String uploadId = "exampleuploadid";
AbortMultipartUploadRequest abortMultipartUploadRequest = new AbortMultipartUploa
dRequest(bucketName, key, uploadId);
try {
cosClient.abortMultipartUpload(abortMultipartUploadRequest);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
// 确保不再使用 cosClient 之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	分块上传到 COS 上的路径，即 对象键。例如对象键为 folder/picture.jpg	String
uploadId	构造函数或 set 方法	标识指定分块上传的 uploadId	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（例如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。 详情请参见 异常处理。

列出对象

最近更新时间：2022-04-07 15:06:45

简介

本文档提供关于列出对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象
GET Bucket Object Versions	查询对象及其历史版本列表	查询存储桶下的部分或者全部对象及其历史版本信息

简单操作

简单操作由 COSClient 类型发起请求，使用简单操作之前必须先创建一个 COSClient 实例。

COSClient 实例是并发安全的，这里推荐一个进程只创建一个 COSClient 实例，当不会再通过这个实例发起请求的时候，再选择关闭这个实例。

创建 COSClient

调用 COS 的接口之前，必须先创建一个 COSClient 的实例。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 设置用户身份信息。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
String secretId = "SECRETID";
String secretKey = "SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议, http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
```

```
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

使用临时密钥创建 COSClient

如果要使用临时密钥请求 COS，则需要用临时密钥创建 COSClient。

本 SDK 并不能生成临时密钥，而需要使用额外的操作来生成，参考 临时密钥生成。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 这里需要已经获取到临时密钥的结果。
// 临时密钥的生成参考 https://intl.cloud.tencent.com/document/product/436/14048
String tmpSecretId = "TMPSECRETID";
String tmpSecretKey = "TMPSECRETKEY";
String sessionToken = "SESSIONTOKEN";
COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sess
ionToken);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
```

```
return new COSClient(cred, clientConfig);
}
```

列出第一页对象

列出存储桶下的部分对象。

方法原型

```
public ObjectListing listObjects(ListObjectsRequest listObjectsRequest) throws Co
sClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
// 设置 bucket 名称
listObjectsRequest.setBucketName(bucketName);
// 设置列出的对象名以 prefix 为前缀
listObjectsRequest.setPrefix("");
// 设置最大列出多少个对象，一次 listobject 最大支持1000
listObjectsRequest.setMaxKeys(1000);
// 保存列出的结果
ObjectListing objectListing = null;
try {
objectListing = cosclient.listObjects(listObjectsRequest);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
// object summary 表示此次列出的对象列表
List<COSObjectSummary> cosObjectSummaries = objectListing.getObjectSummaries();
for (COSObjectSummary cosObjectSummary : cosObjectSummaries) {
// 对象的 key
String key = cosObjectSummary.getKey();
// 对象的etag
String etag = cosObjectSummary.getETag();
// 对象的长度
long fileSize = cosObjectSummary.getSize();
// 对象的存储类型
String storageClasses = cosObjectSummary.getStorageClass();
```

版权所有：腾讯云计算（北京）有限责任公司

```
}
if (objectListing.isTruncated()) {
// 表示还没有列完，被截断了
// 下一次开始的位置
String nextMarker = objectListing.getNextMarker();
}
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
listObjectsRequest	获取文件列表请求	ListObjectsRequest

Request 成员说明 ：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
prefix	构造函数或 set 方法	限制返回的结果对象，以 prefix 为前缀。默认不进行限制，即 Bucket 下所有的成员。 默认值为 `""` ，表示空	String
marker	构造函数或 set 方法	标记 list 的起点位置，第一次可设置为空，后续请求需设置为上一次 listObjects 返回值中的 nextMarker	String
delimiter	构造函数或 set 方法	分隔符，限制返回的是以 prefix 开头，并以 delimiter 第一次出现的结束的路径	String
maxKeys	构造函数或 set 方法	最大返回的成员个数（不得超过1000）。 默认值： 1000	Integer

返回结果说明

- 成功：返回 ObjectListing 类型， 包含所有的成员， 以及 nextMarker。
- 失败：抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

列出第二页对象

列出被截断的剩下的部分对象。

方法原型

```
public ObjectListing listObjects(ListObjectsRequest listObjectsRequest) throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 下一次开始的位置
// 通过被截断的列对象请求获取，参考上面的 列出第一页对象
String nextMarker = "nextMarker";
ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
// 设置 bucket 名称
listObjectsRequest.setBucketName(bucketName);
// 设置列出的对象名以 prefix 为前缀
listObjectsRequest.setPrefix("");
// 设置最大列出多少个对象，一次 listobject 最大支持1000
listObjectsRequest.setMaxKeys(1000);
// 设置被截断开始的位置
listObjectsRequest.setMarker(nextMarker);
// 保存列出的结果
ObjectListing objectListing = null;
try {
objectListing = cosclient.listObjects(listObjectsRequest);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
// object summary 表示此次列出的对象列表
List<COSObjectSummary> cosObjectSummaries = objectListing.getObjectSummaries();
for (COSObjectSummary cosObjectSummary : cosObjectSummaries) {
// 对象的 key
String key = cosObjectSummary.getKey();
// 对象的etag
String etag = cosObjectSummary.getETag();
// 对象的长度
long fileSize = cosObjectSummary.getSize();
// 对象的存储类型
String storageClasses = cosObjectSummary.getStorageClass();
}
if (objectListing.isTruncated()) {
// 表示还没有列完，被截断了
```

```
// 下一次开始的位置
String nextMarker = objectListing.getNextMarker();
}
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
listObjectsRequest	获取文件列表请求	ListObjectsRequest

Request 成员说明 ：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
prefix	构造函数或 set 方法	限制返回的结果对象，以 prefix 为前缀。默认不进行限制，即 Bucket 下所有的成员。 默认值为 `""` ，表示空	String
marker	构造函数或 set 方法	标记 list 的起点位置，第一次可设置为空，后续请求需设置为上一次 listObjects 返回值中的 nextMarker	String
delimiter	构造函数或 set 方法	分隔符，限制返回的是以 prefix 开头，并以 delimiter 第一次出现的结束的路径	String
maxKeys	构造函数或 set 方法	最大返回的成员个数（不得超过1000）。 默认值： 1000	Integer

返回结果说明

- 成功：返回 ObjectListing 类型， 包含所有的成员， 以及 nextMarker。
- 失败：抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

列出目录下的对象和子目录

方法原型

```
public ObjectListing listObjects(ListObjectsRequest listObjectsRequest) throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
// 设置 bucket 名称
listObjectsRequest.setBucketName(bucketName);
// prefix 表示列出的对象名以 prefix 为前缀
// 这里填要列出的目录的相对 bucket 的路径
listObjectsRequest.setPrefix("/dir/");
// delimiter 表示目录的截断符，例如：设置为 / 则表示对象名遇到 / 就当做一级目录）
listObjectsRequest.setDelimiter("/");
// 设置最大遍历出多少个对象，一次 listobject 最大支持1000
listObjectsRequest.setMaxKeys(1000);
// 保存每次列出的结果
ObjectListing objectListing = null;
do {
try {
objectListing = cosclient.listObjects(listObjectsRequest);
} catch (CosServiceException e) {
e.printStackTrace();
return;
} catch (CosClientException e) {
e.printStackTrace();
return;
}
// 这里保存列出来的子目录
List<String> commonPrefixes = objectListing.getCommonPrefixes();
for (String commonPrefix : commonPrefixes) {
System.out.println(commonPrefix);
}
// 这里保存列出的对象列表
List<COSObjectSummary> cosObjectSummaries = objectListing.getObjectSummaries();
for (COSObjectSummary cosObjectSummary : cosObjectSummaries) {
// 对象的 key
String key = cosObjectSummary.getKey();
// 对象的etag
String etag = cosObjectSummary.getETag();
// 对象的长度
long fileSize = cosObjectSummary.getSize();
// 对象的存储类型
String storageClasses = cosObjectSummary.getStorageClass();
}
// 标记下一次开始的位置
```

```
String nextMarker = objectListing.getNextMarker();
listObjectsRequest.setMarker(nextMarker);
} while (objectListing.isTruncated());
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
listObjectsRequest	获取文件列表请求	ListObjectsRequest

Request 成员说明 ：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
prefix	构造函数或 set 方法	限制返回的结果对象，以 prefix 为前缀。默认不进行限制，即 Bucket 下所有的成员。 默认值为 `""` ，表示空	String
marker	构造函数或 set 方法	标记 list 的起点位置，第一次可设置为空，后续请求需设置为上一次 listObjects 返回值中的 nextMarker	String
delimiter	构造函数或 set 方法	分隔符，限制返回的是以 prefix 开头，并以 delimiter 第一次出现的结束的路径	String
maxKeys	构造函数或 set 方法	最大返回的成员个数（不得超过1000）。 默认值： 1000	Integer

返回结果说明

- 成功：返回 ObjectListing 类型， 包含所有的成员， 以及 nextMarker。
- 失败：抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

列出对象历史版本

列出对象的每一个历史版本。

> 说明：
> 需要存储桶开启版本控制。

方法原型

```
public VersionListing listVersions(ListVersionsRequest listVersionsRequest)
throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
ListVersionsRequest listVersionsRequest = new ListVersionsRequest();
// 设置 bucket 名称
listVersionsRequest.setBucketName(bucketName);
// prefix 表示列出的对象名以 prefix 为前缀
// 这里填要列出的目录的相对 bucket 的路径
listVersionsRequest.setPrefix("");
// delimiter 表示目录的截断符，例如：设置为 / 则表示对象名遇到 / 就当做一级目录）
listVersionsRequest.setDelimiter("/");
// 设置最大遍历出多少个对象，一次 listobject 最大支持1000
listVersionsRequest.setMaxKeys(1000);
VersionListing versionListing = null;
do {
try {
versionListing = cosclient.listVersions(listVersionsRequest);
} catch (CosServiceException e) {
e.printStackTrace();
return;
} catch (CosClientException e) {
e.printStackTrace();
return;
}
List<COSVersionSummary> cosVersionSummaries = versionListing.getVersionSummaries
();
for (COSVersionSummary cosVersionSummary : cosVersionSummaries) {
System.out.println(cosVersionSummary.getKey() + ":" + cosVersionSummary.getVersio
nId());
}
// 下一次开始的位置有 2 个标志，一个是对象名，一个是对象版本
String keyMarker = versionListing.getNextKeyMarker();
String versionIdMarker = versionListing.getNextVersionIdMarker();
listVersionsRequest.setKeyMarker(keyMarker);
listVersionsRequest.setVersionIdMarker(versionIdMarker);
```

版权所有：腾讯云计算（北京）有限责任公司

```
} while (versionListing.isTruncated());
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
listVersionsRequest	获取对象版本信息请求	ListVersionsRequest

Request 成员说明 ：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
prefix	构造函数或 set 方法	限制返回的结果对象，以 prefix 为前缀。默认不进行限制，即 Bucket 下所有的成员。<默认值为 `""` ，表示空	String
keyMarker	构造函数或 set 方法	标记 list 的起点位置，第一次可设置为空，后续请求需设置为上一次 listObjects 返回值中的 nextKeyMarker	String
versionIdMarker	构造函数或 set 方法	标记 list 的起点位置，第一次可设置为空，后续请求需设置为上一次 listObjects 返回值中的 nextVersionIdMarker	String
delimiter	构造函数或 set 方法	分隔符，限制返回的是以 prefix 开头，并以 delimiter 第一次出现的结束的路径	String
maxResults	构造函数或 set 方法	最大返回的成员个数（不得超过1000）。默认值： 1000	Integer

返回结果说明

- 成功：返回 ObjectListing 类型， 包含所有的成员， 以及 nextKeyMarker 和 nextVersionIdMarker。
- 失败：抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

版权所有：腾讯云计算（北京）有限责任公司

删除对象

最近更新时间：2023-02-23 16:51:36

简介

本文档提供关于对象的删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

简单操作

简单操作由 COSClient 类型发起请求，使用简单操作之前必须先创建一个 COSClient 实例。

COSClient 实例是并发安全的，这里推荐一个进程只创建一个 COSClient 实例，当不会再通过这个实例发起请求的时候，再选择关闭这个实例。

创建 COSClient

调用 COS 的接口之前，必须先创建一个 COSClient 的实例。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 设置用户身份信息。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
String secretId = System.getenv("secretId");//用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
String secretKey = System.getenv("secretKey");//用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);

// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();

// 设置 bucket 的地域
```

```
// COS_REGION 请参见 https://cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));

// 设置请求协议, http 或者 https
// 5.6.53 及更低的版本, 建议设置使用 https 协议
// 5.6.54 及更高版本, 默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);

// 以下的设置, 是可选的:

// 设置 socket 读取超时, 默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时, 默认 30s
clientConfig.setConnectionTimeout(30*1000);

// 如果需要的话, 设置 http 代理, ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);

// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

使用临时密钥创建 COSClient

如果要使用临时密钥请求 COS，则需要用临时密钥创建 COSClient。
本 SDK 并不能生成临时密钥，而需要使用额外的操作来生成，详情请参见 临时密钥生成。

```
// 创建 COSClient 实例, 这个实例用来后续调用请求
COSClient createCOSClient() {
// 这里需要已经获取到临时密钥的结果。
// 临时密钥的生成参见 https://intl.cloud.tencent.com/document/product/436/14048
String tmpSecretId = "TMPSECRETID";
String tmpSecretKey = "TMPSECRETKEY";
String sessionToken = "SESSIONTOKEN";

COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sess
ionToken);

// ClientConfig 中包含了后续请求 COS 的客户端设置:
ClientConfig clientConfig = new ClientConfig();

// 设置 bucket 的地域
// COS_REGION 请参见 https://cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
```

```
// 设置请求协议, http 或者 https
// 5.6.53 及更低的版本, 建议设置使用 https 协议
// 5.6.54 及更高版本, 默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);

// 以下的设置, 是可选的:

// 设置 socket 读取超时, 默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时, 默认 30s
clientConfig.setConnectionTimeout(30*1000);

// 如果需要的话, 设置 http 代理, ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);

// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

删除对象

删除指定的对象（DELETE Object）。

方法原型

```
public void deleteObject(String bucketName, String key)
throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例, 如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();

// 存储桶的命名格式为 BucketName-APPID, 此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";

try {
cosClient.deleteObject(bucketName, key);
} catch (CosServiceException e) {
e.printStackTrace();
```

```
} catch (CosClientException e) {
e.printStackTrace();
}

// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

批量删除对象

删除指定的一批对象（DELETE Multiple Objects）。

方法原型

```
public DeleteObjectsResult deleteObjects(DeleteObjectsRequest deleteObjectsReques
t)
throws MultiObjectDeleteException, CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();

// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";

DeleteObjectsRequest deleteObjectsRequest = new DeleteObjectsRequest(bucketName);
// 设置要删除的key列表，最多一次删除1000个
```

```
ArrayList<KeyVersion> keyList = new ArrayList<>();
// 传入要删除的文件名
// 注意文件名不允许以正斜线/或者反斜线\开头，例如：
// 存储桶目录下有a/b/c.txt文件，如果要删除，只能是 keyList.add(new KeyVersion("a/b/c.txt"))，若使用 keyList.add(new KeyVersion("/a/b/c.txt"))会导致删除不成功
keyList.add(new KeyVersion("aaa"));
keyList.add(new KeyVersion("bbb"));
keyList.add(new KeyVersion("ccc"));
deleteObjectsRequest.setKeys(keyList);

try {
DeleteObjectsResult deleteObjectsResult = cosClient.deleteObjects(deleteObjectsRequest);
List<DeletedObject> deleteObjectResultArray = deleteObjectsResult.getDeletedObjects();
} catch (MultiObjectDeleteException mde) {
// 如果部分删除成功部分失败，返回 MultiObjectDeleteException
List<DeletedObject> deleteObjects = mde.getDeletedObjects();
List<DeleteError> deleteErrors = mde.getErrors();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}

// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
deleteObjectsRequest	请求	DeleteObjectsRequest

Request 成员说明：

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String
quiet	指明删除的返回结果方式，可选值为 true，false，默认为 false。设置为 true 只返回失败的错误信息，设置为 false 时返回成功和失败的所有信息	boolean

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
keys	对象路径列表，对象的版本号为可选	List<DeleteObjectsRequest.KeyVersion>

DeleteObjectsRequest.KeyVersion 成员说明：

参数名称	描述	类型
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
version	在开启存储桶版本控制时，指定被删除对象的版本号，可选	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException，详情请参见 异常处理。

批量删除指定版本的对象

删除指定的一批带版本号的对象（DELETE Multiple Objects）。

> 说明：
> 需要对应的 bucket 开启版本控制。

方法原型

```
public DeleteObjectsResult deleteObjects(DeleteObjectsRequest deleteObjectsRequest)
throws MultiObjectDeleteException, CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();

// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
```

```
String bucketName = "examplebucket-1250000000";

DeleteObjectsRequest deleteObjectsRequest = new DeleteObjectsRequest(bucketName);
// 设置要删除的 key 列表, 最多一次删除1000个
ArrayList<KeyVersion> keyList = new ArrayList<>();
// 传入要删除的文件名和对应的版本号
// 版本号可以通过：列出对象 -> 简单操作 -> 列出对象历史版本 查到
keyList.add(new KeyVersion("aaa", "aaa versionId"));
keyList.add(new KeyVersion("bbb", "bbb versionId"));
keyList.add(new KeyVersion("ccc", "ccc versionId"));
deleteObjectsRequest.setKeys(keyList);

try {
DeleteObjectsResult deleteObjectsResult = cosClient.deleteObjects(deleteObjectsRe
quest);
List<DeletedObject> deleteObjectResultArray = deleteObjectsResult.getDeletedObjec
ts();
} catch (MultiObjectDeleteException mde) {
// 如果部分删除成功部分失败, 返回 MultiObjectDeleteException
List<DeletedObject> deleteObjects = mde.getDeletedObjects();
List<DeleteError> deleteErrors = mde.getErrors();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}

// 确认本进程不再使用 cosClient 实例之后, 关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
deleteObjectsRequest	请求	DeleteObjectsRequest

Request 成员说明：

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String

参数名称	描述	类型
quiet	指明删除的返回结果方式，可选值为 true，false，默认为 false。设置为 true 只返回失败的错误信息，设置为 false 时返回成功和失败的所有信息	boolean
keys	对象路径列表，对象的版本号为可选	List<DeleteObjectsRequest.KeyVersion>

DeleteObjectsRequest.KeyVersion 成员说明：

参数名称	描述	类型
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
version	在开启存储桶版本控制时，指定被删除对象的版本号，可选	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException，详情请参见 异常处理。

删除目录

删除目录操作包括两个步骤：列出要删除的目录下的所有对象，使用批量删除对象。

```
// 列目录实现参考：列出对象 -> 简单操作 -> 列出目录下的对象和子目录
// 批量删除实现参考本页：批量删除对象

// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();

// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";

// 要删除的目录，这里是相对于 bucket 的路径
String delDir = "exampledir";

ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
// 设置 bucket 名称
```

```
listObjectsRequest.setBucketName(bucketName);
// prefix 表示列出的对象名以 prefix 为前缀
// 这里填要列出的目录的相对 bucket 的路径
listObjectsRequest.setPrefix(delDir);
// 设置最大遍历出多少个对象, 一次 listobject 最大支持1000
listObjectsRequest.setMaxKeys(1000);

// 保存每次列出的结果
ObjectListing objectListing = null;

do {
try {
objectListing = cosClient.listObjects(listObjectsRequest);
} catch (CosServiceException e) {
e.printStackTrace();
return;
} catch (CosClientException e) {
e.printStackTrace();
return;
}

// 这里保存列出的对象列表
List<COSObjectSummary> cosObjectSummaries = objectListing.getObjectSummaries();

ArrayList<KeyVersion> delObjects = new ArrayList<KeyVersion>();

for (COSObjectSummary cosObjectSummary : cosObjectSummaries) {
delObjects.add(new KeyVersion(cosObjectSummary.getKey()));
}

DeleteObjectsRequest deleteObjectsRequest = new DeleteObjectsRequest(bucketName);

deleteObjectsRequest.setKeys(delObjects);

try {
DeleteObjectsResult deleteObjectsResult = cosClient.deleteObjects(deleteObjectsRe
quest);
List<DeletedObject> deleteObjectResultArray = deleteObjectsResult.getDeletedObjec
ts();
} catch (MultiObjectDeleteException mde) {
// 如果部分删除成功部分失败, 返回 MultiObjectDeleteException
List<DeletedObject> deleteObjects = mde.getDeletedObjects();
List<DeleteError> deleteErrors = mde.getErrors();
} catch (CosServiceException e) {
e.printStackTrace();
return;
} catch (CosClientException e) {
```

```
e.printStackTrace();
return;
}

// 标记下一次开始的位置
String nextMarker = objectListing.getNextMarker();
listObjectsRequest.setMarker(nextMarker);
} while (objectListing.isTruncated());
```

判断对象是否存在

最近更新时间：2022-01-27 13:10:20

简介

本文档提供快捷查询存储桶中某个对象是否存在的示例代码。示例代码实际调用了 HEAD Object COS API，是该接口的简化版。

HEAD Object 除了检查对象是否存在，主要功能为返回对象元数据。如需查看包含了 HEAD Object 完整功能的 SDK 接口，请参见 查询对象元数据。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

简单操作

简单操作由 COSClient 类型发起请求，使用简单操作之前必须先创建一个 COSClient 实例。

COSClient 实例是并发安全的，这里推荐一个进程只创建一个 COSClient 实例，当不会再通过这个实例发起请求的时候，再选择关闭这个实例。

创建 COSClient

调用 COS 的接口之前，必须先创建一个 COSClient 的实例。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 设置用户身份信息。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.co
m/cam/capi 进行查看和管理
String secretId = "SECRETID";
String secretKey = "SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
```

```
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

使用临时密钥创建 COSClient

如果要使用临时密钥请求 COS，则需要用临时密钥创建 COSClient。

本 SDK 并不能生成临时密钥，而需要使用额外的操作来生成，参考 临时密钥生成。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 这里需要已经获取到临时密钥的结果。
// 临时密钥的生成参考 https://intl.cloud.tencent.com/document/product/436/14048
String tmpSecretId = "TMPSECRETID";
String tmpSecretKey = "TMPSECRETKEY";
String sessionToken = "SESSIONTOKEN";
COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sess
ionToken);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
```

```
return new COSClient(cred, clientConfig);
}
```

检查对象是否存在

通过调用查询对象元数据接口，来检查一个对象是否存在。

方法原型

```
public boolean doesObjectExist(String bucketName, String key)
throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
try {
boolean objectExists = cosClient.doesObjectExist(bucketName, key);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String

返回结果说明

- 成功：返回 boolean 类型， true 代表存在，false 代表不存在。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询对象元数据

最近更新时间：2022-01-27 13:00:24

简介

本文档提供关于查询对象元数据操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

简单操作

简单操作由 COSClient 类型发起请求，使用简单操作之前必须先创建一个 COSClient 实例。

COSClient 实例是并发安全的，这里推荐一个进程只创建一个 COSClient 实例，当不会再通过这个实例发起请求的时候，再选择关闭这个实例。

创建 COSClient

调用 COS 的接口之前，必须先创建一个 COSClient 的实例。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 设置用户身份信息。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.co
m/cam/capi 进行查看和管理
String secretId = "SECRETID";
String secretKey = "SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
```

```
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

使用临时密钥创建 COSClient

如果要使用临时密钥请求 COS，则需要用临时密钥创建 COSClient。

本 SDK 并不能生成临时密钥，而需要使用额外的操作来生成，参考 临时密钥生成。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 这里需要已经获取到临时密钥的结果。
// 临时密钥的生成参考 https://intl.cloud.tencent.com/document/product/436/14048
String tmpSecretId = "TMPSECRETID";
String tmpSecretKey = "TMPSECRETKEY";
String sessionToken = "SESSIONTOKEN";
COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sess
ionToken);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

查询对象元数据

查询指定对象的元数据信息（HEAD Object）。

方法原型

```
public ObjectMetadata getObjectMetadata(String bucketName, String key)
throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
try {
ObjectMetadata objectMetadata = cosClient.getObjectMetadata(bucketName, key);
System.out.println(objectMetadata.getCrc64Ecma());
System.out.println(objectMetadata.getLastModified());
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String

返回结果说明

- 成功：返回 ObjectMetadata 类型， 包含用户自定义头部、Etag等对象元信息。

版权所有：腾讯云计算（北京）有限责任公司

- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

返回参数说明

ObjectMetadata 类用于记录对象的元信息，其主要成员说明如下：

成员名称	描述	类型
httpExpiresDate	缓存的超时时间，为 HTTP 响应头部中 Expires 字段的值	Date
ongoingRestore	正在从归档存储类型恢复该对象	Boolean
userMetadata	前缀为 x-cos-meta- 的用户自定义元信息	Map<String, String>
metadata	除用户自定义元信息以外的其他头部	Map<String, String>
restoreExpirationTime	归档对象恢复副本的过期时间	Date

版权所有：腾讯云计算（北京）有限责任公司

修改对象元数据

最近更新时间：2022-06-30 14:33:34

简介

本文档提供关于修改对象元数据的 SDK 示例代码。

修改对象元数据利用了复制对象的接口，在复制过程中设置新的元数据。

API	操作名	操作描述
PUT Object - Copy	对象复制	复制文件到目标路径

简单操作

简单操作由 COSClient 类型发起请求，使用简单操作之前必须先创建一个 COSClient 实例。

COSClient 实例是并发安全的，这里推荐一个进程只创建一个 COSClient 实例，当不会再通过这个实例发起请求的时候，再选择关闭这个实例。

创建 COSClient

调用 COS 的接口之前，必须先创建一个 COSClient 的实例。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 设置用户身份信息。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.co
m/cam/capi 进行查看和管理
String secretId = "SECRETID";
String secretKey = "SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
```

```
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

使用临时密钥创建 COSClient

如果要使用临时密钥请求 COS，则需要用临时密钥创建 COSClient。

本 SDK 并不能生成临时密钥，而需要使用额外的操作来生成，参考 临时密钥生成。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 这里需要已经获取到临时密钥的结果。
// 临时密钥的生成参考 https://intl.cloud.tencent.com/document/product/436/14048#cos-sts-sdk
String tmpSecretId = "TMPSECRETID";
String tmpSecretKey = "TMPSECRETKEY";
String sessionToken = "SESSIONTOKEN";
COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sessionToken);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
```

```
return new COSClient(cred, clientConfig);
}
```

修改对象元数据

使用复制对象接口，在复制过程中设置新的元数据。在复制接口中仅仅修改元数据，不会执行对象数据的复制。

方法原型

```
public CopyObjectResult copyObject(CopyObjectRequest copyObjectRequest)
throws CosClientException, CosServiceException
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单上传 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 桶所在的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
Region region = new Region("ap-beijing");
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
// 获取当前的对象元数据
ObjectMetadata objectMetadata = cosclient.getObjectMetadata(bucketName, key);
// 修改对象元数据必须设置 replaced
objectMetadata.setHeader("x-cos-metadata-directive", "Replaced");
// 设置新的对象元数据
// 注意：Content-Disposition 、自定义元数据或者其他有中文的头域值，在设置前请先调用 UrlEnco
derUtils.encode(String) 编码，避免签名问题
objectMetadata.setHeader("x-cos-storage-class", "STANDARD_IA");
objectMetadata.setContentType("text/plain");
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(region, bucketName, k
ey, bucketName, key);
copyObjectRequest.setNewObjectMetadata(objectMetadata);
try {
CopyObjectResult copyObjectResult = cosclient.copyObject(copyObjectRequest);
System.out.print(copyObjectResult.getRequestId());
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
copyObjectRequest	拷贝文件请求	CopyObjectRequest

Request 成员说明：

参数名称	描述	类型
sourceBucketRegion	源 Bucket region。默认值：与当前 clientConfig 的 region 一致，表示同地域拷贝	String
sourceBucketName	源存储桶名称，命名格式为 BucketName-APPID，详情请参见 命名规范	String
sourceKey	源对象键，对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
sourceVersionId	源文件 version id（适用于开启了版本控制的源 Bucket）。默认值：源文件当前最新版本	String
destinationBucketName	目标存储桶名称，Bucket 的命名格式为 BucketName-APPID ，name 由字母数字和中划线构成	String
destinationKey	目的对象键，对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
storageClass	拷贝的目的文件的存储类型。枚举值：Standard，Standard_IA。默认值：Standard。更多存储类型请参见 存储类型概述	String

返回结果说明

- 成功：返回 CopyObjectResult，包含新文件的 Etag 等信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

版权所有：腾讯云计算（北京）有限责任公司

对象访问 URL

最近更新时间：2022-01-27 13:00:24

简介

本文档提供关于生成对象访问 URL 的示例代码。
这里生成的 URL 显示了对象存在 COS 的路径。

> 说明：
> 如果您的文件是**私有读**权限，那么本接口生成的 URL 不能直接用于访问资源。

获取对象访问 URL

方法原型

```
public URL getObjectUrl(String bucketName, String key);
```

请求示例

```
// 不需要验证身份信息
COSCredentials cred = new AnonymousCOSCredentials();
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置生成的 url 的请求协议, http 或者 https
// 5.6.53 及更低的版本, 建议设置使用 https 协议
// 5.6.54 及更高版本, 默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// 存储桶的命名格式为 BucketName-APPID, 此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
System.out.println(cosclient.getObjectUrl(bucketName, key));
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识，详情请参见 对象键	String	是

版权所有：腾讯云计算（北京）有限责任公司

生成预签名 URL

最近更新时间：2023-06-14 14:31:29

简介

本文档提供关于生成对象预签名链接的示例代码。

关于使用预签名 url 上传的说明请参见 预签名授权上传， 使用预签名 url 下载的说明请参见 预签名授权下载。

说明：

- 建议用户 使用临时密钥 生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄露目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。
- 获取签名/预签名函数，默认签入 Header Host；您也可以选择不签入 Header Host，但可能导致请求失败或安全漏洞。

简单操作

简单操作由 COSClient 类型发起请求，使用简单操作之前必须先创建一个 COSClient 实例。

COSClient 实例是并发安全的，这里推荐一个进程只创建一个 COSClient 实例，当不会再通过这个实例发起请求的时候，再选择关闭这个实例。

使用临时密钥创建 COSClient（推荐）

如果要使用临时密钥请求 COS，则需要用临时密钥创建 COSClient。
本 SDK 并不能生成临时密钥，而需要使用额外的操作来生成，详情请参见 临时密钥生成。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 这里需要已经获取到临时密钥的结果。
// 临时密钥的生成参见 https://intl.cloud.tencent.com/document/product/436/14048#cos-sts-sdk
String tmpSecretId = "TMPSECRETID";
String tmpSecretKey = "TMPSECRETKEY";
```

```
String sessionToken = "SESSIONTOKEN";

COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sess
ionToken);

// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();

// 设置 bucket 的地域
// COS_REGION 请参见 https://cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));

// 设置请求协议, http 或者 https
// 5.6.53 及更低的版本, 建议设置使用 https 协议
// 5.6.54 及更高版本, 默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);

// 以下的设置, 是可选的：

// 设置 socket 读取超时, 默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时, 默认 30s
clientConfig.setConnectionTimeout(30*1000);

// 如果需要的话, 设置 http 代理, ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);

// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

使用永久密钥创建 COSClient（不推荐）

调用 COS 的接口之前，必须先创建一个 COSClient 的实例。

```
// 创建 COSClient 实例, 这个实例用来后续调用请求
COSClient createCOSClient() {
// 设置用户身份信息。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.co
m/cam/capi 进行查看和管理
String secretId = System.getenv("secretId");//用户的 SecretId, 建议使用子账号密钥, 授
权遵循最小权限指引, 降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/
product/598/37140
String secretKey = System.getenv("secretKey");//用户的 SecretKey, 建议使用子账号密
钥, 授权遵循最小权限指引, 降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/docu
```

```
ment/product/598/37140
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);

// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();

// 设置 bucket 的地域
// COS_REGION 请参见 https://cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));

// 设置请求协议, http 或者 https
// 5.6.53 及更低的版本, 建议设置使用 https 协议
// 5.6.54 及更高版本, 默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);

// 以下的设置, 是可选的：

// 设置 socket 读取超时, 默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时, 默认 30s
clientConfig.setConnectionTimeout(30*1000);

// 如果需要的话, 设置 http 代理, ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);

// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

生成预签名 URL

方法原型

```
public URL generatePresignedUrl(String bucketName, String key, Date expiration, H
ttpMethodName method, Map<String, String> headers, Map<String, String> params, Bo
olean signPrefixMode, Boolean signHost) throws CosClientException
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例, 如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
```

```
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.com/document/product/436/13324)
String key = "exampleobject";

// 设置签名过期时间(可选)，若未进行设置则默认使用 ClientConfig 中的签名过期时间(1小时)
// 这里设置签名在半个小时后过期
Date expirationDate = new Date(System.currentTimeMillis() + 30 * 60 * 1000);

// 填写本次请求的参数，需与实际请求相同，能够防止用户篡改此签名的 HTTP 请求的参数
Map<String, String> params = new HashMap<String, String>();
params.put("param1", "value1");

// 填写本次请求的头部，需与实际请求相同，能够防止用户篡改此签名的 HTTP 请求的头部
Map<String, String> headers = new HashMap<String, String>();
headers.put("header1", "value1");

// 请求的 HTTP 方法，上传请求用 PUT，下载请求用 GET，删除请求用 DELETE
HttpMethodName method = HttpMethodName.GET;

URL url = cosClient.generatePresignedUrl(bucketName, key, expirationDate, method, headers, params);
System.out.println(url.toString());

// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型	是否必填
method	HTTP 方法，可选：GET、POST、PUT、DELETE、HEAD	HttpMethodName	是
bucketName	存储桶名称，存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String	是
key	对象键（Key）是对象在存储桶中的唯一标识，详情请参见 对象键（**注意：用户无需对key进行编码操作**）	String	是
expiration	签名过期的时间，可以设置任意一个未来的时间，不设置则默认是1小时之后过期	Date	否
headers	签名头部	Map<String, String>	否

参数名称	描述	类型	是否必填
params	签名参数	Map<String, String>	否
signPrefixMode	是否以 sign 参数指定签名（不推荐），默认 false	boolean	否
signHost	是否签入 Host 头部（推荐），默认 true	boolean	否

生成覆盖返回头部的预签名下载 URL

方法原型

```
public URL generatePresignedUrl(GeneratePresignedUrlRequest req, boolean signHost) throws CosClientException
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();

// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.com/document/product/436/13324)
String key = "exampleobject";

GeneratePresignedUrlRequest req =
new GeneratePresignedUrlRequest(bucketName, key, HttpMethodName.GET);

// 设置下载时返回的 http 头
ResponseHeaderOverrides responseHeaders = new ResponseHeaderOverrides();
String responseContentType = "image/x-icon";
String responseContentLanguage = "zh-CN";
// 设置返回头部里包含文件名信息
String responseContentDispositon = "filename=exampleobject";
String responseCacheControl = "no-cache";
String cacheExpireStr =
DateUtils.formatRFC822Date(new Date(System.currentTimeMillis() + 24L * 3600L * 1000L));
responseHeaders.setContentType(responseContentType);
responseHeaders.setContentLanguage(responseContentLanguage);
responseHeaders.setContentDisposition(responseContentDispositon);
```

```
responseHeaders.setCacheControl(responseCacheControl);
responseHeaders.setExpires(cacheExpireStr);

req.setResponseHeaders(responseHeaders);

// 设置签名过期时间(可选)，若未进行设置，则默认使用 ClientConfig 中的签名过期时间(1小时)
// 这里设置签名在半个小时后过期
Date expirationDate = new Date(System.currentTimeMillis() + 30L * 60L * 1000L);
req.setExpiration(expirationDate);

// 填写本次请求的参数
req.addRequestParameter("param1", "value1");
// 填写本次请求的头部
// host 必填
req.putCustomRequestHeader(Headers.HOST, cosClient.getClientConfig().getEndpointB
uilder().buildGeneralApiEndpoint(bucketName));
req.putCustomRequestHeader("header1", "value1");

URL url = cosClient.generatePresignedUrl(req);
System.out.println(url.toString());

// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型	是否必填
req	预签名请求类	GeneratePresignedUrlRequest	是
signHost	是否签入 Host 头部（推荐）默认 true	boolean	否

Request 成员说明：

Request 成员	设置方法	描述	类型
method	构造函数或 set 方法	HTTP 方法，可选：GET、POST、PUT、DELETE、HEAD	HttpMethodName
bucketName	构造函数或 set 方法	存储桶名称，存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String

Request 成员	设置方法	描述	类型
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识，详情请参见 对象键（**注意：用户无需对key进行编码操作**）	String
expiration	set 方法	签名过期的时间，可以设置任意一个未来的时间，不设置则默认是1小时之后过期	Date
contentType	set 方法	要签名的请求中的 Content-Type	String
contentMd5	set 方法	要签名的请求中的 Content-Md5	String
responseHeaders	set 方法	签名的下载请求中要覆盖的返回的 HTTP 头	ResponseHeaderOverrides
versionId	set 方法	在存储桶开启版本控制的时候，指定对象的版本号	String

生成签名

构造 COS 请求的签名。推荐使用临时密钥来生成签名。

使用临时密钥（推荐）

```
// 这里需要已经获取到临时密钥的结果。
// 临时密钥的生成参见 https://intl.cloud.tencent.com/document/product/436/14048#cos-sts-sdk
String tmpSecretId = "TMPSECRETID";
String tmpSecretKey = "TMPSECRETKEY";
String sessionToken = "SESSIONTOKEN";

COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sessionToken);

// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.com/document/product/436/13324)
String key = "exampleobject";
//若key不是以“/”开头，则需要在 key 的开头加上“/”，否则直接 resource_path=key
String resource_path="/" + key;

ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));

// 用来生成签名
COSSigner signer = new COSSigner();
// 设置签名过期时间(可选)，若未进行设置，则默认使用 ClientConfig 中的签名过期时间(1小时)
```

```
// 这里设置签名在半个小时后过期
Date expirationDate = new Date(System.currentTimeMillis() + 30L * 60L * 1000L);

// 填写本次请求的参数
Map<String, String> params = new HashMap<String, String>();
params.put("param1", "value1");

// 填写本次请求的头部
Map<String, String> headers = new HashMap<String, String>();
// host 必填
headers.put(Headers.HOST, clientConfig.getEndpointBuilder().buildGeneralApiEndpoi
nt(bucketName));
headers.put("header1", "value1");

// 请求的 HTTP 方法，上传请求用 PUT，下载请求用 GET，删除请求用 DELETE
HttpMethodName method = HttpMethodName.GET;

String sign = signer.buildAuthorizationStr(method, resource_path, headers, param
s, cred, expirationDate, true);
```

使用永久密钥

```
// 设置用户身份信息。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.co
m/cam/capi 进行查看和管理
String secretId = System.getenv("secretId");//用户的 SecretId，建议使用子账号密钥，授
权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/
product/598/37140
String secretKey = System.getenv("secretKey");//用户的 SecretKey，建议使用子账号密
钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/docu
ment/product/598/37140
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);

// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
//若 key不是以“/”开头，则需要在 key 的开头加上“/”，否则直接 resource_path=key
String resource_path="/" + key;

ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));

// 用来生成签名
COSSigner signer = new COSSigner();
// 设置签名过期时间(可选)，若未进行设置，则默认使用 ClientConfig 中的签名过期时间(1小时)
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 这里设置签名在半个小时后过期
Date expirationDate = new Date(System.currentTimeMillis() + 30L * 60L * 1000L);

// 填写本次请求的参数
Map<String, String> params = new HashMap<String, String>();
params.put("param1", "value1");

// 填写本次请求的头部
Map<String, String> headers = new HashMap<String, String>();
// host 必填
headers.put(Headers.HOST, clientConfig.getEndpointBuilder().buildGeneralApiEndpoi
nt(bucketName));
headers.put("header1", "value1");

// 请求的 HTTP 方法，上传请求用 PUT，下载请求用 GET，删除请求用 DELETE
HttpMethodName method = HttpMethodName.GET;

String sign = signer.buildAuthorizationStr(method, resource_path, headers, param
s, cred, expirationDate, true);
```

生成限速的预签名下载 URL

关于限速的使用说明，可参见 单链接限速。

请求示例

以生成限速的预签名下载 URL 为例：

```
public static void GenerateSimplePresignedDownloadUrl() {
// 1 初始化用户身份信息(secretId, secretKey)
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.co
m/cam/capi 进行查看和管理
String secretId = System.getenv("secretId");//用户的 SecretId，建议使用子账号密钥，授
权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/
product/598/37140
String secretKey = System.getenv("secretKey");//用户的 SecretKey，建议使用子账号密
钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/docu
ment/product/598/37140
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 2 设置 bucket的区域
// COS_REGION 参数：配置成存储桶 bucket 的实际地域，例如 ap-beijing，更多 COS 地域的简称
请参见 https://cloud.tencent.com/document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("COS_REGION"));
// 3 生成 cos 客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket 名需包含 appid
String bucketName = "examplebucket-1250000000";
```

```
String key = "exampleobject";
GeneratePresignedUrlRequest req =
new GeneratePresignedUrlRequest(bucketName, key, HttpMethodName.GET);
// 设置签名过期时间(可选), 若未进行设置则默认使用 ClientConfig 中的签名过期时间(1小时)
// 这里设置签名在半个小时后过期
Date expirationDate = new Date(System.currentTimeMillis() + 30 * 60 * 1000);
req.setExpiration(expirationDate);

// 填写本次请求的参数
// 设定限速值, 例如128KB/s
req.addRequestParameter("x-cos-traffic-limit", "1048576");

// 填写本次请求的头部。Host 必填
req.putCustomRequestHeader(Headers.HOST,
cosclient.getClientConfig().getEndpointBuilder().buildGeneralApiEndpoint(bucketNa
me));
//req.putCustomRequestHeader("header1", "value1");

URL url = cosclient.generatePresignedUrl(req);
System.out.println(url.toString());

cosclient.shutdown();
}
```

恢复归档对象

最近更新时间：2022-01-27 13:00:24

简介

本文档提供关于恢复归档对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
POST Object restore	恢复归档对象	将归档类型的对象取回访问

简单操作

简单操作由 COSClient 类型发起请求，使用简单操作之前必须先创建一个 COSClient 实例。

COSClient 实例是并发安全的，这里推荐一个进程只创建一个 COSClient 实例，当不会再通过这个实例发起请求的时候，再选择关闭这个实例。

创建 COSClient

调用 COS 的接口之前，必须先创建一个 COSClient 的实例。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 设置用户身份信息。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.co
m/cam/capi 进行查看和管理
String secretId = "SECRETID";
String secretKey = "SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
```

```
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

使用临时密钥创建 COSClient

如果要使用临时密钥请求 COS，则需要用临时密钥创建 COSClient。

本 SDK 并不能生成临时密钥，而需要使用额外的操作来生成，参考 临时密钥生成。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 这里需要已经获取到临时密钥的结果。
// 临时密钥的生成参考 https://intl.cloud.tencent.com/document/product/436/14048
String tmpSecretId = "TMPSECRETID";
String tmpSecretKey = "TMPSECRETKEY";
String sessionToken = "SESSIONTOKEN";
COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sess
ionToken);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

恢复归档对象

将归档类型的对象取回访问（POST Object restore）。

方法原型

```
public void restoreObject(RestoreObjectRequest restoreObjectRequest)
throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
// 设置 restore 得到的临时副本过期天数为1天
RestoreObjectRequest restoreObjectRequest = new RestoreObjectRequest(bucketName,
key, 1);
// 设置恢复模式为 Standard，其他的可选模式包括 Expedited 和 Bulk。若恢复归档存储类型数据，
则支持上述三种恢复模式，选择不同恢复模式，在费用和恢复速度上不一样。若恢复深度归档存储类型数据，
则仅支持 Standard 和 Bulk 恢复模式
CASJobParameters casJobParameters = new CASJobParameters();
casJobParameters.setTier(Tier.Standard);
restoreObjectRequest.setCASJobParameters(casJobParameters);
try {
cosClient.restoreObject(restoreObjectRequest);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	描述	类型
restoreObjectRequest	请求类	RestoreObjectRequest

Request 成员说明：

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
expirationInDays	恢复出的临时文件的过期天数	int
casJobParameters	描述恢复类型的配置信息。若恢复归档存储类型数据，可调用 setTier 函数设置为 Tier.Standard、Tier.Expedited、Tier.Bulk 三种恢复类型之一；若恢复深度归档存储类型，则仅支持 Tier.Standard 和 Tier.Bulk	CASJobParameters

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败）， 抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

服务端加密

最近更新时间：2022-03-04 09:50:47

简介

本文档提供关于如何使用在上传对象时开启服务端加密。服务端加密的密钥分为三种：

- COS 托管加密密钥
- 客户提供的加密密钥
- KMS 托管加密密钥

使用 COS 托管加密密钥的服务端加密（SSE-COS）保护数据

由腾讯云 COS 托管主密钥和管理数据。COS 会帮助您在数据写入数据中心时自动加密，并在您取用该数据时自动解密。目前支持使用 COS 主密钥对数据进行 AES-256 加密。

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见上传对象：高级接口 -> 示例代码：创建 TransferManager
TransferManager transferManager = createTransferManager();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
// 本地文件路径
String localFilePath = "/path/to/localFile";
File localFile = new File(localFilePath);
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFi
le);
ObjectMetadata objectMetadata = new ObjectMetadata();
// 设置加密算法为AES256
objectMetadata.setServerSideEncryption(SSEAlgorithm.AES256.getAlgorithm());
putObjectRequest.setMetadata(objectMetadata);
// 服务端加密场景下，返回的etag不再代表文件的md5，所以需要去掉客户端的md5校验
// 如有需要，可获取crc64，自行校验
System.setProperty(SkipMd5CheckStrategy.DISABLE_PUT_OBJECT_MD5_VALIDATION_PROPERT
Y, "true");
try {
// 高级接口会返回一个异步结果Upload
// 可同步地调用 waitForUploadResult 方法等待上传完成，成功返回UploadResult，失败抛出异常
Upload upload = transferManager.upload(putObjectRequest);
UploadResult uploadResult = upload.waitForUploadResult();
} catch (CosServiceException e) {
```

```
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

使用客户提供的加密密钥的服务端加密 （SSE-C）保护数据

加密密钥由用户自己提供，用户在上传对象时，COS 将使用用户提供的加密密钥对用户的数据进行 AES-256 加密。

注意：

- 该加密所运行的服务需要使用 HTTPS 请求。
- 用户需要提供一个32字节的字符串作为密钥，支持数字、字母、字符的组合，不支持中文。
- 如果上传文件时设置了密钥加密，那么在使用 GET（下载）、HEAD（查询）源对象时也需要在请求中带上相同的密钥，才能正常响应。

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见上传对象：高级接口 -> 示例代码：创建 TransferManager
TransferManager transferManager = createTransferManager();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
// 本地文件路径
String localFilePath = "/path/to/localFile";
File localFile = new File(localFilePath);
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFi
le);
String base64EncodedKey = "MDEyMzQ1Njc4OUFCQ0RFRjAxMjM0NTY3ODlBQkNERUY=";
// sseCustomerKey是base64编码的密钥
SSECustomerKey sseCustomerKey = new SSECustomerKey(base64EncodedKey);
putObjectRequest.setSSECustomerKey(sseCustomerKey);
// 服务端加密场景下，返回的etag不再代表文件的md5，所以需要去掉客户端的md5校验
// 如有需要，可获取crc64，自行校验
System.setProperty(SkipMd5CheckStrategy.DISABLE_PUT_OBJECT_MD5_VALIDATION_PROPERT
Y, "true");
try {
```

```
// 高级接口会返回一个异步结果Upload
// 可同步地调用 waitForUploadResult 方法等待上传完成，成功返回UploadResult，失败抛出异常
Upload upload = transferManager.upload(putObjectRequest);
UploadResult uploadResult = upload.waitForUploadResult();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

使用 KMS 托管加密密钥的服务端加密（SSE-KMS）保护数据

SSE-KMS 加密即使用 KMS 托管密钥的服务端加密。KMS 是腾讯云推出的一款安全管理类服务，使用经过第三方认证的硬件安全模块 HSM（Hardware Security Module）来生成和保护密钥。它能够帮助用户轻松创建和管理密钥，满足用户多应用多业务的密钥管理需求以及满足监管和合规要求。关于如何开通 KMS 服务请参考：服务端加密概述。

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见上传对象：高级接口 -> 示例代码：创建 TransferManager
TransferManager transferManager = createTransferManager();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
// 本地文件路径
String localFilePath = "/path/to/localFile";
File localFile = new File(localFilePath);
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFi
le);
// KMS 的主密钥
String kmsKeyId = "your-kms-key-id";
String encryptionContext = Base64.encodeAsString("{\"Ssekmstest\":\"Ssekmstest
\"}".getBytes());
SSECOSKeyManagementParams ssecosKeyManagementParams = new SSECOSKeyManagementPara
ms(kmsKeyId, encryptionContext);
putObjectRequest.setSSECOSKeyManagementParams(ssecosKeyManagementParams);
// 服务端加密场景下，返回的etag不再代表文件的md5，所以需要去掉客户端的md5校验
// 如有需要，可获取crc64，自行校验
System.setProperty(SkipMd5CheckStrategy.DISABLE_PUT_OBJECT_MD5_VALIDATION_PROPERT
```

```
Y, "true");
try {
// 高级接口会返回一个异步结果Upload
// 可同步地调用 waitForUploadResult 方法等待上传完成，成功返回UploadResult，失败抛出异常
Upload upload = transferManager.upload(putObjectRequest);
UploadResult uploadResult = upload.waitForUploadResult();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

客户端加密

最近更新时间：2023-06-16 15:09:03

简介

Java SDK 支持客户端加密，将文件加密后再进行上传，并在下载时进行解密，适用于存储敏感数据的客户。

客户端加密支持以下两种方式：

- KMS 服务托管密钥：用户只需提供 KMS 服务的用户主密钥 ID（即 CMK ID）给 SDK。使用这种方式需要用户开通 KMS 服务，更多 KMS 服务信息参见 腾讯云密钥管理系统。
- 用户自主保管密钥：用户提供和保管加密密钥，支持对称 AES 和非对称 RSA 加密。

> 说明：
> 这里的对称和非对称只用于加密每次生成的随机密钥，对文件数据的加密始终使用 AES256 对称加密。

注意事项

- 用户数据在加密前不会上传到腾讯云对象存储（Cloud Object Storage，COS）。
- 客户端加密会消耗部分上传速度。
- 针对 SDK 内部的分块上传，将使用串行的方式进行上传。

准备事项

客户端加密内部使用 AES256 来对数据进行加密，默认 JDK6 - JDK8 早期的版本不支持256位加密。如果运行时会报出以下异常 `java.security.InvalidKeyException: Illegal key size or default parameters`，那么我们需要补充 Oracle 的 JCE 无政策限制权限文件，将其部署在 JRE 的环境中。 请根据目前使用的 JDK 版本，分别下载对应的文件，将其解压后保存在 JAVA_HOME 下的 `jre/lib/security` 目录下。

- JDK6 JCE 补充包
- JDK7 JCE 补充包
- JDK8 JCE 补充包

上传加密流程

版权所有：腾讯云计算（北京）有限责任公司

1. 每次上传一个文件对象前，将随机生成一个对称加密密钥。随机生成的密钥通过 KMS 服务（或者用户提供的密钥）进行加密，将加密后的结果 base64 编码存储在对象的元数据中。
2. 进行文件对象的上传，上传时在内存使用 AES256 算法加密。

下载解密流程

1. 获取文件元数据中加密必要的信息，Base64 解码后使用 KMS 服务（或者用户提供的密钥）进行解密，得到当时加密数据的密钥。
2. 使用密钥对下载输入流进行使用 AES256 解密，得到解密后的文件输入流。

请求示例

示例1

使用腾讯云 KMS 服务加密，创建加密客户端示例，完整的示例代码请参见 KMS 加密客户端加密完整示例。

```
// 初始化用户身份信息(secretId, secretKey)
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
String secretId = System.getenv("secretId");//用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
String secretKey = System.getenv("secretKey");//用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 设置存储桶地域，COS 地域的简称请参照 https://www..com/document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("COS_REGION"));
// 为防止请求头部被篡改导致的数据无法解密，强烈建议只使用 https 协议发起请求。
clientConfig.setHttpProtocol(HttpProtocol.https);

// 用户 KMS 服务的主密钥
String cmk = "XXXXXXX";
//// 如果 KMS 服务的地域 与 COS 的存储桶地域不一致则需要单独设置。
//String kmsRegion = "XXXXX";

// 初始化 KMS 加密材料
KMSEncryptionMaterials encryptionMaterials = new KMSEncryptionMaterials(cmk);
// 使用 AES/GCM 模式，并将加密信息存储在文件元信息中.
CryptoConfiguration cryptoConf = new CryptoConfiguration(CryptoMode.AuthenticatedEncryption)
.withStorageMode(CryptoStorageMode.ObjectMetadata);
```

版权所有：腾讯云计算（北京）有限责任公司

```
//// 如果 KMS 服务的地域 与 COS 的地域不一致，则在加密配置里指定 KMS 服务的地域
//cryptoConf.setKmsRegion(kmsRegion);

//// 如果需要可以为 KMS 服务的 cmk 设置对应的描述信息。
// encryptionMaterials.addDescription("yourDescKey", "yourDescValue");

// 生成加密客户端 EncryptionClient，COSEncryptionClient 是 COSClient 的子类，所有 COS Client 支持的接口他都支持。
// EncryptionClient 覆盖了 COSClient 上传下载逻辑，操作内部会执行加密操作，其他操作执行逻辑和 COSClient 一致
COSEncryptionClient cosEncryptionClient =
new COSEncryptionClient(new COSStaticCredentialsProvider(cred),
new KMSEncryptionMaterialsProvider(encryptionMaterials), clientConfig,
cryptoConf);

// 上传文件
// 这里给出 putObject 的示例，对于高级 API 上传，只用在生成 TransferManager 时传入 COSEncryptionClient 对象即可
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
File localFile = new File("localFilePath");
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
cosEncryptionClient.putObject(putObjectRequest);
cosEncryptionClient.shutdown();
```

示例2

使用对称 AES256 加密每次生成的随机密钥示例，完整的示例代码请参见 客户端对称密钥加密完整示例。

```
// 初始化用户身份信息(secretId, secretKey)
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
String secretId = System.getenv("secretId");//用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
String secretKey = System.getenv("secretKey");//用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 设置存储桶地域，COS 地域的简称请参见 https://www..com/document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("COS_REGION"));

// 生成对称密钥，您可以将其保存在文件中
KeyGenerator symKeyGenerator = KeyGenerator.getInstance("AES");
```

```
symKeyGenerator.init(256);
SecretKey symKey = symKeyGenerator.generateKey();

EncryptionMaterials encryptionMaterials = new EncryptionMaterials(symKey);
// 使用 AES/GCM 模式，并将加密信息存储在文件元数据中
CryptoConfiguration cryptoConf = new CryptoConfiguration(CryptoMode.AuthenticatedEncryption)
.withStorageMode(CryptoStorageMode.ObjectMetadata);

// 生成加密客户端 EncryptionClient，COSEncryptionClient 是 COSClient 的子类，所有 COSClient 支持的接口他都支持。
// EncryptionClient 覆盖了 COSClient 上传下载逻辑，操作内部会执行加密操作，其他操作执行逻辑和 COSClient 一致
COSEncryptionClient cosEncryptionClient =
new COSEncryptionClient(new COSStaticCredentialsProvider(cred),
new StaticEncryptionMaterialsProvider(encryptionMaterials), clientConfig,
cryptoConf);

// 上传文件
// 这里给出 putObject 的示例，对于高级 API 上传，只用在生成 TransferManager 时传入 COSEncryptionClient 对象即可
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
File localFile = new File(localFilePath);
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
cosEncryptionClient.putObject(putObjectRequest);
cosEncryptionClient.shutdown();
```

示例3

使用非对称 RSA 加密每次生成的随机密钥示例，完整的示例代码请参见 客户端非对称密钥加密完整示例。

```
// 初始化用户身份信息(secretId, secretKey)
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
String secretId = System.getenv("secretId");//用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
String secretKey = System.getenv("secretKey");//用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 设置存储桶地域，COS 地域的简称请参见 https://intl.cloud.tencent.com/document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("COS_REGION"));
```

```
// 生成非对称密钥
KeyPairGenerator keyGenerator = KeyPairGenerator.getInstance("RSA");
SecureRandom srand = new SecureRandom();
keyGenerator.initialize(1024, srand);
KeyPair asymKeyPair = keyGenerator.generateKeyPair();

EncryptionMaterials encryptionMaterials = new EncryptionMaterials(asymKeyPair);
// 使用 AES/GCM 模式，并将加密信息存储在文件元数据中
CryptoConfiguration cryptoConf = new CryptoConfiguration(CryptoMode.AuthenticatedEncryption)
.withStorageMode(CryptoStorageMode.ObjectMetadata);

// 生成加密客户端 EncryptionClient，COSEncryptionClient 是 COSClient 的子类，所有COSClient 支持的接口他都支持。
// EncryptionClient 覆盖了 COSClient 上传下载逻辑，操作内部会执行加密操作，其他操作执行逻辑和 COSClient 一致
COSEncryptionClient cosEncryptionClient =
new COSEncryptionClient(new COSStaticCredentialsProvider(cred),
new StaticEncryptionMaterialsProvider(encryptionMaterials), clientConfig,
cryptoConf);

// 上传文件
// 这里给出 putObject 的示例，对于高级 API 上传，只用在生成 TransferManager 时传入 COSEncryptionClient 对象即可
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
File localFile = new File(localFilePath);
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
cosEncryptionClient.putObject(putObjectRequest);
cosEncryptionClient.shutdown();
```

单链接限速

最近更新时间：2022-03-04 11:11:49

简介

本文档提供关于单链接限速相关的介绍和 SDK 示例代码。

> 说明：
> 限速值设置范围为819200 - 838860800，单位默认为 bit/s，即100KB/s - 100MB/s，如果超出该范围将返回400错误。

设置上传限速

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 创建 TransferManager
TransferManager transferManager = createTransferManager();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.com/document/product/436/13324)
String key = "exampleobject";
// 这里创建一个 ByteArrayInputStream 来作为示例，实际中这里应该是您要上传的 InputStream 类型的流
long inputStreamLength = 1024 * 1024;
byte data[] = new byte[inputStreamLength];
InputStream inputStream = new ByteArrayInputStream(data);
// 限速使用的单位是 bit/s，这里设置上传限制为 10MB/s
putObjectRequest.setTrafficLimit(80*1024*1024);
try {
// 高级接口会返回一个异步结果Upload
// 可同步地调用 waitForUploadResult 方法等待上传完成，成功返回UploadResult，失败抛出异常
Upload upload = transferManager.upload(putObjectRequest);
UploadResult uploadResult = upload.waitForUploadResult();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
```

```
// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

设置下载限速

```
// 使用高级接口必须先保证本进程存在一个 TransferManager 实例，如果没有则创建
// 详细代码参见本页：高级接口 -> 创建 TransferManager
TransferManager transferManager = createTransferManager();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.com/document/product/436/13324)
String key = "exampleobject";
// 本地文件路径
String localFilePath = "/path/to/localFile";
File downloadFile = new File(localFilePath);
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
// 限速使用的单位是 bit/s，这里设置下载限制为 10MB/s
getObjectRequest.setTrafficLimit(80*1024*1024);
try {
// 返回一个异步结果 Donload，可同步的调用 waitForCompletion 等待下载结束，成功返回 void，失败抛出异常
Download download = transferManager.download(getObjectRequest, downloadFile);
download.waitForCompletion();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
// 确定本进程不再使用 transferManager 实例之后，关闭之
// 详细代码参见本页：高级接口 -> 关闭 TransferManager
shutdownTransferManager(transferManager);
```

版权所有：腾讯云计算（北京）有限责任公司

检索对象内容

最近更新时间：2022-01-27 13:10:21

简介

本文档提供关于检索对象内容操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
SELECT Object Content	检索对象内容	从指定对象（CSV 格式或者 JSON 格式）中检索内容

简单操作

简单操作由 COSClient 类型发起请求，使用简单操作之前必须先创建一个 COSClient 实例。

COSClient 实例是并发安全的，这里推荐一个进程只创建一个 COSClient 实例，当不会再通过这个实例发起请求的时候，再选择关闭这个实例。

创建 COSClient

调用 COS 的接口之前，必须先创建一个 COSClient 的实例。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 设置用户身份信息。
// SECRETID 和 SECRETKEY 请登录访问管理控制台 https://console.intl.cloud.tencent.co
m/cam/capi 进行查看和管理
String secretId = "SECRETID";
String secretKey = "SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
```

```
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

使用临时密钥创建 COSClient

如果要使用临时密钥请求 COS，则需要用临时密钥创建 COSClient。

本 SDK 并不能生成临时密钥，而需要使用额外的操作来生成，参考 临时密钥生成。

```
// 创建 COSClient 实例，这个实例用来后续调用请求
COSClient createCOSClient() {
// 这里需要已经获取到临时密钥的结果。
// 临时密钥的生成参考 https://intl.cloud.tencent.com/document/product/436/14048
String tmpSecretId = "TMPSECRETID";
String tmpSecretKey = "TMPSECRETKEY";
String sessionToken = "SESSIONTOKEN";
COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sess
ionToken);
// ClientConfig 中包含了后续请求 COS 的客户端设置：
ClientConfig clientConfig = new ClientConfig();
// 设置 bucket 的地域
// COS_REGION 请参照 https://intl.cloud.tencent.com/document/product/436/6224
clientConfig.setRegion(new Region("COS_REGION"));
// 设置请求协议，http 或者 https
// 5.6.53 及更低的版本，建议设置使用 https 协议
// 5.6.54 及更高版本，默认使用了 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 以下的设置，是可选的：
// 设置 socket 读取超时，默认 30s
clientConfig.setSocketTimeout(30*1000);
// 设置建立连接超时，默认 30s
clientConfig.setConnectionTimeout(30*1000);
// 如果需要的话，设置 http 代理，ip 以及 port
clientConfig.setHttpProxyIp("httpProxyIp");
clientConfig.setHttpProxyPort(80);
// 生成 cos 客户端。
return new COSClient(cred, clientConfig);
}
```

检索对象内容

COS Select 支持检索以下格式的对象数据：

- CSV 格式：对象以 CSV 格式存储，并以固定的分隔符划分。
- JSON 格式：对象以 JSON 格式存储，可以是 JSON 文件或者 JSON 列表。

> 注意：

- 使用 COS Select，您必须具有 `cos:GetObject` 的授权。
- CSV、JSON 对象需要以 UTF-8 格式编码。
- COS Select 支持检索 GZIP 或者 BZIP2 压缩的 CSV、JSON 对象。
- COS Select 支持检索 SSE-COS 加密的 CSV、JSON 对象。

检索 CSV 格式

方法原型

```
public SelectObjectContentResult selectObjectContent(SelectObjectContentRequest s
electRequest) throws CosClientException, CosServiceException {
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
String query = "select s._1 from COSObject s";
SelectObjectContentRequest request = new SelectObjectContentRequest();
request.setBucketName(bucketName);
request.setKey(key);
request.setExpression(query);
request.setExpressionType(ExpressionType.SQL);
InputSerialization inputSerialization = new InputSerialization();
CSVInput csvInput = new CSVInput();
csvInput.setFieldDelimiter(",");
csvInput.setRecordDelimiter("\n");
```

```
inputSerialization.setCsv(csvInput);
inputSerialization.setCompressionType(CompressionType.NONE);
request.setInputSerialization(inputSerialization);
OutputSerialization outputSerialization = new OutputSerialization();
outputSerialization.setCsv(new CSVOutput());
request.setOutputSerialization(outputSerialization);
final AtomicBoolean isResultComplete = new AtomicBoolean(false);
SelectObjectContentResult result = cosclient.selectObjectContent(request);
InputStream resultInputStream = result.getPayload().getRecordsInputStream(
new SelectObjectContentEventVisitor() {
@Override
public void visit(SelectObjectContentEvent.StatsEvent event)
{
System.out.println(
"Received Stats, Bytes Scanned: " + event.getDetails().getBytesScanned()
+ " Bytes Processed: " + event.getDetails().getBytesProcessed());
}
@Override
public void visit(SelectObjectContentEvent.EndEvent event)
{
isResultComplete.set(true);
System.out.println("Received End Event. Result is complete.");
}
}
);
BufferedReader reader = new BufferedReader(new InputStreamReader(resultInputStrea
m));
StringBuffer stringBuffer = new StringBuffer();
String line;
while((line = reader.readLine())!= null){
stringBuffer.append(line).append("\n");
}
System.out.println(stringBuffer.toString());
// 检查结果是否接受完整
if (!isResultComplete.get()) {
throw new Exception("result was incomplete");
}
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	参数描述	类型
selectRequest	请求	SelectObjectContentRequest

SelectObjectContentRequest 成员说明：

参数名称	设置方法	描述	类型
bucketName	set 方法	Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	set 方法	指定分块上传到 COS 上的路径，即 对象键。例如对象键为 folder/picture.jpg	String
expression	set 方法	请求表达式	String
expressionType	set 方法	请求表达式类型	String
inputSerialization	set 方法	描述待检索对象的格式	InputSerialization
outputSerialization	set 方法	描述检索结果的输出格式	OutputSerialization

返回结果说明

- 成功：返回 InputStream。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

检索 JSON 格式

方法原型

```
public SelectObjectContentResult selectObjectContent(SelectObjectContentRequest s
electRequest) throws CosClientException, CosServiceException {
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见本页：简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](https://intl.cloud.tencent.
com/document/product/436/13324)
String key = "exampleobject";
String query = "select * from COSObject s where mathScore > 85'";
SelectObjectContentRequest request = new SelectObjectContentRequest();
request.setBucketName(bucketName);
request.setKey(key);
```

```
request.setExpression(query);
request.setExpressionType(ExpressionType.SQL);
InputSerialization inputSerialization = new InputSerialization();
JSONInput jsonInput = new JSONInput();
jsonInput.setType(JSONType.LINES);
inputSerialization.setJson(jsonInput);
inputSerialization.setCompressionType(CompressionType.NONE);
request.setInputSerialization(inputSerialization);
OutputSerialization outputSerialization = new OutputSerialization();
outputSerialization.setJson(new JSONOutput());
request.setOutputSerialization(outputSerialization);
final AtomicBoolean isResultComplete = new AtomicBoolean(false);
SelectObjectContentResult result = cosclient.selectObjectContent(request);
InputStream resultInputStream = result.getPayload().getRecordsInputStream(
new SelectObjectContentEventVisitor() {
@Override
public void visit(SelectObjectContentEvent.StatsEvent event)
{
System.out.println(
"Received Stats, Bytes Scanned: " + event.getDetails().getBytesScanned()
+ " Bytes Processed: " + event.getDetails().getBytesProcessed());
}
@Override
public void visit(SelectObjectContentEvent.EndEvent event)
{
isResultComplete.set(true);
System.out.println("Received End Event. Result is complete.");
}
}
);
BufferedReader reader = new BufferedReader(new InputStreamReader(resultInputStrea
m));
StringBuffer stringBuffer = new StringBuffer();
String line;
while((line = reader.readLine())!= null){
stringBuffer.append(line).append("\n");
}
System.out.println(stringBuffer.toString());
// 检查结果是否接受完整
if (!isResultComplete.get()) {
throw new Exception("result was incomplete");
}
// 确认本进程不再使用 cosClient 实例之后，关闭之
cosClient.shutdown();
```

参数说明

参数名称	参数描述	类型
selectRequest	请求	SelectObjectContentRequest

SelectObjectContentRequest 成员说明:

参数名称	设置方法	描述	类型
bucketName	set 方法	Bucket 的命名格式为 BucketName-APPID ，详情请参见 命名规范	String
key	set 方法	指定分块上传到 COS 上的路径，即 对象键。例如对象键为 folder/picture.jpg	String
expression	set 方法	请求表达式	String
expressionType	set 方法	请求表达式类型	String
inputSerialization	set 方法	描述待检索对象的格式	InputSerialization
outputSerialization	set 方法	描述检索结果的输出格式	OutputSerialization

返回结果说明

- 成功：返回 InputStream。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

版权所有：腾讯云计算（北京）有限责任公司

自定义域名上传下载对象

最近更新时间：2022-12-15 16:03:52

简介

本文主要介绍如何使用自定义域名上传和下载对象，本文示例中使用了上传和下载对象的高级接口，接口详细信息可参见 Java SDK 上传对象 和 Java SDK 下载对象。

在使用自定义域名上传和下载对象之前，首先需要为存储桶设置自定义源站域名，详情请参见 开启自定义源站域名 的方式来设置自定义源站域名。

通过自定义域名上传对象

方法原型

```
// UserSpecifiedEndpointBuilder 构造函数
com.qcloud.cos.endpoint.UserSpecifiedEndpointBuilder(String generalApiEndpoint, S
tring getServiceApiEndpoint);
```

请求示例

```
import java.io.File;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import com.qcloud.cos.COSClient;
import com.qcloud.cos.ClientConfig;
import com.qcloud.cos.auth.BasicCOSCredentials;
import com.qcloud.cos.auth.COSCredentials;
import com.qcloud.cos.endpoint.UserSpecifiedEndpointBuilder;
import com.qcloud.cos.exception.CosClientException;
import com.qcloud.cos.exception.CosServiceException;
import com.qcloud.cos.http.HttpProtocol;
import com.qcloud.cos.model.PutObjectRequest;
import com.qcloud.cos.model.UploadResult;
import com.qcloud.cos.region.Region;
import com.qcloud.cos.transfer.TransferManager;
import com.qcloud.cos.transfer.Upload;
public static void uploadFile() {
// 1 初始化用户身份信息 (secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("secretId", "secretKey");
```

```
// 2 设置 bucket 的地域, COS_REGION 请参照 https://www.qcloud.com/document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("COS_REGION"));
// 设置请求协议, 推荐使用 https 协议
clientConfig.setHttpProtocol(HttpProtocol.https);
//若设置自定义源站域名时未上传 https 证书, 则改为 clientConfig.setHttpProtocol(HttpProtocol.http);
// 设置自定义的域名
UserSpecifiedEndpointBuilder endpointBuilder = new UserSpecifiedEndpointBuilder("generalApiEndpoint", "getServiceApiEndpoint");
clientConfig.setEndpointBuilder(endpointBuilder);
// 3 生成 cos 客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket 名需包含 appid
String bucketName = "BucketName-APPID";
ExecutorService threadPool = Executors.newFixedThreadPool(32);
// 传入一个 threadpool, 若不传入线程池, 默认 TransferManager 中会生成一个单线程的线程池。
TransferManager transferManager = new TransferManager(cosclient, threadPool);
// 设置对象的 key
String key = "exampleobject";
// 本地文件路径
File localFile = new File("/path/to/localFile");
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
try {
// 返回一个异步结果 Upload , 可同步的调用 waitForUploadResult 等待 upload 结束, 成功返回 UploadResult , 失败抛出异常.
Upload upload = transferManager.upload(putObjectRequest);
UploadResult uploadResult = upload.waitForUploadResult();
System.out.printf("RequestId : %s\n",uploadResult.getRequestId());
System.out.println(uploadResult.getETag());
System.out.println(uploadResult.getCrc64Ecma());
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
transferManager.shutdownNow();
}
```

参数说明

参数名	描述	类型

参数名	描述	类型
generalApiEndpoint	自定义源站域名	String
getServiceApiEndpoint	代表列举存储桶列表时的域名，推荐填 service.cos.myqcloud.com	String

通过自定义域名下载对象

方法原型

```
// UserSpecifiedEndpointBuilder 构造函数
com.qcloud.cos.endpoint.UserSpecifiedEndpointBuilder(String generalApiEndpoint, S
tring getServiceApiEndpoint);
```

请求示例

```
import java.io.File;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import com.qcloud.cos.COSClient;
import com.qcloud.cos.ClientConfig;
import com.qcloud.cos.auth.BasicCOSCredentials;
import com.qcloud.cos.auth.COSCredentials;
import com.qcloud.cos.endpoint.UserSpecifiedEndpointBuilder;
import com.qcloud.cos.exception.CosClientException;
import com.qcloud.cos.exception.CosServiceException;
import com.qcloud.cos.http.HttpProtocol;
import com.qcloud.cos.model.GetObjectRequest;
import com.qcloud.cos.region.Region;
import com.qcloud.cos.transfer.Download;
import com.qcloud.cos.transfer.TransferManager;
public static void downLoadFile() {
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("secretId", "secretKey");
// 2 设置 bucket 的地域, COS_REGION 请参照 https://www.qcloud.com/document/product/
436/6224
ClientConfig clientConfig = new ClientConfig(new Region("COS_REGION"));
// 设置请求协议, 推荐使用 https 协议
clientConfig.setHttpProtocol(HttpProtocol.https);
//若设置自定义源站域名时未上传 https 证书, 则改为 clientConfig.setHttpProtocol(HttpProt
ocol.http);
// 设置自定义的域名
UserSpecifiedEndpointBuilder endpointBuilder = new UserSpecifiedEndpointBuilder(
"generalApiEndpoint", "getServiceApiEndpoint");
```

```
clientConfig.setEndpointBuilder(endpointBuilder);
// 3 生成 cos 客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket 名需包含 appid
String bucketName = "BucketName-APPID";
ExecutorService threadPool = Executors.newFixedThreadPool(32);
// 传入一个 threadpool, 若不传入线程池, 默认 TransferManager 中会生成一个单线程的线程池。
TransferManager transferManager = new TransferManager(cosclient, threadPool);
// 设置对象的 key
String key = "exampleobject";
// 本地文件路径
File downloadFile = new File("/path/to/localFile");
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
try {
// 返回一个异步结果 download, 可同步的调用 waitForCompletion 等待 download 结束, 成功返回 void , 失败抛出异常.
Download download = transferManager.download(getObjectRequest, downloadFile);
download.waitForCompletion();
System.out.printf("RequestId : %s\n",download.getObjectMetadata().getRequestId());
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
transferManager.shutdownNow();
}
```

参数说明

参数名	描述	类型
generalApiEndpoint	自定义源站域名	String
getServiceApiEndpoint	代表列举存储桶列表时的域名，推荐填 service.cos.myqcloud.com	String

数据管理
生命周期

最近更新时间：2020-12-08 17:02:31

本文档提供关于生命周期的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket lifecycle	设置生命周期	设置存储桶的生命周期管理的配置
GET Bucket lifecycle	查询生命周期	查询存储桶生命周期管理的配置
DELETE Bucket lifecycle	删除生命周期	删除存储桶生命周期管理的配置

设置生命周期

功能说明

设置指定存储桶的生命周期配置信息（PUT Bucket lifecycle）。

方法原型

```
public void setBucketLifecycleConfiguration(String bucketName, BucketLifecycleCon
figuration bucketLifecycleConfiguration)
throws CosClientException, CosServiceException;
```

请求示例

```
List<BucketLifecycleConfiguration.Rule> rules = new ArrayList<BucketLifecycleConf
iguration.Rule>();
// 规则1 30天后删除路径以 hongkong_movie/ 为开始的文件
BucketLifecycleConfiguration.Rule deletePrefixRule = new BucketLifecycleConfigura
tion.Rule();
deletePrefixRule.setId("delete prefix xxxy after 30 days");
deletePrefixRule.setFilter(new LifecycleFilter(new LifecyclePrefixPredicate("hong
kong_movie/")));
// 文件上传或者变更后, 30天后删除
deletePrefixRule.setExpirationInDays(30);
// 设置规则为生效状态
deletePrefixRule.setStatus(BucketLifecycleConfiguration.ENABLED);
```

```
// 规则2 20天后沉降到低频，一年后删除
BucketLifecycleConfiguration.Rule standardIaRule = new BucketLifecycleConfigurati
on.Rule();
standardIaRule.setId("standard_ia transition");
standardIaRule.setFilter(new LifecycleFilter(new LifecyclePrefixPredicate("standa
rd_ia/")));
List<BucketLifecycleConfiguration.Transition> standardIaTransitions = new ArrayLi
st<BucketLifecycleConfiguration.Transition>();
BucketLifecycleConfiguration.Transition standardTransition = new BucketLifecycleC
onfiguration.Transition();
standardTransition.setDays(20);
standardTransition.setStorageClass(StorageClass.Standard_IA.toString());
standardIaTransitions.add(standardTransition);
standardIaRule.setTransitions(standardIaTransitions);
standardIaRule.setStatus(BucketLifecycleConfiguration.ENABLED);
standardIaRule.setExpirationInDays(365);

// 将两条规则添加到策略集合中
rules.add(deletePrefixRule);
rules.add(standardIaRule);

// 生成 bucketLifecycleConfiguration
BucketLifecycleConfiguration bucketLifecycleConfiguration =
new BucketLifecycleConfiguration();
bucketLifecycleConfiguration.setRules(rules);

// 存储桶的命名格式为 BucketName-APPID
String bucketName = "examplebucket-1250000000";
SetBucketLifecycleConfigurationRequest setBucketLifecycleConfigurationRequest =
new SetBucketLifecycleConfigurationRequest(bucketName, bucketLifecycleConfigurati
on);

// 设置生命周期
cosClient.setBucketLifecycleConfiguration(setBucketLifecycleConfigurationReques
t);
```

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
bucketLifecycleConfiguration	生命周期配置	BucketLifecycleConfiguration

返回结果说明

版权所有：腾讯云计算（北京）有限责任公司

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。具体请参见 异常处理。

查询生命周期

功能说明

查询存储桶的生命周期管理配置（GET Bucket lifecycle）。

方法原型

```
public BucketLifecycleConfiguration getBucketLifecycleConfiguration(String bucketName)
throws CosClientException, CosServiceException;
```

请求示例

```
// 存储桶的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
BucketLifecycleConfiguration queryLifeCycleRet =
cosClient.getBucketLifecycleConfiguration(bucketName);
List<BucketLifecycleConfiguration.Rule> ruleLists = queryLifeCycleRet.getRules();
```

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String

返回结果说明

- 成功：返回 BucketLifecycleConfiguration 类， 包含存储桶的生命周期规则。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。具体请参见 异常处理。

删除生命周期

功能说明

删除存储桶生命周期管理的配置（DELETE Bucket lifecycle）。

版权所有：腾讯云计算（北京）有限责任公司

方法原型

```
public void deleteBucketLifecycleConfiguration(String bucketName)
throws CosClientException, CosServiceException;
```

请求示例

```
//存储桶的命名格式为 BucketName-APPID
String bucketName = "examplebucket-1250000000";
cosClient.deleteBucketLifecycleConfiguration(bucketName);
```

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。具体请参见 异常处理。

日志管理

最近更新时间：2020-03-11 17:48:25

简介

本文档提供关于日志管理的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket logging	设置日志管理	为源存储桶开启日志记录
GET Bucket logging	查询日志管理	查询源存储桶的日志配置信息

设置日志管理

功能说明

PUT Bucket logging 用于为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中。

方法原型

```
public void setBucketLoggingConfiguration(SetBucketLoggingConfigurationRequest setBucketLoggingConfigurationRequest);
```

请求示例

```
String bucketName = "examplebucket-1250000000";
BucketLoggingConfiguration bucketLoggingConfiguration = new BucketLoggingConfiguration();
bucketLoggingConfiguration.setDestinationBucketName(bucketName);
bucketLoggingConfiguration.setLogFilePrefix("logs");
SetBucketLoggingConfigurationRequest setBucketLoggingConfigurationRequest = new SetBucketLoggingConfigurationRequest(bucketName, bucketLoggingConfiguration);
cosclient.setBucketLoggingConfiguration(setBucketLoggingConfigurationRequest);
```

参数说明

参数名称	描述	类型
setBucketLoggingConfigurationRequest	开启日志功能请求	SetBucketLoggingConfigurationRequest

版权所有：腾讯云计算（北京）有限责任公司

Request 成员说明 ：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	开启日志功能的源存储桶名称，命名格式为 BucketName-APPID ，详情请参见 命名规范	String
loggingConfiguration	构造函数或 set 方法	存储桶的日志功能配置	BucketLoggingConfiguration

BucketLoggingConfiguration 成员说明:

参数名称	描述	类型
destinationBucketName	存放日志的目标存储桶名称，命名格式为 BucketName-APPID ，详情请参见 命名规范	String
logFilePrefix	日志存放在目标存储桶的指定路径	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询日志管理

功能说明

GET Bucket logging 用于查询指定存储桶的日志配置信息。

方法原型

```
public BucketLoggingConfiguration getBucketLoggingConfiguration(String bucketNam
e);
```

请求示例

```
String bucketName = "examplebucket-1250000000";
BucketLoggingConfiguration bucketLoggingConfiguration = cosclient.getBucketLoggin
gConfiguration(bucketName);
```

版权所有：腾讯云计算（北京）有限责任公司

参数说明

参数名称	描述	类型
bucketName	存放日志的目标存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String

返回结果说明

- 成功：返回 BucketLoggingConfiguration，包含存放日志的目标存储桶名称、日志存放在目标存储桶的指定路径。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

存储桶标签

最近更新时间：2021-08-31 17:34:19

简介

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket tagging	设置存储桶标签	为已存在的存储桶设置标签
GET Bucket tagging	查询存储桶标签	查询指定存储桶下已有的存储桶标签
DELETE Bucket tagging	删除存储桶标签	删除指定的存储桶标签

设置存储桶标签

功能说明

PUT Bucket tagging 用于为已存在的存储桶设置标签。

方法原型

```
public void setBucketTaggingConfiguration(SetBucketTaggingConfigurationRequest se
tBucketTaggingConfigurationRequest);
public void setBucketTaggingConfiguration(String bucketName, BucketTaggingConfigu
ration bucketTaggingConfiguration);
```

请求示例

```
String bucketName = "examplebucket-1250000000";
List<TagSet> tagSetList = new LinkedList<TagSet>();
TagSet tagSet = new TagSet();
tagSet.setTag("age", "18");
tagSet.setTag("name", "xiaoming");
tagSetList.add(tagSet);
BucketTaggingConfiguration bucketTaggingConfiguration = new BucketTaggingConfigur
ation();
bucketTaggingConfiguration.setTagSets(tagSetList);
SetBucketTaggingConfigurationRequest setBucketTaggingConfigurationRequest =
```

版权所有：腾讯云计算（北京）有限责任公司

```
new SetBucketTaggingConfigurationRequest(bucketName, bucketTaggingConfiguration);
cosClient.setBucketTaggingConfiguration(setBucketTaggingConfigurationRequest);
```

参数说明

参数名称	描述	类型
setBucketTaggingConfigurationRequest	存储桶标签设置请求	SetBucketTaggingConfigurationRequest

Request 成员说明 ：

Request 成员	设置方法	描述	类型
bucketName	构造函数 或 set 方法	设置标签的存储桶，格式为 BucketName-APPID，详情请参见 命名规范	String
taggingConfiguration	构造函数 或 set 方法	存储桶的标签配置	BucketTaggingConfiguration

BucketLoggingConfiguration 成员说明:

参数名称	描述	类型
tagSets	存储桶的标签配置集合	List&dxlt;TagSet&dxgt;

TagSet 成员说明：

参数名称	描述	类型
tags	标签的 Key 和 Value，长度不超过128字节, Key 和 Value 支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	Map&dxlt;String, String&dxgt;

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询存储桶标签

功能说明

GET Bucket tagging 用于查询指定存储桶下已有的存储桶标签。

方法原型

```
public BucketTaggingConfiguration getBucketTaggingConfiguration(String bucketNam
e);
```

请求示例

```
String bucketName = "examplebucket-1250000000";
BucketTaggingConfiguration bucketTaggingConfiguration = cosClient.getBucketTaggin
gConfiguration(bucketName);
```

参数说明

参数名称	描述	类型
bucketName	查询标签的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String

返回结果说明

- 成功：返回 BucketTaggingConfiguration，包含存储桶的标签设置信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

删除存储桶标签

功能说明

DELETE Bucket tagging 用于删除指定存储桶下已有的存储桶标签。

方法原型

```
public void deleteBucketTaggingConfiguration(String bucketName);
```

请求示例

```
String bucketName = "examplebucket-1250000000";
cosClient.deleteBucketTaggingConfiguration(bucketName);
```

参数说明

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
bucketName	被删除标签的存储桶，格式为 BucketName-APPID ， 详情请参见 命名规范	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

静态网站

最近更新时间：2020-03-11 17:49:13

简介

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket website	设置静态网站	设置存储桶的静态网站配置
GET Bucket website	查询静态网站配置	查询存储桶的静态网站配置
DELETE Bucket website	删除静态网站配置	删除存储桶的静态网站配置

设置静态网站

功能说明

PUT Bucket website 用于为存储桶配置静态网站。

方法原型

```
public void setBucketWebsiteConfiguration(String bucketName, BucketWebsiteConfigu
ration configuration)
throws CosClientException, CosServiceException;
public void setBucketWebsiteConfiguration(SetBucketWebsiteConfigurationRequest se
tBucketWebsiteConfigurationRequest)
throws CosClientException, CosServiceException;
```

请求示例

```
String bucketName = "examplebucket-1250000000";
// 设置bucket website
BucketWebsiteConfiguration bucketWebsiteConfiguration = new BucketWebsiteConfigur
ation();
// 索引文件
bucketWebsiteConfiguration.setIndexDocumentSuffix("index.html");
// 路由规则
List<RoutingRule> routingRuleList = new ArrayList<RoutingRule>();
RoutingRule routingRule = new RoutingRule();
```

版权所有：腾讯云计算（北京）有限责任公司

```
RoutingRuleCondition routingRuleCondition = new RoutingRuleCondition();
routingRuleCondition.setHttpErrorCodeReturnedEquals("404");
routingRule.setCondition(routingRuleCondition);
RedirectRule redirectRule = new RedirectRule();
redirectRule.setProtocol("https");
redirectRule.setReplaceKeyPrefixWith("404.html");
routingRule.setRedirect(redirectRule);
routingRuleList.add(routingRule);
bucketWebsiteConfiguration.setRoutingRules(routingRuleList);
cosclient.setBucketWebsiteConfiguration(bucketName, bucketWebsiteConfiguration);
```

参数说明

参数名称	描述	类型
setBucketWebsiteConfigurationRequest	存储桶静态网站设置请求	SetBucketWebsiteConfigurationRequest

Request 成员说明 ：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	设置静态网站的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
configuration	构造函数或 set 方法	存储桶的静态网站配置	BucketWebsiteConfiguration

BucketWebsiteConfiguration 成员说明:

参数名称	描述	类型
indexDocumentSuffix	索引文档	String
errorDocument	错误文档	String
redirectAllRequestsTo	重定向所有请求配置	RedirectRule
routingRules	设置重定向规则，最多设置100条 RoutingRule	List

RoutingRule 成员说明:

参数名称	描述	类型

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
condition	指定重定向发生的条件，前缀匹配重定向和错误码重定向只能指定一个	RoutingRuleCondition
redirect	重定向规则	RedirectRule

RoutingRuleCondition 成员说明:

参数名称	描述	类型
keyPrefixEquals	指定前缀重定向的路径	RoutingRuleCondition
httpErrorCodeReturnedEquals	指定重定向错误码	RedirectRule

RedirectRule 成员说明:

参数名称	描述	类型
protocol	指定全站重定向的协议	String
replaceKeyPrefixWith	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String
replaceKeyWith	替换整个 Key 为指定的内容	String
httpRedirectCode	指定全站重定向的协议	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询静态网站配置

功能说明

GET Bucket website 用于查询与存储桶关联的静态网站配置信息。

方法原型

```
public BucketWebsiteConfiguration getBucketWebsiteConfiguration(String bucketNam
e)
throws CosClientException, CosServiceException;
```

请求示例

```
String bucketName = "examplebucket-1250000000";
BucketWebsiteConfiguration bucketWebsiteConfiguration = cosclient.getBucketWebsit
eConfiguration(bucketName);
```

参数说明

参数名称	描述	类型
bucketName	查询静态网站配置的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String

返回结果说明

- 成功：返回 BucketWebsiteConfiguration，包含存储桶的静态网站设置信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

删除静态网站配置

功能说明

DELETE Bucket website 用于删除存储桶中的静态网站配置。

方法原型

```
public void deleteBucketWebsiteConfiguration(String bucketName) throws CosClientE
xception, CosServiceException;
```

请求示例

```
String bucketName = "examplebucket-1250000000";
cosclient.deleteBucketWebsiteConfiguration(bucketName);
```

参数说明

参数名称	描述	类型
bucketName	被删除静态网站配置的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String

返回结果说明

- 成功：无返回值。

- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

对象标签

最近更新时间：2021-11-19 12:55:37

简介

本文档提供关于对象标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Object tagging	设置对象标签	为对象设置标签
GET Object tagging	查询对象标签	查询指定对象下已有的对象标签
DELETE Object tagging	删除对象标签	删除指定对象下已有的对象标签

设置对象标签

功能说明

用于为对象设置标签。

方法原型

```
public SetObjectTaggingResult setObjectTagging(SetObjectTaggingRequest setObjectTaggingRequest);
```

请求示例1：对已上传的对象设置标签

```
String bucketName = "examplebucket-1250000000";
String key = "exampletkey";
List<Tag> tags = new LinkedList<>();
tags.add(new Tag("tag1", "value1"));
tags.add(new Tag("tag2", "value2"));
ObjectTagging objectTagging = new ObjectTagging(tags);
SetObjectTaggingRequest setObjectTaggingRequest = new SetObjectTaggingRequest(bucketName, key, objectTagging);
cosclient.setObjectTagging(setObjectTaggingRequest);
```

请求示例2：上传对象时设置标签

```
String bucketName = "examplebucket-1250000000";
String key = "testfiles/testTagging.txt";
InputStream is = new ByteArrayInputStream(new byte[]{'d', 'a', 't', 'a'});
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setHeader("x-cos-tagging", "tag1=value1&tag2=value2");
cosclient.putObject(bucketName, key, is, objectMetadata);
```

参数说明

参数名称	描述	类型
setObjectTaggingRequest	对象标签设置请求	SetObjectTaggingRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	设置标签的对象所在的存储桶，格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或者 set 方法	设置标签的对象键，对象键（Key）是对象在存储桶中的唯一标识，详情请参见 对象键	String
objectTagging	构造函数或 set 方法	对象的标签配置	ObjectTagging

ObjectTagging 成员说明：

参数名称	描述	类型
tagSet	对象的标签配置集合	List&dxlt;Tag>

Tag 成员说明：

参数名称	描述	类型
key	标签的 key	String
value	标签的 value	String

返回结果说明

- 成功：无返回值。

- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询对象标签

功能说明

查询指定对象下已有的对象标签。

方法原型

```
public GetObjectTaggingResult getObjectTagging(GetObjectTaggingRequest getObjectTaggingRequest);
```

请求示例

```
String bucketName = "exampletbucket-1250000000";
String key = "exampletkey";
GetObjectTaggingRequest getObjectTaggingRequest = new GetObjectTaggingRequest(bucketName, key);
GetObjectTaggingResult getObjectTaggingResult = cosclient.getObjectTagging(getObjectTaggingRequest);
List<Tag> resultTagSet = getObjectTaggingResult.getTagSet();
System.out.println(resultTagSet.toString());
```

参数说明

参数名称	描述	类型
getObjectTaggingRequest	对象标签查询请求	GetObjectTaggingRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	设置标签的对象所在的存储桶，格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或者 set 方法	设置标签的对象键，对象键（Key）是对象在存储桶中的唯一标识，详情请参见 对象键	String

返回结果说明

- 成功：返回 GetObjectTaggingResult，包含对象的标签信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

删除对象标签

功能说明

用于删除指定对象的已有标签。

方法原型

```
public DeleteObjectTaggingResult deleteObjectTagging(DeleteObjectTaggingRequest d
eleteObjectTaggingRequest);
```

请求示例

```
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
DeleteObjectTaggingRequest deleteObjectTaggingRequest = new DeleteObjectTaggingRe
quest(bucketName, key);
cosclient.deleteObjectTagging(deleteObjectTaggingRequest);
```

参数说明

参数名称	描述	类型
deleteObjectTaggingRequest	对象标签删除请求	DeleteObjectTaggingRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	设置标签的对象所在的存储桶，格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或者 set 方法	设置标签的对象键，对象键（Key）是对象在存储桶中的唯一标识，详情请参见 对象键	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

清单

最近更新时间：2020-12-07 17:52:05

简介

本文档提供关于清单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket inventory	设置清单任务	设置存储桶的清单任务
GET Bucket inventory	查询清单任务	查询存储桶的清单任务
DELETE Bucket inventory	删除清单任务	删除存储桶的清单任务

设置清单任务

功能说明

PUT Bucket inventory 用于在存储桶中创建清单任务。

方法原型

```
public SetBucketInventoryConfigurationResult setBucketInventoryConfiguration(Stri
ng bucketName, InventoryConfiguration inventoryConfiguration);
public SetBucketInventoryConfigurationResult setBucketInventoryConfiguration(SetB
ucketInventoryConfigurationRequest setBucketInventoryConfigurationRequest);
```

请求示例

```
// bucket 名需包含 appid
String bucketName = "examplebucket-1250000000";

InventoryConfiguration inventoryConfiguration = new InventoryConfiguration();
InventoryCosBucketDestination inventoryCosBucketDestination = new InventoryCosBuc
ketDestination();
// 设置清单的输出目标存储桶的格式和前缀等
inventoryCosBucketDestination.setAccountId("2779643970");
inventoryCosBucketDestination.setBucketArn("qcs::cos:ap-guangzhou::examplebucket-
1250000000");
inventoryCosBucketDestination.setEncryption(new ServerSideEncryptionCOS());
```

版权所有：腾讯云计算（北京）有限责任公司

```
inventoryCosBucketDestination.setFormat(InventoryFormat.CSV);
inventoryCosBucketDestination.setPrefix("inventory-output");
InventoryDestination inventoryDestination = new InventoryDestination();
inventoryDestination.setCosBucketDestination(inventoryCosBucketDestination);
inventoryConfiguration.setDestination(inventoryDestination);

// 设置清单的调度周期，扫描前缀和 ID 等
inventoryConfiguration.setEnabled(true);
inventoryConfiguration.setId("1");
InventorySchedule inventorySchedule = new InventorySchedule();
inventorySchedule.setFrequency(InventoryFrequency.Daily);
inventoryConfiguration.setSchedule(inventorySchedule);
InventoryPrefixPredicate inventoryFilter = new InventoryPrefixPredicate("test/");
inventoryConfiguration.setInventoryFilter(new InventoryFilter(inventoryFilter));
inventoryConfiguration.setIncludedObjectVersions(InventoryIncludedObjectVersions.
All);
// 设置可选的输出字段
List<String> optionalFields = new LinkedList<String>();
optionalFields.add(InventoryOptionalField.Size.toString());
optionalFields.add(InventoryOptionalField.LastModifiedDate.toString());
inventoryConfiguration.setOptionalFields(optionalFields);
SetBucketInventoryConfigurationRequest setBucketInventoryConfigurationRequest = n
ew SetBucketInventoryConfigurationRequest();
setBucketInventoryConfigurationRequest.setBucketName(bucketName);
setBucketInventoryConfigurationRequest.setInventoryConfiguration(inventoryConfigu
ration);
cosClient.setBucketInventoryConfiguration(setBucketInventoryConfigurationReques
t);

inventoryConfiguration.setId("2");
inventorySchedule.setFrequency(InventoryFrequency.Weekly);
cosClient.setBucketInventoryConfiguration(setBucketInventoryConfigurationReques
t);
```

参数说明

参数名称	描述	类型
setBucketInventoryConfigurationRequest	存储桶静态网站设置请求	SetBucketInventoryConfigurationRequest

Request 成员说明 ：

Request 成员	设置方法	描述	类型
bucketName	构造函数或	设置清单任务的存储桶，格式为	String

	set 方法	BucketName-APPID，详情请参见 命名规范	
inventoryConfiguration	构造函数或 set 方法	存储桶的清单配置	InventoryConfiguration

InventoryConfiguration 成员说明:

参数名称	描述	类型
id	清单任务的名称，合法字符：a-z，A-Z，0-9，-，_，.	String
destination	清单结果导出后存放的存储桶信息	InventoryDestination
isEnabled	清单是否启用的标识	Boolean
inventoryFilter	筛选待分析对象。清单功能将分析符合 Filter 中设置的前缀的对象	InventoryFilter
includedObjectVersions	是否在清单中包含对象版本 • 如果设置为 All，清单中将会包含所有对象版本，并在清单中增加 VersionId，IsLatest，DeleteMarker 这几个字段 • 如果设置为 Current，则清单中不包含对象版本信息	String
optionalFields	清单结果中可选的字段名称	List
schedule	配置清单任务周期	InventorySchedule

InventoryDestination 成员说明：

参数名称	描述	类型
cosBucketDestination	清单结果导出后存放的存储桶信息	InventoryCosBucketDestination

InventoryCosBucketDestination 成员说明：

参数名称	描述	类型
accountId	存储桶的所有者 ID	String
bucketArn	清单分析结果的存储桶名	String
format	清单分析结果的文件形式，可选项为 CSV 格式	String
prefix	清单分析结果的前缀	String

版权所有：腾讯云计算（北京）有限责任公司

encryption	为清单结果提供服务端加密的选项	InventoryEncryption

InventoryFilter 成员说明：

参数名称	描述	类型
predicate	筛选待分析对象	InventoryFilterPredicate

InventoryFilter 的一个实现类为 InventoryPrefixPredicate，成员说明如下：

参数名称	描述	类型
prefix	需要分析的对象的前缀	String

InventorySchedule 成员说明：

参数名称	描述	类型
frequency	清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

错误码说明

该请求可能会发生的一些常见的特殊错误如下：

错误码	描述	状态码
InvalidArgument	不合法的参数值	HTTP 400 Bad Request
TooManyConfigurations	清单数量已经达到1000条的上限	HTTP 400 Bad Request
AccessDenied	未授权的访问。您可能不具备访问该存储桶的权限	HTTP 403 Forbidden

查询清单任务

功能说明

GET Bucket inventory 用于查询存储桶中用户的清单任务信息。

方法原型

```
public GetBucketInventoryConfigurationResult getBucketInventoryConfiguration(String bucketName, String id) throws CosClientException, CosServiceException;
```

请求示例

```
String bucketName = "examplebucket-1250000000";
String id = "1";
GetBucketInventoryConfigurationResult getBucketInventoryConfigurationResult = cosClient.getBucketInventoryConfiguration(bucketName, id);
```

参数说明

参数名称	描述	类型
bucketName	查询清单任务的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
id	清单任务的名称，合法字符：a-z，A-Z，0-9，-，_，.	String

返回结果说明

- 成功：返回 GetBucketInventoryConfigurationResult，包含存储桶对应 id 清单的配置信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询所有清单

功能说明

List Bucket Inventory Configurations 用于请求返回一个存储桶中的所有清单任务。

方法原型

```
public ListBucketInventoryConfigurationsResult listBucketInventoryConfigurations(ListBucketInventoryConfigurationsRequest listBucketInventoryConfigurationsRequest) throws CosClientException, CosServiceException;
```

请求示例

```
String bucketName = "examplebucket-1250000000";
ListBucketInventoryConfigurationsRequest listBucketInventoryConfigurationsRequest = new ListBucketInventoryConfigurationsRequest();
```

```
listBucketInventoryConfigurationsRequest.setBucketName(bucketName);
ListBucketInventoryConfigurationsResult listBucketInventoryConfigurationsResult =
cosClient.listBucketInventoryConfigurations(listBucketInventoryConfigurationsRequ
est);
```

参数说明

参数名称	描述	类型
bucketName	存放清单的目标存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
continuationToken	当 COS 响应体中 IsTruncated 为 true，且 NextContinuationToken 节点中存在参数值时，您可以将这个参数作为 continuation-token 参数值，以获取下一页的清单任务信息。	String

返回结果说明

- 成功：返回 ListBucketInventoryConfigurationsResult，包含存储桶对应 id 清单的配置信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

删除清单任务

功能说明

DELETE Bucket inventory 用于删除存储桶中指定的清单任务。

方法原型

```
public DeleteBucketInventoryConfigurationResult deleteBucketInventoryConfiguratio
n(String bucketName, String id) throws CosClientException, CosServiceException;
```

请求示例

```
String bucketName = "examplebucket-1250000000";
String id = "1";
cosClient.deleteBucketInventoryConfiguration(bucketName, id);
```

参数说明

参数名称	描述	类型
bucketName	被删除清单任务的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String

id	清单任务的名称，合法字符：a-z，A-Z，0-9，-，_，.	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

异地容灾
版本控制

最近更新时间：2020-12-08 17:02:31

简介

本文档提供关于版本控制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

设置版本控制

功能说明

设置指定存储桶的版本控制功能（PUT Bucket versioning）。

方法原型

```
public void setBucketVersioningConfiguration(SetBucketVersioningConfigurationRequest setBucketVersioningConfigurationRequest)
throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
setBucketVersioningConfigurationRequest	版本控制配置	SetBucketVersioningConfigurationRequest

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

开启版本控制

```
String bucketName = "examplebucket-1250000000";
// 开启版本控制
cosClient.setBucketVersioningConfiguration(
new SetBucketVersioningConfigurationRequest(bucketName,
new BucketVersioningConfiguration(BucketVersioningConfiguration.ENABLED)));
```

暂停版本控制

```
String bucketName = "examplebucket-1250000000";
// 暂停版本控制
cosClient.setBucketVersioningConfiguration(
new SetBucketVersioningConfigurationRequest(bucketName,
new BucketVersioningConfiguration(BucketVersioningConfiguration.SUSPENDED)));
```

查询版本控制

功能说明

查询指定存储桶的版本控制信息（GET Bucket versioning）。

方法原型

```
// 方法1 传入存储桶名称即可
public BucketVersioningConfiguration getBucketVersioningConfiguration(String buck
etName)
throws CosClientException, CosServiceException;

// 方法2 通过GetBucketVersioningConfigurationRequest 获取
public BucketVersioningConfiguration getBucketVersioningConfiguration(
GetBucketVersioningConfigurationRequest getBucketVersioningConfigurationRequest)
throws CosClientException, CosServiceException;
```

请求示例

```
String bucketName = "examplebucket-1250000000";
// 获取版本控制
BucketVersioningConfiguration bvc =
cosClient.getBucketVersioningConfiguration(bucketName);
// 获取版本控制
BucketVersioningConfiguration bvc2 = cosClient.getBucketVersioningConfiguration(
new GetBucketVersioningConfigurationRequest(bucketName));
```

版权所有：腾讯云计算（北京）有限责任公司

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
getBucketVersioningConfigurationRequest	获取版本控制配置请求	GetBucketVersioningConfigurationRequest

返回结果说明

- 成功：返回存储桶的多版本配置。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

存储桶复制

最近更新时间：2021-11-18 12:38:03

简介

本文档提供关于存储桶复制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket replication	设置存储桶复制	设置存储桶的存储桶复制规则
GET Bucket replication	查询存储桶复制	查询存储桶的存储桶复制规则
DELETE Bucket replication	删除存储桶复制	删除存储桶的存储桶复制规则

设置存储桶复制

功能说明

设置指定存储桶的存储桶复制规则（PUT Bucket replication）。

方法原型

```
public void setBucketReplicationConfiguration(
SetBucketReplicationConfigurationRequest setBucketReplicationConfigurationReques
t)
throws CosClientException, CosServiceException;
```

请求示例

```
// 源存储桶名称，需包含 appid
String bucketName = "examplebucket-1250000000";
BucketReplicationConfiguration bucketReplicationConfiguration = new BucketReplica
tionConfiguration();
// 设置发起者身份，格式为： qcs::cam::uin/<OwnerUin>:uin/<SubUin>
bucketReplicationConfiguration.setRoleName("qcs::cam::uin/100000000001:uin/100000
000001");
// 设置目标存储桶和存储类型，QCS 的格式为：qcs::cos:[region]::[bucketname-AppId]
ReplicationDestinationConfig replicationDestinationConfig = new ReplicationDestin
ationConfig();
replicationDestinationConfig.setBucketQCS("qcs::cos:ap-beijing::destinationbucket
```

```
-1250000000");
replicationDestinationConfig.setStorageClass(StorageClass.Standard);
// 设置规则状态和前缀
ReplicationRule replicationRule = new ReplicationRule();
replicationRule.setStatus(ReplicationRuleStatus.Enabled);
replicationRule.setPrefix("");
replicationRule.setDestinationConfig(replicationDestinationConfig);
// 添加规则
String ruleId = "replication-to-beijing";
bucketReplicationConfiguration.addRule(replicationRule);
SetBucketReplicationConfigurationRequest setBucketReplicationConfigurationRequest
=
new SetBucketReplicationConfigurationRequest(bucketName, bucketReplicationConfigu
ration);
cosClient.setBucketReplicationConfiguration(setBucketReplicationConfigurationRequ
est);
```

参数说明

参数名	参数描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
setBucketReplicationConfigurationRequest	存储桶复制配置	SetBucketReplicationConfigurationRequest

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询存储桶复制

功能说明

查询指定存储桶的存储桶复制规则（GET Bucket replication）。

方法原型

```
// 获取存储桶存储桶复制配置方法1
public BucketReplicationConfiguration getBucketReplicationConfiguration(String bucketName)
throws CosClientException, CosServiceException;
// 获取存储桶存储桶复制方法2
public BucketReplicationConfiguration getBucketReplicationConfiguration(
GetBucketReplicationConfigurationRequest getBucketReplicationConfigurationRequest)
throws CosClientException, CosServiceException;
```

请求示例

```
String bucketName = "examplebucket-1250000000";
// 获取存储桶存储桶复制配置方法1
BucketReplicationConfiguration brcfRet = cosClient.getBucketReplicationConfiguration(bucketName);
// 获取存储桶存储桶复制配置方法2
BucketReplicationConfiguration brcfRet2 = cosClient.getBucketReplicationConfiguration(
new GetBucketReplicationConfigurationRequest(bucketName));
```

参数说明

参数名	参数描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
getBucketReplicationConfigurationRequest	获取存储桶复制配置请求	GetBucketReplicationConfigurationRequest

返回结果说明

- 成功：返回存储桶的存储桶复制规则。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

删除存储桶复制

功能说明

删除指定存储桶的存储桶复制规则（DELETE Bucket replication）。

方法原型

```
// 删除存储桶存储桶复制配置方法1
public void deleteBucketReplicationConfiguration(String bucketName)
throws CosClientException, CosServiceException;
// 删除存储桶存储桶复制方法2
public void deleteBucketReplicationConfiguration(
DeleteBucketReplicationConfigurationRequest deleteBucketReplicationConfigurationR
equest)
throws CosClientException, CosServiceException;
```

请求示例

```
String bucketName = "examplebucket-1250000000";
// 删除存储桶存储桶复制配置方法1
cosClient.deleteBucketReplicationConfiguration(bucketName);
// 删除存储桶存储桶复制配置方法2
cosClient.deleteBucketReplicationConfiguration(new DeleteBucketReplicationConfigu
rationRequest(bucketName));
```

参数说明

参数名	参数描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
deleteBucketReplicationConfigurationRequest	删除存储桶复制配置请求	DeleteBucketReplicationConfigurationRequest

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

版权所有：腾讯云计算（北京）有限责任公司

访问管理
防盗链

最近更新时间：2021-11-18 14:29:35

简介

本文档提供关于存储桶 Referer 白名单或者黑名单的 API 概览以及 SDK 示例代码。

> 注意：
> 需要 COS Java SDK v5.6.52 及以上版本。

API	操作名	操作描述
PUT Bucket referer	设置存储桶 Referer	设置存储桶 Referer 白名单或者黑名单
GET Bucket referer	查询存储桶 Referer	查询存储桶 Referer 白名单或者黑名单

设置存储桶 Referer

功能说明

设置指定存储桶的 Referer 白名单或者黑名单（PUT Bucket referer）。

方法原型

```
public void setBucketRefererConfiguration(String bucketName, BucketRefererConfigu
ration configuration) throws CosClientException, CosServiceException;
public void setBucketRefererConfiguration(SetBucketRefererConfigurationRequest se
tBucketRefererConfigurationRequest) throws CosClientException, CosServiceExceptio
n;
```

请求示例

```
// 源存储桶名称，需包含 appid
String bucketName = "examplebucket-1250000000";
BucketRefererConfiguration configuration = new BucketRefererConfiguration();
// 启用防盗链
```

```
configuration.setStatus(BucketRefererConfiguration.DISABLED);
// 设置防盗链类型为白名单
//configuration.setRefererType(BucketRefererConfiguration.WHITELIST);
// 设置防盗链类型为黑名单 （与白名单二选一）
configuration.setRefererType(BucketRefererConfiguration.BLACKLIST);
// 填写要设置的域名
configuration.addDomain("test.com");
configuration.addDomain("test.1.com");
//（可选）设置是否允许空防盗链访问，缺省就是 DENY
configuration.setEmptyReferConfiguration(BucketRefererConfiguration.DENY);
// configuration.setEmptyReferConfiguration(BucketRefererConfiguration.ALLOW);
cosClient.setBucketRefererConfiguration(bucketName, configuration);
```

参数说明

参数名	参数描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
configuration	存储桶 Referer 配置	BucketRefererConfiguration

BucketRefererConfiguration 说明：

参数名	参数描述	类型	必选	方法
Status	是否开启防盗链，枚举值：Enabled、Disabled	String	是	setStatus
RefererType	防盗链类型，枚举值：Black-List、White-List	String	是	setRefererType
Domain	生效域名，支持带端口和 IP、支持通配符*, 支持多条	String	是	addDomain
EmptyReferConfiguration	是否允许空 Refer 访问，枚举值： Allow、Deny	String	否	setEmptyReferConfiguration

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询存储桶 Referer

功能说明

查询指定存储桶 Referer 白名单或者黑名单（GET Bucket referer）。

方法原型

```
public BucketRefererConfiguration getBucketRefererConfiguration(String bucketNam
e) throws CosClientException, CosServiceException
```

请求示例

```
// 源存储桶名称，需包含 appid
String bucketName = "examplebucket-1250000000";
BucketRefererConfiguration configuration = cosClient.getBucketRefererConfiguratio
n(bucketName);
if (configuration == null) {
System.out.printf("bucket %s does not have referer configuration\n", bucketName);
return;
}
System.out.printf("status: %s\n", configuration.getStatus());
System.out.printf("referer type: %s\n", configuration.getRefererType());
System.out.printf("empty referer config: %s\n", configuration.getEmptyReferConfig
uration());
for (String domain : configuration.getDomainList()) {
System.out.printf("domain: %s\n", domain);
}
```

参数说明

参数名	参数描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String

返回结果说明

- 成功：返回存储桶的 Referer 白名单或者黑名单。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

跨域访问

最近更新时间：2022-05-09 14:27:13

简介

本文档提供关于跨域访问的 API 概览以及 SDK 示例代码。

关于跨域访问的更多说明详见 跨域访问， 通过控制台设置跨域的操作步骤请参见 设置跨域访问 或最佳实践文档 跨域访问。

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域名访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域名访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域名访问配置信息

设置跨域配置

功能说明

设置指定存储桶的跨域名访问配置信息（PUT Bucket cors）。

方法原型

```
public void setBucketCrossOriginConfiguration(String bucketName, BucketCrossOrigi
nConfiguration bucketCrossOriginConfiguration);
```

请求示例

```
// bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
BucketCrossOriginConfiguration bucketCORS = new BucketCrossOriginConfiguration();
List<CORSRule> corsRules = new ArrayList<CORSRule>();
CORSRule corsRule = new CORSRule();
// 规则名称
corsRule.setId("set-bucket-cors-test");
// 允许的 HTTP 方法
corsRule.setAllowedMethods(CORSRule.AllowedMethods.PUT, CORSRule.AllowedMethods.G
ET, CORSRule.AllowedMethods.HEAD);
```

版权所有：腾讯云计算（北京）有限责任公司

```
corsRule.setAllowedHeaders("x-cos-grant-full-control");
corsRule.setAllowedOrigins("http://mail.qq.com", "http://www.qq.com", "http://vid
eo.qq.com");
corsRule.setExposedHeaders("x-cos-request-id");
corsRule.setMaxAgeSeconds(60);
corsRules.add(corsRule);
bucketCORS.setRules(corsRules);
cosClient.setBucketCrossOriginConfiguration(bucketName, bucketCORS);
```

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
bucketCrossOriginConfiguration	设置的存储桶跨域策略	BucketCrossOriginConfiguration

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。具体请参见 异常处理。

查询跨域配置

功能说明

查询指定存储桶的跨域名访问配置信息（GET Bucket cors）。

方法原型

```
public BucketCrossOriginConfiguration getBucketCrossOriginConfiguration(String bu
cketName)
throws CosClientException, CosServiceException;
```

请求示例

```
// bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
BucketCrossOriginConfiguration corsGet = cosClient.getBucketCrossOriginConfigurat
ion(bucketName);
List<CORSRule> corsRules = corsGet.getRules();
for (CORSRule rule : corsRules) {
```

版权所有：腾讯云计算（北京）有限责任公司

```
List<CORSRule.AllowedMethods> allowedMethods = rule.getAllowedMethods();
List<String> allowedHeaders = rule.getAllowedHeaders();
List<String> allowedOrigins = rule.getAllowedOrigins();
List<String> exposedHeaders = rule.getExposedHeaders();
int maxAgeSeconds = rule.getMaxAgeSeconds();
}
```

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String

返回结果说明

- 成功：返回存储桶的跨域规则。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。具体请参见 异常处理。

删除跨域配置

功能说明

删除指定存储桶的跨域名访问配置（DELETE Bucket cors）。

方法原型

```
public void deleteBucketCrossOriginConfiguration(String bucketName)
throws CosClientException, CosServiceException;
```

请求示例

```
//存储桶的命名格式为 BucketName-APPID
String bucketName = "examplebucket-1250000000";
cosClient.deleteBucketCrossOriginConfiguration(bucketName);
```

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。 具体请参见 异常处理。

自定义域名

最近更新时间：2022-05-09 14:27:13

简介

本文档提供关于自定义域名的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket domain	设置自定义域名	设置存储桶的自定义域名信息
GET Bucket domain	查询自定义域名	查询存储桶的自定义域名信息

设置自定义域名

功能说明

PUT Bucket domain 用于为存储桶配置自定义域名。

方法原型

```
public void setBucketDomainConfiguration(String bucketName, BucketDomainConfigura
tion configuration);
public void setBucketDomainConfiguration(SetBucketDomainConfigurationRequest setB
ucketDomainConfigurationRequest);
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见对象操作 -> 上传对象 -> 简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
BucketDomainConfiguration bucketDomainConfiguration = new BucketDomainConfigurati
on();
DomainRule domainRule = new DomainRule();
domainRule.setStatus(DomainRule.ENABLED);
domainRule.setType(DomainRule.REST);
domainRule.setName("qq.com");
domainRule.setForcedReplacement(DomainRule.CNAME);
```

```
bucketDomainConfiguration.getDomainRules().add(domainRule);
cosClient.setBucketDomainConfiguration(bucketName, bucketDomainConfiguration);
```

参数说明

参数名称	描述	类型
setBucketDomainConfigurationRequest	设置存储桶自定义域名请求	SetBucketDomainConfigurationRequest

Request 成员说明 ：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	设置自定义域名的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
configuration	构造函数或 set 方法	存储桶的自定义域名配置	BucketDomainConfiguration

BucketDomainConfiguration 成员说明:

参数名称	描述	类型
domainRules	存储桶的自定义域名配置	List<DomainRule> domainRules

DomainRule 成员说明：

参数名称	描述	类型
name	自定义域名	String
status	域名上线状态，可选值有 ENABLED、DISABLED	String
type	绑定的源站类型，可选值有 REST、WEBSITE	String
forcedReplacement	强制覆盖已存在的配置，可选值有 CNAME、TXT	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

版权所有：腾讯云计算（北京）有限责任公司

返回错误码说明

该请求可能会发生的一些常见的特殊错误如下：

状态码	说明
HTTP 409 Conflict	该域名记录已存在，且请求中没有设置强制覆盖。或者该域名记录不存在，且请求中设置了强制覆盖
HTTP 451 Unavailable For Legal Reasons	该域名是中国境内域名，并且没有备案

查询自定义域名

功能说明

GET Bucket domain 用于查询存储桶的自定义域名信息。

方法原型

```
public BucketDomainConfiguration getBucketDomainConfiguration(String bucketName)
throws CosClientException, CosServiceException;
```

请求示例

```
// 调用 COS 接口之前必须保证本进程存在一个 COSClient 实例，如果没有则创建
// 详细代码参见对象操作 -> 上传对象 -> 简单操作 -> 创建 COSClient
COSClient cosClient = createCOSClient();
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
BucketDomainConfiguration bucketDomainConfiguration = cosClient.getBucketDomainCo
nfiguration(bucketName);
String domainTxtVerification = bucketDomainConfiguration1.getDomainTxtVerificatio
n();
System.out.println(domainTxtVerification);
for (DomainRule rule : bucketDomainConfiguration.getDomainRules()) {
System.out.println(rule.getName());
System.out.println(rule.getStatus());
System.out.println(rule.getType());
System.out.println(rule.getClass());
}
```

参数说明

参数名称	描述	类型
bucketName	查询自定义域名的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String

返回结果说明

- 成功：返回 BucketDomainConfiguration，包含存储桶的自定义域名设置信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

返回参数说明

参数名称	描述	类型
domainTxtVerification	域名校验信息，该字段是一个 MD5 校验值，原串格式为：`cos[Region][BucketName-APPID][BucketCreateTime]`，其中 Region 为存储桶所在地域，BucketCreateTime 为存储桶 GMT 创建时间	String

访问控制

最近更新时间：2021-11-11 14:30:25

简介

本文档提供关于存储桶、对象的访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

存储桶 ACL

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶访问权限控制列表
GET Bucket acl	查询存储桶 ACL	查询存储桶的访问控制列表

对象 ACL

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表
GET Object acl	查询对象 ACL	查询对象的访问控制列表

存储桶 ACL

设置存储桶 ACL

功能说明

设置指定存储桶的访问权限控制列表（PUT Bucket acl）。该操作是覆盖操作，会覆盖已有的权限设置。ACL 包括预定义权限策略（CannedAccessControlList）或者自定义的权限控制（AccessControlList）。两类权限当同时设置时将忽略预定义策略，以自定义策略为主。

方法原型

```
// 方法 1（设置自定义策略）
public void setBucketAcl(String bucketName, AccessControlList acl)
throws CosClientException, CosServiceException;
// 方法 2（设置预定义策略）
public void setBucketAcl(String bucketName, CannedAccessControlList acl) throws C
osClientException, CosServiceException;
```

```
// 方法 3（以上两种方法的封装，包含两种策略设置，如果同时设置以自定定义策略为主）
public void setBucketAcl(SetBucketAclRequest setBucketAclRequest)
throws CosClientException, CosServiceException;
```

请求示例

```
// bucket的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 设置自定义 ACL
AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
// uin 可以在控制台查看到： https://console.intl.cloud.tencent.com/developer
owner.setId("qcs::cam::uin/100000000001:uin/100000000001");
acl.setOwner(owner);
String id = "qcs::cam::uin/100000000001:uin/100000000001";
UinGrantee uinGrantee = new UinGrantee("qcs::cam::uin/100000000001:uin/100000000001");
uinGrantee.setIdentifier(id);
acl.grantPermission(uinGrantee, Permission.FullControl);
cosClient.setBucketAcl(bucketName, acl);
// 设置预定义 ACL
// 设置私有读写（默认新建的 bucket 都是私有读写）
cosClient.setBucketAcl(bucketName, CannedAccessControlList.Private);
// 设置公有读私有写
cosClient.setBucketAcl(bucketName, CannedAccessControlList.PublicRead);
// 设置公有读写
cosClient.setBucketAcl(bucketName, CannedAccessControlList.PublicReadWrite);
```

参数说明

方法3参数同时包含1和2，因此以方法3为例进行介绍。

参数名称	描述	类型
setBucketAclRequest	权限设置请求类	SetBucketAclRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String

版权所有：腾讯云计算（北京）有限责任公司

Request 成员	设置方法	描述	类型
acl	构造函数或 set 方法	自定义权限策略	AccessControlList
cannedAcl	构造函数或 set 方法	预定义策略如公有读、公有读写、私有读	CannedAccessControlList

成员名	描述	类型
List&dxlt;Grant>	包含所有要授权的信息	数组
owner	表示 Object 或者 Owner 的拥有者	Owner 类

Grant 类成员说明：

成员名	描述	类型
grantee	被授权人的身份信息	Grantee
permission	被授权的权限信息（如可读，可写，可读可写）	Permission

Owner 类成员说明：

成员名	描述	类型
id	拥有者的身份信息	String
displayname	拥有者的名字（目前和 id 相同）	String

CannedAccessControlList 表示预设的策略，针对的是所有人。是一个枚举类，枚举值如下所示：

枚举值	描述
Private	私有读写（仅有 owner 可以读写）
PublicRead	公有读私有写（ owner 可以读写， 其他客户可以读）
PublicReadWrite	公有读写（即所有人都可以读写）

返回结果说明

- 成功：无返回值。

版权所有：腾讯云计算（北京）有限责任公司

- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询存储桶 ACL

功能说明

查询指定存储桶的访问权限控制列表。

方法原型

```
public AccessControlList getBucketAcl(String bucketName)
throws CosClientException, CosServiceException
```

请求示例

```
// bucket的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
AccessControlList accessControlList = cosClient.getBucketAcl(bucketName);
// 将存储桶权限转换为预设 ACL，可选值为：Private，PublicRead，PublicReadWrite
CannedAccessControlList cannedAccessControlList = accessControlList.getCannedAcce
ssControl();
```

参数说明

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID，详情请参见 命名规范	String

返回结果说明

- 成功：返回一个 Bucket 的 ACL。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

对象 ACL

设置对象 ACL

功能说明

设置存储桶中某个对象的访问控制列表。

ACL 包括预定义权限策略（CannedAccessControlList）或者自定义的权限控制（AccessControlList）。两类权限当同时设置时将忽略预定义策略，以自定义策略为主。

方法原型

```
// 方法1 (设置自定义策略)
public void setObjectAcl(String bucketName, String key, AccessControlList acl)
throws CosClientException, CosServiceException
// 方法2 (设置预定义策略)
public void setObjectAcl(String bucketName, String key, CannedAccessControlList a
cl)
throws CosClientException, CosServiceException
// 方法3 (以上两种方法的封装, 包含两种策略设置, 如果同时设置以自定定义策略为主)
public void setObjectAcl(SetObjectAclRequest setObjectAclRequest)
throws CosClientException, CosServiceException;
```

请求示例

```
// 权限信息中身份信息有格式要求, 对于主账号与子账号的范式如下：
// uin 可以在控制台查看到： https://console.intl.cloud.tencent.com/developer
// 下面的 root_uin 和 sub_uin 都必须是有效的 UIN 号
// 主账号 qcs::cam::uin/<root_uin>:uin/<root_uin> 表示授予主账号 root_uin 这个用户（即前后填的 uin 一样）
// 如 qcs::cam::uin/100000000001:uin/100000000001
// 子账号 qcs::cam::uin/<root_uin>:uin/<sub_uin> 表示授予 root_uin 的子账号 sub_uin 这个客户
// 如 qcs::cam::uin/100000000001:uin/100000000011
// 存储桶的命名格式为 BucketName-APPID
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
// 设置自定义 ACL
AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
// 设置 owner 的信息, owner 只能是主账号
owner.setId("qcs::cam::uin/100000000001:uin/100000000001");
acl.setOwner(owner);
// 授权给主账号100000000001可读可写权限
UinGrantee uinGrantee1 = new UinGrantee("qcs::cam::uin/100000000001:uin/100000000
001");
acl.grantPermission(uinGrantee1, Permission.FullControl);
cosClient.setObjectAcl(bucketName, key, acl);
// 设置预定义 ACL
// 设置私有读写（Object 的权限默认集成 Bucket的）
cosClient.setObjectAcl(bucketName, key, CannedAccessControlList.Private);
// 设置公有读私有写
cosClient.setObjectAcl(bucketName, key, CannedAccessControlList.PublicRead);
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 设置公有读写
cosClient.setObjectAcl(bucketName, key, CannedAccessControlList.PublicReadWrite);
```

参数说明

- **方法3** 参数同时包含1和2，因此以方法3为例进行介绍。

参数名称	描述	类型
SetObjectAclRequest	请求类	setObjectAclRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
acl	构造函数或 set 方法	自定义权限策略	AccessControlList
cannedAcl	构造函数或 set 方法	预定义策略如公有读、公有读写、私有读	CannedAccessControlList

成员名	描述	类型
List&dxlt;Grant>	包含所有要授权的信息	数组
owner	表示 Object 或者 Owner 的拥有者	Owner 类

Grant 类成员说明：

成员名	描述	类型

版权所有：腾讯云计算（北京）有限责任公司

成员名	描述	类型
grantee	被授权人的身份信息	Grantee
permission	被授权的权限信息（例如可读、可写、可读可写）	Permission

Owner 类成员说明：

成员名	描述	类型
id	拥有者的身份信息	String
displayname	拥有者的名字（目前和 ID 相同）	String

CannedAccessControlList 表示预设的策略，针对的是所有人。是一个枚举类，枚举值如下所示：

枚举值	描述
Private	私有读写（仅有 owner 可以读写）
PublicRead	公有读私有写（ owner 可以读写， 其他客户可以读）
PublicReadWrite	公有读写（即所有人都可以读写）

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询对象 ACL

功能说明

查询对象的访问控制列表。

方法原型

```
public AccessControlList getObjectAcl(String bucketName, String key)
throws CosClientException, CosServiceException;
```

请求示例

```
// 存储桶的命名格式为 BucketName-APPID
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
AccessControlList accessControlList = cosClient.getObjectAcl(bucketName, key);
// 将文件权限转换为预设 ACL，可选值为：Private, PublicRead, Default
CannedAccessControlList cannedAccessControlList = accessControlList.getCannedAcce
ssControl();
```

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String

返回结果说明

- 成功：返回一个 Object 所在的 ACL。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

图片处理
基础图片处理

最近更新时间：2023-02-23 16:33:22

简介

腾讯云对象存储（Cloud Object Storage，COS）集成了 数据万象（Cloud Infinite，CI）专业的一体化多媒体解决方案，涵盖以下图片处理功能，详情可见 图片处理概述。

服务	功能	说明
基础图片处理服务	缩放	等比缩放、设定目标宽高缩放等多种方式
	裁剪	普通裁剪、缩放裁剪、内切圆、人脸智能裁剪
	旋转	自适应旋转、普通旋转
	格式转换	格式转换、GIF 格式优化、渐进显示
	质量变换	针对 JPG 和 WEBP 图片进行质量变换
	高斯模糊	对图片进行模糊处理
	锐化	对图片进行锐化处理
	添加水印	图片水印、文字水印
	获取图片信息	基本信息、EXIF 信息、主色调
	去除元信息	包括 EXIF 信息
	快速缩略模板	快速实现图片格式转换、缩略、剪裁等功能，生成缩略图

基础图片处理

进行基础图片处理时，需要给定图片处理规则 rule，详情可见 图片处理概述。
所有示例代码请参见 SDK 基础图片处理示例。
部分示例如下：

缩放

```
String bucketName = "examplebucket-1250000000";
String key = "qrcode.png";
GetObjectRequest getObj = new GetObjectRequest(bucketName, key);
// 宽高缩放为原图的50%
String rule = "imageMogr2/thumbnail/!50p";
getObj.putCustomQueryParameter(rule, null);
cosClient.getObject(getObj, new File("qrcode-50p.png"));
```

裁剪

```
String bucketName = "examplebucket-1250000000";
String key = "qrcode.png";
GetObjectRequest getObj = new GetObjectRequest(bucketName, key);
// 裁剪内切圆的半径，取值范围为大于0且小于原图最小边一半的整数。内切圆的圆心为图片的中心
String rule = "imageMogr2/iradius/150";
getObj.putCustomQueryParameter(rule, null);
cosClient.getObject(getObj, new File("qrcode-cropping.png"));
```

旋转

```
String bucketName = "examplebucket-1250000000";
String key = "qrcode.png";
GetObjectRequest getObj = new GetObjectRequest(bucketName, key);
// 顺时针旋转90度
String rule = "imageMogr2/rotate/90";
getObj.putCustomQueryParameter(rule, null);
cosClient.getObject(getObj, new File("qrcode-rotate.png"));
```

获取主色调

```
String bucketName = "examplebucket-1250000000";
String key = "qrcode.png";
GetObjectRequest getObj = new GetObjectRequest(bucketName, key);
// 获取图片主色调
String rule = "imageAve";
COSObject object = cosClient.getObject(getObj);
COSObjectInputStream objectContent = object.getObjectContent();
```

获取图片基本信息

```
String bucketName = "examplebucket-1250000000";
String key = "qrcode.png";
```

版权所有：腾讯云计算（北京）有限责任公司

```
GetObjectRequest getObj = new GetObjectRequest(bucketName, key);
// 图片基本信息
String rule = "imageInfo";
getObj.putCustomQueryParameter(rule, null);
COSObject object = cosClient.getObject(getObj);
COSObjectInputStream objectContent = object.getObjectContent();
```

版权所有：腾讯云计算（北京）有限责任公司

图片持久化处理

最近更新时间：2021-07-12 15:52:30

简介

下面示例展示了如何在上传图片时自动实现图片处理。

图片上传完成后，对象存储（Cloud Object Storage，COS）会存储原始图片和已处理过的图片。后续用户可以通过普通的下载请求获取处理结果。

上传时图片持久化处理

示例代码

```
// bucket名需包含appid
// api 请参考 https://cloud.tencent.com/document/product/436/54050
String bucketName = "examplebucket-1250000000";
String key = "test.jpg";
File localFile = new File("E://test.jpg");
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFi
le);
PicOperations picOperations = new PicOperations();
picOperations.setIsPicInfo(1);
// 添加图片处理规则
List<PicOperations.Rule> ruleList = new LinkedList<>();
PicOperations.Rule rule1 = new PicOperations.Rule();
rule1.setBucket(bucketName);
rule1.setFileId("test-1.jpg");
rule1.setRule("imageMogr2/rotate/90");
ruleList.add(rule1);
PicOperations.Rule rule2 = new PicOperations.Rule();
rule2.setBucket(bucketName);
rule2.setFileId("test-2.jpg");
rule2.setRule("imageMogr2/rotate/180");
ruleList.add(rule2);
picOperations.setRules(ruleList);
putObjectRequest.setPicOperations(picOperations);
try {
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
CIUploadResult ciUploadResult = putObjectResult.getCiUploadResult();
System.out.println(putObjectResult.getRequestId());
```

版权所有：腾讯云计算（北京）有限责任公司

```
System.out.println(ciUploadResult.getOriginalInfo().getEtag());
for(CIObject ciObject:ciUploadResult.getProcessResults().getObjectList()) {
System.out.println(ciObject.getLocation());
System.out.println(ciObject.getEtag());
}
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
```

参数说明

PicOperations 类用于记录图像操作，其主要成员说明如下：

成员名称	描述	类型
isPicInfo	是否返回原图信息，0不返回原图信息，1返回原图信息，默认为0	int
rules	处理规则，一条规则对应一个处理结果（目前支持五条规则），不填则不进行图片处理	List

返回参数说明

CIUploadResult 类用于返回图片处理结果信息，其主要成员说明如下：

成员名称	描述	类型
originalInfo	原图信息	OriginalInfo
processResults	图片处理结果	ProcessResults

OriginalInfo 类用于记录原图信息，其主要成员说明如下：

成员名称	描述	类型
key	原图文件名	String
location	图片路径	String
imageInfo	原图图片信息	ImageInfo
etag	原图 ETag 信息（若处理结果图覆盖原图则为结果图 ETag 信息）	String

ImageInfo 类用于记录原图图片信息，其主要成员说明如下：

版权所有：腾讯云计算（北京）有限责任公司

成员名称	描述	类型
format	格式	String
width	图片宽度	Integer
height	图片高度	Integer
quality	图片质量	Integer
ave	图片主色调	String
orientation	图片旋转角度	Integer

ProcessResults 类用于记录图片处理结果，其主要成员说明如下：

成员名称	描述	类型
objectList	每一个图片处理结果	List

CIObject 类用于记录一个图片处理结果，其主要成员说明如下：

成员名称	描述	类型
key	文件名	String
location	图片路径	String
format	图片格式	String
width	图片宽度	Integer
height	图片高度	Integer
size	图片大小	Integer
quality	图片质量	Integer
etag	处理结果图 ETag 信息	String

对云上数据进行图片处理

下面示例展示了如何对已存储在 COS 的图片进行相应处理操作，并将结果存入到 COS。

版权所有：腾讯云计算（北京）有限责任公司

示例代码

```
String bucketName = "examplebucket-1250000000";
String key = "test.jpg";
ImageProcessRequest imageReq = new ImageProcessRequest(bucketName, key);
PicOperations picOperations = new PicOperations();
picOperations.setIsPicInfo(1);
List<PicOperations.Rule> ruleList = new LinkedList<>();
PicOperations.Rule rule1 = new PicOperations.Rule();
rule1.setBucket(bucketName);
rule1.setFileId("test-1.jpg");
rule1.setRule("imageMogr2/rotate/90");
ruleList.add(rule1);
PicOperations.Rule rule2 = new PicOperations.Rule();
rule2.setBucket(bucketName);
rule2.setFileId("test-2.jpg");
rule2.setRule("imageMogr2/rotate/180");
ruleList.add(rule2);
picOperations.setRules(ruleList);
imageReq.setPicOperations(picOperations);
CIUploadResult result = cosClient.processImage(imageReq);
result.getProcessResults();
result.getOriginalInfo();
try {
CIUploadResult ciUploadResult = cosClient.processImage(imageReq);
System.out.println(ciUploadResult.getOriginalInfo().getEtag());
for(CIObject ciObject:ciUploadResult.getProcessResults().getObjectList()) {
System.out.println(ciObject.getLocation());
System.out.println(ciObject.getEtag());
}
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
```

版权所有：腾讯云计算（北京）有限责任公司

图片高级压缩

最近更新时间：2021-05-27 16:07:46

简介

本文档提供关于图片高级压缩的 API 概览以及 SDK 示例代码。

API	操作描述
图片高级压缩	图片高级压缩可以更加高效地将图片转换为 TPG 或 HEIF 等高压缩比格式，有效降低图片传输链路及加载耗时，降低带宽及流量成本

示例代码

```
GetObjectRequest getObj = new GetObjectRequest(bucketName, key);
// 这里是图片压缩参数，具体请参考数据万象API，这里仅是示例
String compress = "imageMogr2/format/tpg";
getObj.putCustomQueryParameter(compress, null);
```

版权所有：腾讯云计算（北京）有限责任公司

内容审核
图片审核

最近更新时间：2022-12-14 15:21:11

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于图片审核的 API 概览和 SDK 示例代码。

API	操作描述
图片同步审核	对对象存储（Cloud Object Storage，COS）存量数据进行涉黄、违法违规以及广告引导类图片的扫描
图片批量审核	对多个图片进行批量审核

图片同步审核

功能说明

图片审核的存量扫描功能通过借助数据万象的持久化处理接口，实现对 COS 存量数据的涉黄、违法违规以及广告引导类图片的扫描。

方法原型

```
ImageAuditingResponse imageAuditing(ImageAuditingRequest request);
```

请求示例

```
//1.创建任务请求对象
ImageAuditingRequest request = new ImageAuditingRequest();
//2.添加请求参数 参数详情请见api接口文档
//2.1设置请求bucket
request.setBucketName("examplebucket-1250000000");
//2.2设置审核类型
request.setDetectType("porn,ads");
//2.3设置bucket中的图片位置
request.setObjectKey("1.png");
//3.调用接口,获取任务响应对象
ImageAuditingResponse response = client.imageAuditing(request);
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	描述	类型	是否必选
ObjectKey	COS 存储桶中的图片文件名称，COS 存储桶由 `Host` 指定，例如在北京的 examplebucket-1250000000存储桶中的目录 test 下的文件 img.jpg，则 `Host` 填写 examplebucket-1250000000.cos.ap-beijing.myqcloud.com，`ObjectKey` 填写 test/img.jpg。	String	否
bizType	表示审核策略的唯一标识，您可以通过控制台上的审核策略页面，配置您希望审核的场景，如涉黄、广告、违法违规等，配置指引：设置审核策略。您可以在控制台上获取到 bizType。bizType 填写时，此条审核请求将按照该审核策略中配置的场景进行审核。bizType 不填写时，将自动使用默认的审核策略。	String	否
detectType	（后续不再维护）审核的场景类型，有效值：Porn（涉黄）、Ads（广告）等，可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 bizType 参数。	String	否
detectUrl	您可以通过填写 `detectUrl` 审核任意公网可访问的图片链接不填写 `detectUrl` 时，后台会默认审核 `ObjectKey` 填写了 `detectUrl` 时，后台会审核 `detectUrl` 链接，无需再填写 `ObjectKey``detectUrl` 示例：http://www.example.com/abc.jpg。	String	否

节点名称（关键字）	描述	类型	是否必选
interval	审核 GIF 动图时，可使用该参数进行截帧配置，代表截帧的间隔。例如值设为5，则表示从第1帧开始截取，每隔5帧截取一帧，默认值5。	Int	否
maxFrames	针对 GIF 动图审核的最大截帧数量，需大于0。例如值设为5，则表示最大截取5帧，默认值为5。	Int	否
largeImageDetect	对于超过大小限制的图片是否进行压缩后再审核，取值为：0（不压缩），1（压缩）。默认为0。注：压缩最大支持32MB的图片，且会收取图片压缩费用。对于 GIF 等动态图过大时，压缩时间较长，可能会导致审核超时失败。	Int	否
dataId	图片标识，该字段在结果中返回原始内容，长度限制为512字节。	String	否
async	是否异步进行审核，取值 0：同步返回结果，1：异步进行审核，默认为0。	Int	否
callback	审核结果（Detail版本）以回调形式发送至您的回调地址，异步审核时生效，支持以 `http://` 或者 `https://` 开头的地址，例如：`http://www.callback.com`。	String	否

返回结果说明

- 成功：成功则返回 ImageAuditingResponse 实例，内含审核结果内容。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见异常处理。

ImageAuditingResponse对象转为json后内容展示


```
{
"requestId":"NjJkOTI3YTJfNGNhY2ZhMDlfOWJhOV9hZ*****",
"jobId":"si461d9f9608de11ed9b3d525400b*****",
"object":"1.png",
"compressionResult":"0",
"result":"0",
"label":"Normal",
"category":"",
"subLabel":"",
"score":"42",
"text":"",
"state":"Success",
"pornInfo":{
```

```
"code":"0",
"msg":"OK",
"hitFlag":"0",
"score":"0",
"label":"",
"ocrResults":{
"text":"",
"keywords":""
},
"category":""
}
}
```


响应包体具体数据内容如下：

参数名称	类型	描述
DataId	String	图片标识，审核结果会返回原始内容，长度限制为512字节。
JobId	String	图片审核任务的 ID。
State	String	审核任务的状态，值为 Success（审核成功）。
Object	String	存储在 COS 桶中的图片名称，创建任务使用 ObjectKey 时返回。
Url	String	图片文件的链接地址，创建任务使用 detect-url 时返回。
CompressionResult	Int	图片是否被压缩处理，值为 0（未压缩），1（正常压缩）。
Result	Int	该字段表示本次判定的审核结果，您可以根据该结果，进行后续的操作；建议您按照业务所需，对不同的审核结果进行相应处理。有效值：**0**（审核正常），**1**（判定为违规敏感文件），**2**（疑似敏感，建议人工复核）。
Label	String	该字段用于返回检测结果中所对应的**优先级最高的恶意标签**，表示模型推荐的审核结果，建议您按照业务所需，对不同违规类型与建议值进行处理。返回值：**Normal** 表示正常，**Porn** 表示色情，**Ads** 表示广告，以及其他不安全或不适宜的类型。
Category	String	该字段为 `Label` 的子集，表示审核命中的具体审核类别。例如 Sexy，表示色情标签中的性感类别。
SubLabel	String	该图命中的二级标签结果。
Score	Int	该字段表示审核结果命中审核信息的置信度，取值范围：0（**置信度最低**）-100（**置信度最高**），越高代表该内容越有可能属于当前返回审核信息例如：色情 99，则表明该内容非常有可能属于色情内容。

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述
Text	String	该图里的文字内容（OCR），当审核策略开启文本内容检测时返回。
PornInfo	Container	审核场景为**涉黄**的审核结果信息。
AdsInfo	Container	审核场景为**广告引导**的审核结果信息。

审核信息（PornInfo、AdsInfo）中包含如下内容：

参数名称	类型	描述
Code	Int	错误码，0为正确，其他数字对应相应错误。详情请参见 错误码。
Msg	String	具体错误信息，如正常则为 OK。
HitFlag	Int	用于返回该审核场景的审核结果，返回值：0：正常。1：确认为当前场景的违规内容。2：疑似为当前场景的违规内容。
Score	Int	该字段表示审核结果命中审核信息的置信度，取值范围：0（**置信度最低**）-100（**置信度最高**），越高代表该内容越有可能属于当前返回审核信息。其中0 - 60分表示图片正常，61 - 90分表示图片疑似敏感，91 - 100分表示图片确定敏感。例如：色情 99，表明该内容非常有可能属于色情内容。
Label	String	该字段表示该截图的综合结果标签（可能为 SubLabel，可能为人物名字等）。
Category	String	该字段为 `Label` 的子集，表示审核命中的具体审核类别。例如 Sexy，表示色情标签中的性感类别。
SubLabel	String	该字段表示审核命中的具体子标签，例如：Porn 下的 SexBehavior 子标签。 注意：该字段可能返回空，表示未命中具体的子标签。
OcrResults	Container Array	该字段表示 OCR 文本识别的详细检测结果，包括文本识别结果、命中的关键词等信息，有相关违规内容时返回。
LibResults	Container Array	该字段用于返回基于风险库识别的结果。 注意：未命中风险库中样本时，此字段不返回。

Container 节点 LibResults 的内容：

参数名称	类型	描述
ImageId	String	该字段表示命中的风险库中的图片样本 ID。

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述
Score	String	该字段用于返回当前标签下的置信度，取值范围：0（置信度最低）-100（置信度最高），越高代表当前的图片越有可能命中库中的样本。例如：色情 99，表明该数据非常有可能命中库中的色情样本。

Container 节点 OcrResults 的内容：

参数名称	类型	描述
Text	String	图片 OCR 文本识别出的具体文本内容。
Keywords	Array	在当前审核场景下命中的关键词。
Location	Container	该参数用于返回 OCR 检测框在图片中的位置（左上角 xy 坐标、长宽、旋转角度），以方便快速定位识别文字的相关信息。

名称	类型	描述
X	Float	该参数用于返回检测框**左上角位置的横坐标**（x）所在的像素位置，结合剩余参数可唯一确定检测框的大小和位置。
Y	Float	该参数用于返回检测框**左上角位置的纵坐标**（y）所在的像素位置，结合剩余参数可唯一确定检测框的大小和位置。
Width	Float	该参数用于返回**检测框的宽度**（由左上角出发在 x 轴向右延伸的长度），结合剩余参数可唯一确定检测框的大小和位置。
Height	Float	该参数用于返回**检测框的高度**（由左上角出发在 y 轴向下延伸的长度），结合剩余参数可唯一确定检测框的大小和位置。
Rotate	Float	该参数用于返回**检测框的旋转角度**，该参数结合 X 和 Y 两个坐标参数可唯一确定检测框的具体位置；取值：**0-360（角度制）**，方向为**逆时针旋转**。

图片批量审核

功能说明

图片批量审核接口为同步请求方式，您可以通过本接口对多个图片文件进行内容审核。该接口属于 POST 请求。

方法原型

```
BatchImageAuditingResponse batchImageAuditing(BatchImageAuditingRequest request);
```

请求示例

```
//1.创建任务请求对象
BatchImageAuditingRequest request = new BatchImageAuditingRequest();
//2.添加请求参数 参数详情请见api接口文档
//2.1设置请求bucket
request.setBucketName("examplebucket-1250000000");
//2.2添加请求内容
List<BatchImageAuditingInputObject> inputList = request.getInputList();
BatchImageAuditingInputObject input = new BatchImageAuditingInputObject();
input.setObject("1.jpg");
input.setDataId("DataId");
inputList.add(input);
input = new BatchImageAuditingInputObject();
input.setUrl("https://examplebucket-1250000000.cos.ap-chongqing.myqcloud.com/1.png");
input.setDataId("DataId");
inputList.add(input);
//2.2设置审核类型
request.getConf().setDetectType("all");
//3.调用接口,获取任务响应对象
BatchImageAuditingResponse response = client.batchImageAuditing(request);
List<BatchImageJobDetail> jobList = response.getJobList();
```

参数说明

request 的数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Request	无	图片批量审核的具体配置项。	Container	是

Container 类型 Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的内容，如有多个图片，请传入多个 Input 结构。	Container Array	是
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下，使用其中一种：

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	存储在 COS 存储桶中的图片文件名称，例如在目录 test 中的文件 image.jpg，则文件名称为 test/image.jpg。Object 和 Url 只能选择其中一种。	String	否
Url	Request.Input	图片文件的链接地址，例如 http://a-1250000.cos.ap-shanghai.myqcloud.com/image.jpg。Object 和 Url 只能选择其中一种。	String	否
Interval	Request.Input	截帧频率，GIF 图检测专用，默认值为5，表示从第一帧（包含）开始每隔5帧截取一帧。	Int	否
MaxFrames	Request.Input	最大截帧数量，GIF 图检测专用，默认值为5，表示只截取 GIF 的5帧图片进行审核，必须大于0。	Int	否
DataId	Request.Input	图片标识，该字段在结果中返回原始内容，长度限制为512字节。	String	否
LargeImageDetect	Request.Input	对于超过大小限制的图片是否进行压缩后再审核，取值为：0（不压缩），1（压缩）。默认为0。 注：压缩最大支持32M的图片，且会收取压缩费用。	Int	否
UserInfo	Request.Input	用户业务字段。	Container	否

Container 节点 UserInfo 的内容：

节点名称（关键字）	描述	类型	是否必选
TokenId	一般用于表示账号信息，长度不超过128字节。	String	否
Nickname	一般用于表示昵称信息，长度不超过128字节。	String	否
DeviceId	一般用于表示设备信息，长度不超过128字节。	String	否
AppId	一般用于表示 App 的唯一标识，长度不超过128字节。	String	否
Room	一般用于表示房间号信息，长度不超过128字节。	String	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	描述	类型	是否必选
IP	一般用于表示 IP 地址信息，长度不超过128字节。	String	否
Type	一般用于表示业务类型，长度不超过128字节。	String	否
ReceiveTokenId	一般用于表示接收消息的用户账号，长度不超过128字节。	String	否
Gender	一般用于表示性别信息，长度不超过128字节。	String	否
Level	一般用于表示等级信息，长度不超过128字节。	String	否
Role	一般用于表示角色信息，长度不超过128字节。	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
BizType	Request.Conf	表示审核策略的唯一标识，您可以通过控制台上的审核策略页面，配置您希望审核的场景，如涉黄、广告、违法违规等，配置指引：设置审核策略。您可以在控制台上获取到 BizType。BizType 填写时，此条审核请求将按照该审核策略中配置的场景进行审核。BizType 不填写时，将自动使用默认的审核策略。	String	否
DetectType	Request.Conf	（后续不再维护）审核的场景类型，有效值：Porn（涉黄）、Ads（广告）等，可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。	String	否
Async	Request.Conf	是否异步进行审核，0：同步返回结果，1：异步进行审核。默认值为 0。	Integer	否
Callback	Request.Conf	审核结果（Detail版本）以回调形式发送至您的回调地址，异步审核时生效，支持以 `http://` 或者 `https://` 开头的地址，例如：`http://www.callback.com` 。	String	否

注意：

- 通过 Object 进行审核为内网操作，不会产生额外的外网流量。
- 通过 Url 进行审核，会产生图片所在源站对应的外网流量。

版权所有：腾讯云计算（北京）有限责任公司

返回结果说明

- 成功：成功则返回 BatchImageAuditingResponse 实例，内含审核结果内容。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见异常处理。

ImageAuditingResponse对象转为json后内容展示


```
{
"requestId":"NjJkOTJjMjVfMTIwNjUzMDlfNDg5OF8*****",
"jobList":[
{
"dataId":"DataId",
"jobId":"sif6423b8008e011ed9b3d525400*****",
"category":"",
"label":"Normal",
"result":"0",
"object":"1.jpg",
"score":"0",
"subLabel":"",
"text":"",
"code":"",
"message":"",
"url":"",
"compressionResult":"0",
"pornInfo":{
"code":"0",
"msg":"OK",
"hitFlag":"0",
"score":"45",
"label":"",
"keywords":"",
"count":"",
"subLabel":"",
"ocrResults":{
"text":"",
"keywords":""
},
"category":""
},
"adsInfo":{
"code":"0",
"msg":"OK",
"hitFlag":"0",
"score":"0",
```

```
"label":"",
"keywords":"",
"count":"",
"subLabel":"",
"ocrResults":{
"text":"",
"keywords":""
},
"category":""
},
"userInfo":{
"tokenId":"",
"nickname":"",
"deviceId":"",
"appId":"",
"room":"",
"ip":"",
"type":"",
"receiveTokenId":"",
"gender":"",
"level":"",
"role":""
},
"ocrResults":{
"text":"",
"keywords":""
}
},
{
"dataId":"DataId",
"jobId":"sif640a3c108e011ed9b3d525400*****",
"category":"",
"label":"Normal",
"result":"0",
"object":"",
"score":"42",
"subLabel":"",
"text":"",
"code":"",
"message":"",
"url":"https://demo-1234567890.cos.ap-chongqing.myqcloud.com/1.png",
"compressionResult":"0",
"pornInfo":{
"code":"0",
"msg":"OK",
"hitFlag":"0",
"score":"0",
```

```
"label":"",
"keywords":"",
"count":"",
"subLabel":"",
"category":"",
"ocrResults":{
"text":"",
"keywords":""
}
},
"adsInfo":{
"code":"0",
"msg":"OK",
"hitFlag":"0",
"score":"0",
"label":"",
"keywords":"",
"count":"",
"subLabel":"",
"category":"",
"ocrResults":{
"text":"",
"keywords":""
}
},
"userInfo":{
"tokenId":"",
"nickname":"",
"deviceId":"",
"appId":"",
"room":"",
"ip":"",
"type":"",
"receiveTokenId":"",
"gender":"",
"level":"",
"role":""
},
"ocrResults":{
"text":"",
"keywords":""
}
}
]
}
```


响应包体具体数据内容如下：

Response 的内容：

参数名称	类型	描述
JobsDetail	Array	图片审核的结果
RequestId	String	每次请求发送时，服务端将会自动为请求生成一个 ID，遇到问题时，该 ID 能更快地协助定位问题。

JobsDetail 节点内容如下：

参数名称	类型	描述
Code	String	错误码，只有失败时返回。详情请查看 错误码列表。
Message	String	错误描述，只有失败时返回。
DataId	String	图片标识，审核结果会返回原始内容，长度限制为512字节。
JobId	String	本次审核任务的 ID。
State	String	审核任务的状态，值为Success（审核成功）、Failed（审核失败）其中一个。
Object	String	存储在 COS 存储桶中的图片文件名称，创建任务使用Object时返回。
Url	String	图片文件的链接地址，创建任务使用Url时返回。
CompressionResult	Integer	图片是否被压缩处理，值为 0（未压缩），1（正常压缩）。
Label	String	该字段用于返回检测结果中所对应的**优先级最高的恶意标签**，表示模型推荐的审核结果，建议您按照业务所需，对不同违规类型与建议值进行处理。 返回值：**Normal**：正常，**Porn**：色情，**Ads**：广告等。
Result	Integer	该字段表示本次判定的审核结果，您可以根据该结果，进行后续的操作；建议您按照业务所需，对不同的审核结果进行相应处理。有效值：**0**（审核正常），**1**（判定为违规敏感文件），**2**（疑似敏感，建议人工复核）。
Score	Integer	该字段表示审核结果命中审核信息的置信度，取值范围：0（**置信度最低**）-100（**置信度最高**），越高代表该内容越有可能属于当前返回审核信息。例如：色情 99，表明该内容非常有可能属于色情内容。
Category	String	该图命中的审核类别结果。
SubLabel	String	该图命中的二级标签结果。

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述
Text	String	该图里的文字内容（OCR），当审核策略开启文本内容检测时返回。
PornInfo	Container	审核场景为**涉黄**的审核结果信息。
AdsInfo	Container	审核场景为**广告引导**的审核结果信息。
UserInfo	Container	请求中设置的 UserInfo 原样返回。
ListInfo	Container	账号黑白名单结果。

审核信息（PornInfo、AdsInfo 等）中的内容如下：

参数名称	类型	描述
Code	Integer	单个审核场景的错误码，0为成功，其他为失败。详情请参见 错误码。
Msg	String	具体错误信息，如正常则为 OK。
HitFlag	Integer	用于返回该审核场景的审核结果，返回值：0：正常。1：确认为当前场景的违规内容。2：疑似为当前场景的违规内容。
Score	Integer	该字段表示审核结果命中审核信息的置信度，取值范围：0（**置信度最低**）-100（**置信度最高**），越高代表该内容越有可能属于当前返回审核信息例如：色情99，则表明该内容非常有可能属于色情内容。
Label	String	该图的结果标签（为综合标签，可能为 SubLabel，可能为人物名字等）。
Category	String	该字段为 `Label` 的子集，表示审核命中的具体审核类别。例如 Sexy，表示色情标签中的性感类别。
SubLabel	String	该图的二级标签结果。
OcrResults	Container Array	该字段表示 OCR 文本识别的详细检测结果，包括文本坐标信息、文本识别结果等信息，有相关违规内容时返回。
LibResults	Container Array	该字段用于返回基于风险库识别的结果。注意：未命中风险库中样本时，此字段不返回。

Container 节点 LibResults 的内容：

节点名称（关键字）	类型	描述

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	类型	描述
ImageId	String	该字段表示命中的风险库中的图片样本ID。
Score	Integer	该字段用于返回当前标签下的置信度，取值范围：0（置信度最低）-100（置信度最高），越高代表当前的图片越有可能命中库中的样本。例如：色情 99，则表明该数据非常有可能命中库中的色情样本。

OcrResults 的内容如下：

节点名称（关键字）	类型	描述
Text	String	有敏感信息的 Ocr 文本内容。
Keywords	String Array	该段内容中的敏感文字。

UserInfo 的内容（与请求中的 UserInfo 一致）：

节点名称（关键字）	描述	类型	是否必选
TokenId	一般用于表示账号信息，长度不超过128字节。	String	否
Nickname	一般用于表示昵称信息，长度不超过128字节。	String	否
DeviceId	一般用于表示设备信息，长度不超过128字节。	String	否
AppId	一般用于表示App的唯一标识，长度不超过128字节。	String	否
Room	一般用于表示房间号信息，长度不超过128字节。	String	否
IP	一般用于表示IP地址信息，长度不超过128字节。	String	否
Type	一般用于表示业务类型，长度不超过128字节。	String	否
ReceiveTokenId	一般用于表示接收消息的用户账号，长度不超过128字节。	String	否
Gender	一般用于表示性别信息，长度不超过128字节。	String	否
Level	一般用于表示等级信息，长度不超过128字节。	String	否
Role	一般用于表示角色信息，长度不超过128字节。	String	否

ListInfo 的内容：

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	类型	描述
ListResults	Container Array	命中的所有名单结果。

ListResults 的内容：

节点名称（关键字）	类型	描述
ListType	String	命中的名单类型，取值为0（白名单）和1（黑名单）。
ListName	String	命中的名单名称。
Entity	String	命中了名单中的哪条内容。

版权所有：腾讯云计算（北京）有限责任公司

视频审核

最近更新时间：2022-12-14 15:21:11

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于视频审核的 API 概览和 SDK 示例代码。

API	操作描述
提交视频审核任务	用于提交一个视频审核任务
查询视频审核任务结果	用于查询指定的视频审核任务

提交视频审核任务

功能说明

用于提交一个视频审核任务。

方法原型

```
VideoAuditingResponse createVideoAuditingJob(VideoAuditingRequest request);
```

请求示例

```
//1.创建任务请求对象
VideoAuditingRequest request = new VideoAuditingRequest();
//2.添加请求参数 参数详情请见api接口文档
```

```
request.setBucketName("examplebucket-1250000000");
request.getInput().setObject("1.mp4");
request.getConf().setDetectType("Porn,Ads");
request.getConf().getSnapshot().setCount("10");
request.getConf().getSnapshot().setMode("Interval");
request.getConf().getSnapshot().setTimeInterval("10");
//3.调用接口,获取任务响应对象
VideoAuditingResponse response = client.createVideoAuditingJob(request);
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的视频。	Container	是
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	当前 COS 存储桶中的视频文件名称，例如在目录 test 中的文件 video.mp4，则文件名称为 test/video.mp4。	String	是
Url	Request.Input	视频文件的链接地址，例如 http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/test.mp4。Object 和 Url 只能选择其中一种。	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。	String	是
Snapshot	Request.Conf	视频画面的审核通过视频截帧能力截取出一定量的截图，通过对截图逐一审核而实现的，该参数用于指定视频截帧的配置。	Container	是

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Callback	Request.Conf	回调地址，以 `http://` 或者 `https://` 开头的地址。	String	否
CallbackVersion	Request.Conf	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。	String	否
BizType	Request.Conf	审核策略，不带审核策略时使用默认策略。	String	否
DetectContent	Request.Conf	用于指定是否审核视频声音，当值为0时：表示只审核视频画面截图；值为1时：表示同时审核视频画面截图和视频声音。默认值为0。	Integer	否

Container 类型 Snapshot 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Mode	Request.Conf.Snapshot	截帧模式。Interval 表示间隔模式；Average 表示平均模式；Fps 表示固定帧率模式。 • Interval 模式：TimeInterval，Count 参数生效。当设置 Count，未设置 TimeInterval 时，表示截取所有帧，共 Count 张图片。 • Average 模式：Count 参数生效。表示整个视频，按平均间隔截取共 Count 张图片。 • Fps 模式：TimeInterval 表示每秒截取多少帧，Count 表示共截取多少帧。	String	否
Count	Request.Conf.Snapshot	视频截帧数量，范围为(0, 10000]。	String	否
TimeInterval	Request.Conf.Snapshot	视频截帧频率，范围为(0, 60]，单位为秒，支持 float 格式，执行精度精确到毫秒。	Float	否

返回结果说明

- 成功：成功则返回 VideoAuditingResponse 媒体审核任务结果对象。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见异常处理。

版权所有：腾讯云计算（北京）有限责任公司

查询视频审核任务结果

功能说明

用于查询视频审核任务执行情况和结果。

方法原型

```
VideoAuditingResponse describeAuditingJob(VideoAuditingRequest request);
```

请求示例

```
//1.创建任务请求对象
VideoAuditingRequest request = new VideoAuditingRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
request.setJobId("av81591b4bbd2211eb80235254008*****");
//3.调用接口,获取任务响应对象
VideoAuditingResponse response = client.describeAuditingJob(request);
```

参数说明

参数名称	描述	类型	是否必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
jobId	需要查询的任务 ID	String	是

返回结果说明

- 成功：返回 VideoAuditingResponse 审核任务结果对象。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

音频审核

最近更新时间：2022-12-15 16:00:32

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于音频审核的 API 概览和 SDK 示例代码。

API	操作描述
提交音频审核任务	用于提交一个音频审核任务
查询音频审核任务结果	用于查询指定的音频审核任务

提交音频审核任务

功能说明

用于提交一个音频审核任务。

方法原型

```
AudioAuditingResponse createAudioAuditingJob(AudioAuditingRequest request);
```

请求示例

```
//1.创建任务请求对象
AudioAuditingRequest request = new AudioAuditingRequest();
//2.添加请求参数 参数详情请见api接口文档
```

```
request.setBucketName("examplebucket-1250000000");
request.getInput().setObject("pron.mp3");
request.getConf().setDetectType("Porn,Ads");
request.getConf().setCallback("http://cloud.tencent.com/");
//3.调用接口,获取任务响应对象
AudioAuditingResponse response = client.createAudioAuditingJobs(request);
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的内容。	Container	是
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	存储在 COS 存储桶中的音频文件名称，例如在目录 test 中的文件audio.mp3，则文件名称为 test/audio.mp3。Object 和 Url 只能选择其中一种。	String	否
Url	Request.Input	音频文件的链接地址，例如 http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/audio.mp3。Object 和 Url 只能选择其中一种。	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
BizType	Request.Conf	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见设置公共审核策略。	String	否
DetectType	Request.Conf	（后续不再维护）审核的场景类型，有效值：Porn（涉黄）、Ads（广告）等，可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。如您有更多场景的审核需要，请使用 BizType 参数。	String	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Callback	Request.Conf	审核结果以回调形式发送至您的回调地址，支持以 `http://` 或者 `https://` 开头的地址，例如：`http://www.callback.com` 。	String	否
CallbackVersion	Request.Conf	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。	string	否

返回结果说明

- 成功：成功则返回 AudioAuditingResponse 音频审核任务结果对象。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见异常处理。

查询音频审核任务

功能说明

用于查询音频审核任务执行情况和结果。

方法原型

```
AudioAuditingResponse describeAuditingJob(AudioAuditingRequest request);
```

请求示例

```
//1.创建任务请求对象
AudioAuditingRequest request = new AudioAuditingRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
request.setJobId("sacbf7269cbd2e11eba5325254009*****");
//3.调用接口,获取任务响应对象
AudioAuditingResponse response = client.describeAudioAuditingJob(request);
```

参数说明

参数名称	描述	类型	是否必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID	String	是

返回结果说明

- 成功：返回 AudioAuditingResponse 审核任务结果对象，字段含义请参见 查询音频审核任务结果。
- 失败：发生错误（例如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

文档审核

最近更新时间：2022-12-14 15:21:11

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于文档审核的 API 概览和 SDK 示例代码。

API	操作描述
提交文档审核任务	用于提交一个文档审核任务
查询文档审核任务结果	用于查询指定的文档审核任务

提交文档审核任务

功能说明

用于提交一个文档审核任务。

方法原型

```
DocumentAuditingResponse createAuditingDocumentJobs(DocumentAuditingRequest reque
st);
```

请求示例

```
//1.创建任务请求对象
DocumentAuditingRequest request = new DocumentAuditingRequest();
```

```
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
//2.1.1设置对象地址
// request.getInput().setObject("1.txt");
//2.1.2或直接设置文档地址
request.getInput().setUrl("https://markjrzhang-1251704708.cos.ap-chongqing.myqcloud.com/%E9%97%AE%E9%A2%98%E6%B1%87%E6%80%BB.pptx");
//2.2设置审核类型参数
request.getConf().setDetectType("all");
//3.调用接口,获取任务响应对象
DocumentAuditingResponse response = client.createAuditingDocumentJobs(request);
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的内容	Container	是
Conf	Request	审核规则配置	Container	是

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Url	Request.Input	文档文件的链接地址，例如 http://www.example.com/doctest.doc	String	是
Type	Request.Input	指定文档文件的类型，如未指定则默认以文件的后缀为类型。 如果文件没有后缀，该字段必须指定，否则会审核失败。例如：doc、docx、ppt、pptx 等	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
BizType	Request.Conf	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置审核策略	String	否

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads	String	否
Callback	Request.Conf	审核结果可以回调形式发送至您的回调地址，支持以 `http://` 或者 `https://` 开头的地址，例如：`http://www.callback.com`	String	否

返回结果说明

- 成功：成功则返回 DocumentAuditingResponse 文档审核任务结果对象。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见异常处理。

查询文档审核任务

功能说明

用于查询文档审核任务执行情况和结果。

方法原型

```
DocumentAuditingResponse describeAuditingJob(DocumentAuditingRequest request);
```

请求示例

```
//1.创建任务请求对象
DocumentAuditingRequest request = new DocumentAuditingRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
request.setJobId("sdd5d1cc630fdc11ecb3fa5254009******");
//3.调用接口,获取任务响应对象
DocumentAuditingResponse response = client.describeAuditingDocumentJob(request);
```

参数说明

参数名称	描述	类型	是否必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID	String	是

返回结果说明

- 成功： 返回 DocumentAuditingResponse 审核任务结果对象。
- 失败： 发生错误（例如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见异常处理。

文本审核

最近更新时间：2022-12-14 15:21:11

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于文本审核的 API 概览和 SDK 示例代码。

API	操作描述
提交文本审核任务	用于提交一个文本审核任务
查询文本审核任务结果	用于查询指定的文本审核任务

提交文本审核任务

功能说明

用于提交一个文本审核任务。

方法原型

```
TextAuditingResponse createAuditingTextJobs(TextAuditingRequest request);
```

请求示例

```
//1.创建任务请求对象
TextAuditingRequest request = new TextAuditingRequest();
//2.添加请求参数 参数详情请见api接口文档
```

```
request.setBucketName("examplebucket-12500000008");
//2.1.1设置对象地址
//request.getInput().setObject("1.txt");
//2.1.2或直接设置请求内容,文本内容的Base64编码
request.getInput().setContent("Base64Str");
//2.2设置审核类型参数
request.getConf().setDetectType("all");
//2.3设置审核模板（可选）
//request.getConf().setBizType("aa3e9d84a6a079556b0109a935c*****");
//3.调用接口,获取任务响应对象
TextAuditingResponse response = client.createAuditingTextJobs(request);
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Request	无	文本审核的具体配置项。	Container	是

Container 类型 Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的内容。	Container	是
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	当前 COS 存储桶中的文本文件名称，例如在目录 test 中的文件 img.jpg，则文件名称为 test/img.jpg，文本文件仅支持UTF8编码和 GBK 编码的内容，且文件大小不得超过1MB。	String	否
Content	Request.Input	当传入的内容为纯文本信息，需要先经过 base64 编码，文本编码前的原文长度不能超过10000个 utf8 编码字符。若超出长度限制，接口将会报错。	String	否

版权所有：腾讯云计算（北京）有限责任公司

注意：

- 不可同时输入 Object 和 Content。
- 当选择 Object 时，审核结果为异步返回，可通过 查询文本审核任务结果 API 接口获取返回结果。
- 当选择 Content 时，审核结果为同步返回，可通过 响应体 查看返回结果。
- 目前仅支持中文、英文、阿拉伯数字的检测与审核。

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
BizType	Request.Conf	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置公共审核策略。	String	否
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告）、Illegal（违法）、Abuse（谩骂），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。	String	否
Callback	Request.Conf	审核结果可以回调形式发送至您的回调地址，支持以 `http://` 或者 `https://` 开头的地址，例如： `http://www.callback.com` 。	String	否

返回结果说明

- 成功：成功则返回 TextAuditingResponse 文本审核任务结果对象。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

查询文本审核任务

功能说明

用于查询文本审核任务执行情况和结果。

方法原型

```
TextAuditingResponse describeAuditingTextJob(TextAuditingRequest request);
```

请求示例

```
//1.创建任务请求对象
TextAuditingRequest request = new TextAuditingRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
request.setJobId("st68d08596f35011eb9324525400*****");
//3.调用接口,获取任务响应对象
TextAuditingResponse response = client.describeAuditingTextJob(request);
```

参数说明

参数名称	描述	类型	是否必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
jobId	需要查询的任务 ID	String	是

返回结果说明

- 成功：返回 TextAuditingResponse 审核任务结果对象。
- 失败：发生错误（例如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

文件处理
文件压缩

最近更新时间：2023-03-28 14:38:57

简介

本文档提供关于数据万象文件处理压缩任务的相关 API 概览以及 SDK 示例代码。

API	操作描述
提交多文件打包压缩任务	创建一个多文件打包压缩任务
查询多文件打包压缩结果	查询指定的多文件打包压缩任务结果

提交多文件打包压缩任务

功能说明

多文件打包压缩功能可以将您的多个文件，打包为 zip 等压缩包格式，以提交任务的方式进行多文件打包压缩，异步返回打包后的文件。

方法原型

```
public FileProcessJobResponse createFileProcessJob(FileProcessRequest request);
```

参数说明

Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
tag	Request	表示任务的类型，多文件打包压缩默认为：FileCompress。	String	是
operation	Request	包含文件打包压缩的处理规则。	Container	是
queueId	Request	任务所在的队列 ID。	String	是

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
callBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式。	String	否
callBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型。	String	否
callBack	Request	任务回调的地址，优先级高于队列的回调地址。	String	否
callBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情请参见 CallBackMqConfig	Container	否

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
fileCompressConfig	Request.Operation	指定文件打包压缩的处理规则。	Container	是
userData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否
output	Request.Operation	指定打包压缩后的文件保存的地址信息。	Container	是

Container 类型 FileCompressConfig 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
flatten	Request.Operation.FileCompressConfig	文件打包时，是否需要去除源文件已有的目录结构，有效值： • 0：不需要去除目录结构，打包后压缩包中的文件会保留原有的目录结构； • 1：需要，打包后压缩包内的文件会去除原有的目录结构，所有文件都在同一层级。例如：源文件 URL 为 https://domain/source/test.mp4，则源文件路径为 source/test.mp4，如果为 1，则 ZIP 包中该文件路径为 test.mp4；如果为0， ZIP 包中该文件路径为 source/test.mp4。	String	是
format	Request.Operation.FileCompressConfig	打包压缩的类型，有效值：zip、tar、tar.gz。	String	是
urlList	Request.Operation.FileCompressConfig	• 支持将需要打包的文件整理成索引文件，后台将根据索引文件内提供的文件 url，打包为一个压缩包文件。 • 索引文件需要保存在当前存储桶中，本字段需要提供索引文件的对象地址，例如：/test/index.csv。 • 索引文件格式：仅支持 CSV 文件，一行一条 URL（仅支持本存储桶文件），如有多列字段，默认取第一列作为 URL。最多不超过10000个文件, 总大小不超过50G, 否则会导致任务失败。	String	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
prefix	Request.Operation.FileCompressConfig	支持对存储桶中的某个前缀进行打包，如果需要对某个目录进行打包，需要加/，例如 test 目录打包，则值为：test/。最多不超过10000个文件，总大小不超过50G，否则会导致任务失败。	String	否
key	Request.Operation.FileCompressConfig	支持对存储桶中的多个文件进行打包，个数不能超过 1000，总大小不超过50G，否则会导致任务失败。	String 数组	否

> 注意：
> UrlList、Prefix、Key 三者仅能选择一个，不能都为空，也不会同时生效。如果填了多个，会按优先级 UrlList > Prefix > Key 取最高优先级执行。

Container 类型 Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
region	Request.Operation.Output	存储桶的地域。	String	是
bucket	Request.Operation.Output	保存压缩后文件的存储桶。	String	是
object	Request.Operation.Output	压缩后文件的文件名	String	是

返回结果说明

- 成功：返回 FileProcessJobResponse 对象响应信息。
- 失败：发生错误（如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建任务请求对象
FileProcessRequest request=new FileProcessRequest();
//2.添加请求参数 参数详情请见 API 接口文档
```

```
request.setBucketName("demo-1234567890");
request.setTag(FileProcessJobType.FileCompress);
FileCompressConfig fileCompressConfig=request.getOperation().getFileCompressConfi
g();
fileCompressConfig.setFormat("zip");
fileCompressConfig.setFlatten("0");
List<String> keyList=fileCompressConfig.getKey();
keyList.add("mark/pic-1.jpg");
keyList.add("mark/pic-1.pdf");
//QueueId 需要从控制台获取
request.setQueueId("p1ff062b35a494cf0ac4b572df22****");
MediaOutputObject output=request.getOperation().getOutput();
output.setBucket("demo-1234567890");
output.setRegion("ap-shanghai");
output.setObject("output/demo.zip");
//3.调用接口,获取任务响应对象
FileProcessJobResponse response=client.createFileProcessJob(request);
```

查询多文件打包压缩结果

功能说明

查询一个文件处理任务,根据任务 ID 查询任务详情。

方法原型

```
public FileProcessJobResponse describeFileProcessJob(FileProcessRequest request);
```

参数说明

参数名称	描述	类型	是否必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
jobId	要查询的任务 ID	String	是

返回结果说明

- 成功：返回任务详情响应包装类，类中包含一个 FileProcessJobResponse 任务详情对象。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建任务请求对象
FileProcessRequest request = new FileProcessRequest();
//2.添加请求参数 参数详情请见 API 接口文档
request.setBucketName("demo-1234567890");
request.setJobId("fda7eb1607b8411ed8c182156726*****");
//3.调用接口,获取任务响应对象
FileProcessJobResponse response = client.describeFileProcessJob(request);
```

文件哈希值计算

最近更新时间：2023-03-28 14:38:56

简介

本文档提供关于数据万象文件处理哈希值计算任务的相关 API 概览以及 SDK 示例代码。

API	操作描述
提交哈希值计算任务	创建一个文件处理哈希值计算任务
查询哈希值计算结果	查询一个文件处理哈希值计算结果任务

提交哈希值计算任务

功能说明

以提交任务的方式进行文件哈希值计算，异步返回计算得到的哈希值。

方法原型

```
public FileProcessJobResponse createFileProcessJob(FileProcessRequest request);
```

参数说明

Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
tag	Request	表示任务的类型，哈希值计算任务类型为：FileHashCode。	String	是
operation	Request	包含哈希值计算的处理规则。	Container	是
queueId	Request	任务所在的队列 ID。	String	是
callBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式。	String	否
callBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型。	String	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
callBack	Request	任务回调的地址，优先级高于队列的回调地址。	String	否
callBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情请参见 CallBackMqConfig	Container	否

Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
object	Request.Input	文件名，取值为文件在当前存储桶中的完整名称。	String	是

Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
fileHashCodeConfig	Request.Operation	指定哈希值计算的处理规则。	Container	是
userData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否

FileHashCodeConfig 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
type	Request.Operation.FileHashCodeConfig	哈希值的算法类型，有效值：MD5、SHA1、SHA256。	String	是
addToHeader	Request.Operation.FileHashCodeConfig	是否将计算得到的哈希值添加至文件自定义 header, 有效值：true、false，默认值为 false。自定义header根据 `Type` 的值变化，例如 `Type` 值为MD5时，自定义 heade r为 x-cos-meta-md5。	String	否

返回结果说明

- 成功：返回 FileProcessJobResponse 对象响应信息。
- 失败：发生错误（如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建任务请求对象
FileProcessRequest request = new FileProcessRequest();
//2.添加请求参数 参数详情请见 api 接口文档
request.setBucketName("demo-1234567890");
request.setTag(FileProcessJobType.FileHashCode);
request.getInput().setObject("input/1.mp4");
FileHashCodeConfig fileHashCodeConfig = request.getOperation().getFileHashCodeCon
fig();
fileHashCodeConfig.setType("MD5");
fileHashCodeConfig.setAddToHeader("true");
request.setQueueId("p1ff062b35a494cf0ac4b572df22a****");
//3.调用接口,获取任务响应对象
FileProcessJobResponse response = client.createFileProcessJob(request);
```

查询哈希值计算结果

功能说明

查询一个文件处理任务，根据任务 ID 查询任务详情。

方法原型

```
public FileProcessJobResponse describeFileProcessJob(FileProcessRequest request);
```

参数说明

参数名称	描述	类型	是否必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
jobId	要查询的任务 ID	String	是

返回结果说明

- 成功： 返回任务详情响应包装类，类中包含一个 FileProcessJobResponse 任务详情对象。

- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建任务请求对象
FileProcessRequest request = new FileProcessRequest();
//2.添加请求参数 参数详情请见 api 接口文档
request.setBucketName("demo-1234567890");
request.setJobId("fda7eb1607b8411ed8c182156726*****");
//3.调用接口,获取任务响应对象
FileProcessJobResponse response = client.describeFileProcessJob(request);
```

文件解压

最近更新时间：2023-03-28 14:38:57

简介

本文档提供关于数据万象文件处理解压任务的相关 API 概览以及 SDK 示例代码。

API	操作描述
提交文件解压任务	创建一个文件处理解压任务
查询文件解压结果	查询一个文件处理解压任务

提交文件解压任务

功能说明

以提交任务的方式进行文件解压缩，异步返回压缩包内被解压出来的文件

方法原型

```
public FileProcessJobResponse createFileProcessJob(FileProcessRequest request);
```

参数说明

Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
tag	Request	表示任务的类型，文件解压默认为：FileUncompress。	String	是
input	Request	包含待操作的文件信息。	Container	是
operation	Request	包含文件解压的处理规则。	Container	是
queueId	Request	任务所在的队列 ID。	String	是
callBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式。	String	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
callBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型。	String	否
callBack	Request	任务回调的地址，优先级高于队列的回调地址。	String	否
callBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
object	Request.Input	文件名，取值为文件在当前存储桶中的完整名称。	String	是

operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
fileUncompressConfig	Request.Operation	指定文件解压的处理规则。	Container	是
userData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否
output	Request.Operation	指定解压后的文件保存的存储桶信息。	Container	是

FileUncompressConfig 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
prefix	Request.Operation.FileUncompressConfig	指定解压后输出文件的前缀，不填则默认保存在存储桶根路径。	String	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
prefixReplaced	Request.Operation.FileUncompressConfig	指定解压后的文件路径是否需要替换前缀，有效值： - 0：不添加额外的前缀，解压缩将保存在Prefix指定的路径下（不会保留压缩包的名称，仅将压缩包内的文件保存至指定的路径）。 - 1：以压缩包本身的名称作为前缀，解压缩将保存在Prefix指定的路径下。 - 2：以压缩包完整路径作为前缀，此时如果不指定Prefix，就是解压到压缩包所在的当前路径（包含压缩包本身名称）。 - 默认值为0。	String	否

示例：
假设压缩包名为 test.zip，压缩包内的文件为“图片.jpg”，存储在了存储桶 A 的 123 目录下，则压缩包的完整路径为：123/test.zip。
对该压缩包进行解压缩，指定解压缩到存储桶A，并指定 `Prefix` 为 456，则不同的 `PrefixReplaced` 值对应解压后的文件保存情况如下：
0：“图片.jpg”会保存在 456 目录下，“图片.jpg”的完整路径为 456/图片.jpg；
1：“图片.jpg”会以 test 为前缀保存在 456 目录下，“图片.jpg”的完整路径为 456/test/图片.jpg；
2：“图片.jpg”会以 123/test 为前缀保存在 456 目录下，“图片.jpg”的完整路径为 456/123/test/图片.jpg。

Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
region	Request.Operation.Output	存储桶的地域。	String	是
bucket	Request.Operation.Output	保存解压后文件的存储桶。	String	是

版权所有：腾讯云计算（北京）有限责任公司

返回结果说明

- 成功：返回 FileProcessJobResponse 对象响应信息。
- 失败：发生错误（如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建任务请求对象
FileProcessRequest request = new FileProcessRequest();
//2.添加请求参数 参数详情请见 api 接口文档
request.setBucketName("demo-1234567890");
request.setTag(FileProcessJobType.FileUncompress);
request.getInput().setObject("output/demo.zip");
FileUnCompressConfig fileUnCompressConfig = request.getOperation().getFileUnCompr
essConfig();
fileUnCompressConfig.setPrefix("output/");
fileUnCompressConfig.setPrefixReplaced("1");
request.setQueueId("p1ff062b35a494cf0ac4b572df22a5650");
MediaOutputObject output = request.getOperation().getOutput();
output.setBucket("demo-1234567890");
output.setRegion("ap-shanghai");
//3.调用接口,获取任务响应对象
FileProcessJobResponse response = client.createFileProcessJob(request);
```

查询文件解压结果

功能说明

查询一个文件处理任务,根据任务 ID 查询任务详情。

方法原型

```
public FileProcessJobResponse describeFileProcessJob(FileProcessRequest request);
```

参数说明

参数名称	描述	类型	是否必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
jobId	要查询的任务 ID	String	是

返回结果说明

- 成功：返回任务详情响应包装类，类中包含一个 FileProcessJobResponse 任务详情对象。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建任务请求对象
FileProcessRequest request=new FileProcessRequest();
//2.添加请求参数 参数详情请见 api 接口文档
request.setBucketName("demo-1234567890");
request.setJobId("fda7eb1607b8411ed8c182156726*****");
//3.调用接口,获取任务响应对象
FileProcessJobResponse response=client.describeFileProcessJob(request);
```

媒体处理
模板操作

最近更新时间：2022-05-31 11:43:53

简介

本文档提供关于数据万象媒体处理任务的相关 API 概览以及 SDK 示例代码,此处以动图任务举例。

API	操作名	操作描述
CreateMediaTemplate	创建模板	用于创建一个新的模板
DeleteMediaTemplate	删除模板	删除一个模板
DescribeMediaTemplates	查询模板	查询模板列表
UpdateMediaTemplate	修改模板	修改一个模板

基本操作

创建模板

功能说明

用于创建一个新的模板。

方法原型

```
public MediaTemplateResponse createMediaTemplate(MediaTemplateRequest request);
```

参数说明

Request 的具体参数描述如下：

节点名称（关键字）	父节点	描述	类型	必选
Tag	Request	模板类型：Animation 动图，Snapshot 截图，Transcode 转码，Watermark 水印，SmartCover 智能封面	String	是

节点名称（关键字）	父节点	描述	类型	必选
Name	Request	模板名称仅支持中文、英文、数字、_、-和*	String	是
Container	Request	容器格式	Container	是
Video	Request	视频信息	Container	否
TimeInterval	Request	时间区间	Container	否
Snapshot	Request	截图	Container	否
Watermark	Request	水印	Container	否
Audio	Request	音频信息	Container	否
TransConfig	Request	转码配置	Container	否

Request 节点 Container 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
Format	Request.Container	容器格式：gif，hgif，webp hgif 为高质量 gif，即清晰度比较高的 gif 格式图	String	是

Request 节点 Video 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
Codec	Request.Video	编解码格式	String	是	无	gif、webp
Width	Request.Video	宽	String	否	视频原始宽度	• 值范围：[128，4096] • 单位：px • 若只设置 Width 时，按照视频原始比例计算 Height
Height	Request.Video	高	String	否	视频原始高度	• 值范围：[128，4096] • 单位：px • 若只设置 Height 时，按照视频原始比例计算 Width

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
Fps	Request.Video	帧率	String	否	视频原始帧率	• 值范围：(0，60] • 单位：fps • 如果不设置，那么播放速度按照原来的时间戳。这里设置 fps 为动图的播放帧率。
AnimateOnlyKeepKeyFrame	Request.Video	动图只保留关键帧	String	否	无	• true、false • 动图保留关键帧参数
AnimateTimeIntervalOfFrame	Request.Video	动图抽帧间隔时间	String	否	无	• （0，视频时长] • 动图抽帧时间间隔 • 若设置 TimeInterval.Duration，则小于该值
AnimateFramesPerSecond	Request.Video	Animation 每秒抽帧帧数	String	否	无	• （0，视频帧率) • 动图抽帧频率 • 优先级：AnimateFramesPerSecond > AnimateOnlyKeepKeyFrame > AnimateTimeIntervalOfFrame
Quality	Request.Video	设置相对质量	String	否	无	• [1, 100) • webp 图像质量设定生效，gif 没有质量参数

Request 节点 TimeInterval 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
Start	Request.TimeInterval	开始时间	String	否	0	• [0 视频时长] • 单位为秒 • 支持 float 格式，执行精度精确到毫秒
Duration	Request.TimeInterval	持续时间	String	否	视频时长	• [0 视频时长] • 单位为秒 • 支持 float 格式，执行精度精确到毫秒

版权所有：腾讯云计算（北京）有限责任公司

Container 类型 Snapshot 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
Mode	Request.Snapshot	截图模式	String	是	Interval	• 值范围：{Interval, Average} • Interval 表示间隔模式 Average 表示平均模式 • Interval 模式：Start，TimeInterval，Count 参数生效。当设置 Count，未设置 TimeInterval 时，表示截取所有帧，共 Count 张图片 • Average 模式：Start，Count 参数生效。表示从 Start 开始到视频结束，按平均间隔截取共 Count 张图片
Start	Request.Snapshot	开始时间	String	是	0	• [0 视频时长] • 单位为秒 • 支持 float 格式，执行精度精确到毫秒
TimeInterval	Request.Snapshot	截图频率	String	否	无	• (0 3600] • 单位为秒 • 支持 float 格式，执行精度精确到毫秒
Count	Request.Snapshot	截图数量	String	是	无	(0 10000]
Width	Request.Snapshot	宽	String	否	视频原始宽度	• 值范围：[128，4096] • 单位：px • 若只设置 Width 时，按照视频原始比例计算 Height
Height	Request.Snapshot	高	String	否	视频原始高度	• 值范围：[128，4096] • 单位：px • 若只设置 Height 时，按照视频原始比例计算Width

Container 类型 Watermark 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
Type	Request.Watermark	水印类型	String	是	无	Text：文字水印、Image：图片水印
Pos	Request.Watermark	基准位置	String	是	无	TopRight、TopLeft、BottomRight、BottomLeft
LocMode	Request.Watermark	偏移方式	String	是	无	Relativity：按比例、Absolute：固定位置
Dx	Request.Watermark	水平偏移	String	是	无	• 当 locMode 为 Relativity 时，单位为%，取值范围[0 100] • 当 locMode 为 Absolute 时，单位为px，取值范围[0 4096]
Dy	Request.Watermark	垂直偏移	String	是	无	• 当 locMode 为 Relativity 时，单位为%，取值范围[0 100] • 当 locMode 为 Absolute 时，单位为px，取值范围[0 4096]
StartTime	Request.Watermark	水印开始时间	String	否	0	• [0 视频时长] • 单位为秒 • 支持 float 格式，执行精度精确到毫秒
EndTime	Request.Watermark	水印结束时间	String	否	视频结束时间	• [0 视频时长] • 单位为秒 • 支持 float 格式，执行精度精确到毫秒
Image	Request.Watermark	图片水印节点	Container	否	无	无
Text	Request.Watermark	文本水印节点	Container	否	无	无

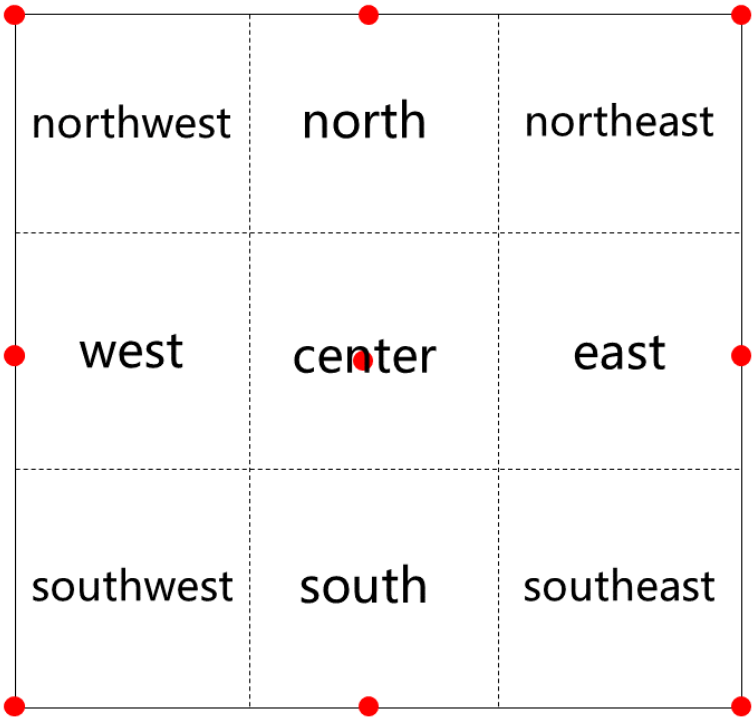
版权所有：腾讯云计算（北京）有限责任公司

Container 类型 Image 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
Url	Request.Watermark.Image	水印图地址	String	是	无	同 bucket 的水印图片地址
Mode	Request.Watermark.Image	尺寸模式	String	是	无	• Original：原有尺寸 • Proportion：按比例 • Fixed：固定大小
Width	Request.Watermark.Image	宽	String	否	无	• 当 Mode 为 Original，水印图宽 • 当 Mode 为 Proportion，单位为%，取值范围：[1 100] • 当 Mode 为 Fixed，单位为px，取值范围：[1，4096] • 若只设置 Width 时，按照视频原始比例计算 Height
Height	Request.Watermark.Image	高	String	否	无	• 当 Mode 为 Original，水印图高 • 当 Mode 为 Proportion，单位为%，取值范围：[1 100] • 当 Mode 为 Fixed，单位为px，取值范围：[1，4096] • 若只设置 Height 时，按照视频原始比例计算 Width
Transparency	Request.Watermark.Image	透明度	String	是	无	值范围：[1 100]，单位%

版权所有：腾讯云计算（北京）有限责任公司

水印位置说明：



Container 类型 Text 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
FontSize	Request.Watermark.Text	字体大小	String	是	无	值范围：[5 100]，单位为px
FontType	Request.Watermark.Text	字体类型	String	是	无	参考下表
FontColor	Request.Watermark.Text	字体颜色	String	是	无	格式：0xRRGGBB
Transparency	Request.Watermark.Text	透明度	String	是	无	值范围：[1 100]，单位为%
Text	Request.Watermark.Text	水印内容	String	是	无	长度不超过64个字符，仅支持中文、英文、数字、_、-和*

Text 的 FontType 具体数据描述如下：

版权所有：腾讯云计算（北京）有限责任公司

字体名称	支持的语言	描述
simfang.ttf	中/英	仿宋
simhei.ttf	中/英	黑体
simkai.ttf	中/英	楷体
simsun.ttc	中/英	宋体
STHeiti-Light.ttc	中/英	华文黑体
STHeiti-Medium.ttc	中/英	华文黑体中
youyuan.TTF	中/英	幼圆
ariblk.ttf	英	无
arial.ttf	英	无
ahronbd.ttf	英	无
Helvetica.dfont	英	无
HelveticaNeue.dfont	英	无

Audio 音频视频支持的格式如下表：

Container	Audio Codecs	Video Codecs
flv/mp4/ts/hls	AAC、MP3	H.264
aac	aac	不支持
mp3	mp3	不支持

Audio 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
Codec	Request.Audio	编解码格式	String	否	aac	取值 aac、mp3

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
Samplerate	Request.Audio	采样率	String	否	44100	• 单位：Hz • 可选 11025、22050、32000、44100、48000、96000 • 不同的封装，mp3 支持不同的采样率，如下表所示
Bitrate	Request.Audio	原始音频码率	String	否	无	• 单位：Kbps • 值范围：[8，1000]
Channels	Request.Audio	声道数	String	否	无	• 当 Codec 设置为 aac，支持1、2、4、5、6、8 • 当 Codec 设置为 mp3，支持1、2
Remove	Request.Audio	是否删除音频流	String	否	false	取值 true、false

Y表示支持这种采样率，N表示不支持

封装格式/音频采样率	11025	22050	32000	44100	48000	96000
flv	Y	Y	N	Y	N	N
mp4	N	Y	Y	Y	Y	N
avi/hls/ts/mp3	Y	Y	Y	Y	Y	N

TransConfig 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
AdjDarMethod	Request.TransConfig	分辨率调整方式	String	否	none	• 取值 scale、crop、pad、none • 当输出视频的宽高比与原视频不等时，需要此参数进行执行调整方式

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
IsCheckReso	Request.TransConfig	是否检查分辨率	String	否	false	• true、false • 当为 false 时，按照配置参数转码
ResoAdjMethod	Request.TransConfig	分辨率调整方式	String	否	0	• 取值0、1；0 表示使用原视频分辨率；1表示返回转码失败 • 当 IsCheckReso 为 true 时生效
IsCheckVideoBitrate	Request.TransConfig	是否检查视频码率	String	否	false	• true、false • 当为 false 时，按照配置参数转码
VideoBitrateAdjMethod	Request.TransConfig	视频码率调整方式	String	否	0	• 取值0、1；0 表示使用原视频码率；1表示返回转码失败 • 当 IsCheckVideoBitrate 为 true 时生效
IsCheckAudioBitrate	Request.TransConfig	是否检查音频码率	String	否	false	• true、false • 当为 false 时，按照配置参数转码
AudioBitrateAdjMethod	Request.TransConfig	音频码率调整方式	String	否	0	• 取值0、1；0 表示使用原音频码率；1表示返回转码失败 • 当 IsCheckAudioBitrate 为 true 时生效

Video 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
Codec	Request.Video	编解码格式	String	否	H.264	H.264

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
Width	Request.Video	宽	String	否	视频原始宽度	• 值范围：[128，4096] • 单位：px • 若只设置 Width 时，按照视频原始比例计算 Height
Height	Request.Video	高	String	否	视频原始高度	• 值范围：[128，4096] • 单位：px • 若只设置 Height 时，按照视频原始比例计算 Width
Fps	Request.Video	帧率	String	否	无	• 值范围：(0，60] • 单位：fps
Remove	Request.Video	是否删除视频流	String	否	false	true、false
Profile	Request.Video	编码级别	String	否	high	• 支持 baseline、main、high • baseline：适合移动设备 • main：适合标准分辨率设备 • high：适合高分辨率设备 • 仅 H.264 支持此参数
Bitrate	Request.Video	视频输出文件的码率	String	否	无	• 值范围：[10，50000] • 单位：Kbps
Crf	Request.Video	码率-质量控制因子	String	否	无	• 值范围：(0，51] • 如果设置了 Crf，则 Bitrate 的设置失效
Gop	Request.Video	关键帧间最大帧数	String	否	无	值范围：[0，100000]
Preset	Request.Video	视频算法器预置	String	否	medium	• 仅 H.264 支持该参数 • 取值 veryfast、fast、medium、slow、slower
Bufsize	Request.Video	缓冲区大小	String	否	无	• 值范围：[1000，128000] • 单位：Kb
Maxrate	Request.Video	视频码率峰值	String	否	无	• 值范围：[10，50000] • 单位：Kbps

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	必选	默认值	限制
HlsTsTime	Request.Video	hls分片时间	String	否	5	• (0 视频时长] • 单位为秒
Pixfmt	Request.Video	视频颜色格式	String	否	无	支持 yuv420p、yuv422p、yuv444p、yuvj420p、yuvj422p、yuvj444p

返回结果说明

- 成功：返回 MediaTemplateResponse 对象响应信息，内含创建的模板详情。
- 失败：发生错误（如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建模板请求对象
MediaTemplateRequest request = new MediaTemplateRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
request.setTag("Animation");
request.setName("TestTemplate40");
request.getContainer().setFormat("gif");
request.getVideo().setCodec("gif");
request.getVideo().setWidth("1280");
request.getVideo().setFps("15");
request.getVideo().setAnimateOnlyKeepKeyFrame("true");
request.getTimeInterval().setStart("0");
request.getTimeInterval().setDuration("60");
//3.调用接口,获取模板响应对象
MediaTemplateResponse response = client.createMediaTemplate(request);
```

删除模板

功能说明

删除一个模板

方法原型

```
public Boolean deleteMediaTemplate(MediaTemplateRequest request);
```

参数说明

参数名称	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
templateId	要取消的模板 id	String	是

返回结果说明

- 成功：返回一个布尔类型，删除成功则返回 true。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
MediaTemplateRequest request = new MediaTemplateRequest();
request.setBucketName("examplebucket-1250000000");
request.setTemplateId("t19c4a60ae1a694621a01f0c7130c*****");
Boolean response = client.deleteMediaTemplate(request);
```

查询模板列表

功能说明

查询模板列表

方法原型

```
public MediaJobResponse describeMediaJob(MediaJobsRequest req);
```

参数说明

参数名称	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
tag	模板 Tag：Animation 动图，Snapshot 截图，Transcode 转码，Watermark 水印，SmartCover 智能封面	String	是
category	Official，Custom，默认值：Custom	String	否
ids	模板 ID，以 `,` 符号分割字符串	String	否
name	模板名称前缀	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型	必选
pageNumber	第几页	Integer	否
pageSize	每页个数	Integer	否

返回结果说明

- 成功：返回模板响应包装类，其中包含模板详情集合 templateList。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建任务请求对象
MediaTemplateRequest request = new MediaTemplateRequest();
request.setBucketName("examplebucket-1250000000");
MediaListTemplateResponse response = client.describeMediaTemplates(request);
List<MediaTemplateObject> templateList = response.getTemplateList();
```

修改模板

功能说明

修改模板

方法原型

```
public Boolean updateMediaTemplate(MediaTemplateRequest request);
```

参数说明

节点名称（关键字	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
templateId	修改的模板 ID	String	是

> 说明：
> 其余参数请参见创建模板接口，与其参数一致

返回结果说明

- 成功：修改成功则返回 true。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
MediaTemplateRequest request = new MediaTemplateRequest();
request.setBucketName("examplebucket-1250000000");
request.setTemplateId("t19c4a60ae1a694621a01f0c7130*****");
request.setTag("Animation");
request.setName("updateName");
request.getContainer().setFormat("gif");
Boolean aBoolean = client.updateMediaTemplate(request);
```

工作流操作

最近更新时间：2022-06-21 14:54:11

简介

本文档提供关于数据万象媒体处理工作流的相关 API 概览以及 SDK 示例代码，此处以动图任务举例。

API	操作名	操作描述
DeleteWorkflow	删除工作流	删除一个工作流
DescribeWorkflow	查询工作流	查询一个工作流
DescribeWorkflowExecution	查询工作流实例详情	查询一个工作流实例详情
DescribeWorkflowExecutions	查询工作流详情列表	获取工作流实例列表
TriggerWorkflowList	触发工作流	工作流执行

删除工作流

功能说明

删除一个工作流。

方法原型

```
public Boolean deleteWorkflow(MediaWorkflowListRequest request);
```

参数说明

Request中的具体数据描述如下：

节点名称（关键字）	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
workflowId	工作流 ID	String	是

返回结果说明

- 成功：成功则返回 true。
- 失败：发生错误（如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建工作流请求对象
MediaWorkflowListRequest request = new MediaWorkflowListRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
request.setWorkflowId("aaaa");
Boolean response = client.deleteWorkflow(request);
```

查询工作流

功能说明

用于搜索工作流。

方法原型

```
public MediaWorkflowListResponse describeWorkflow(MediaWorkflowListRequest reques
t);
```

参数说明

参数名称	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
ids	工作流 ID，以 `,` 符号分割字符串	String	否
name	工作流名称	String	否
pageNumber	第几页	String	否
pageSize	每页个数	String	否

返回结果说明

- 成功： 返回工作流集合响应对象，其中包含一个工作流对象集合。

版权所有：腾讯云计算（北京）有限责任公司

- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建工作流请求对象
MediaWorkflowListRequest request = new MediaWorkflowListRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
MediaWorkflowListResponse response = client.describeWorkflow(request);
List<MediaWorkflowObject> mediaWorkflowList = response.getMediaWorkflowList();
```

查询工作流实例详情

功能说明

查询工作流实例详情。

方法原型

```
public MediaWorkflowExecutionResponse describeWorkflowExecution(MediaWorkflowList
Request request);
```

参数说明

参数名称	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
runId	工作流实例 ID	String	是

返回结果说明

- 成功：返回工作流实例响应包装类，类中包含一个工作流实例详情对象。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建工作流请求对象
MediaWorkflowListRequest request = new MediaWorkflowListRequest();
//2.添加请求参数 参数详情请见api接口文档
```

版权所有：腾讯云计算（北京）有限责任公司

```
request.setBucketName("examplebucket-1250000000");
request.setRunId("i34bfd8d7eae711ea89fe525400c******");
MediaWorkflowExecutionResponse response = client.describeWorkflowExecution(reques
t);
```

查询工作流详情列表

功能说明

查询工作流详情列表。

方法原型

```
public MediaWorkflowExecutionsResponse describeWorkflowExecutions(MediaWorkflowLi
stRequest request);
```

参数说明

节点名称	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
workflowId	工作流 ID	String	是
name	文件名称	String	否
orderByTime	Desc 或者 Asc。默认为 Desc	String	否
size	拉取的最大任务数。默认为10。最大为100	String	否
states	工作流实例状态，以 `,` 分割支持多状态 All，Success，Failed，Running，Cancel。默认为 All	String	否
startCreationTime	拉取创建时间大于该时间。格式为：`%Y-%m-%dT%H:%m:%S%z`	String	否
endCreationTime	拉取创建时间小于该时间。格式为：`%Y-%m-%dT%H:%m:%S%z`	String	否
nextToken	请求的上下文，用于翻页。下一页输入 token	String	否

返回结果说明

- 成功：返回工作流实例集合响应对象，其中包含一个工作流对象实例集合。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建工作流请求对象
MediaWorkflowListRequest request = new MediaWorkflowListRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
request.setWorkflowId("w4e6963a18e2446ed8bc8f09410e******");
MediaWorkflowExecutionsResponse response = client.describeWorkflowExecutions(requ
est);
List<MediaWorkflowExecutionObject> workflowExecutionList = response.getWorkflowEx
ecutionList();
```

手动触发工作流

功能说明

用于手动触发工作流。

方法原型

```
public MediaWorkflowListResponse triggerWorkflowList(MediaWorkflowListRequest req
uest);
```

请求示例

```
//1.创建工作流请求对象
MediaWorkflowListRequest request = new MediaWorkflowListRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("DemoBucket-123456789");
request.setWorkflowId("we32f75950afe4a4682463d8158d*****");
request.setObject("1.mp4");
MediaWorkflowListResponse response = client.triggerWorkflowList(request);
```

参数说明

节点名称	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
object	需要进行工作流处理的对象名称	String	是
workflowId	需要触发的工作流 ID	String	是

节点名称	描述	类型	必选
name	存量触发任务名称，支持中文、英文、数字、—和_，长度限制128字符，默认为空	String	否

返回结果说明

- 成功：返回 MediaWorkflowListResponse 实例。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

任务操作

最近更新时间：2022-05-23 12:27:24

简介

本文档提供关于数据万象媒体处理任务的相关 API 概览以及 SDK 示例代码，此处以动图任务举例。

API	操作名	操作描述
CreateMediaJobs	创建任务	创建一个媒体处理任务
CancelMediaJob	删除任务	删除一个媒体处理任务，正在处理中的任务不可删除
DescribeMediaJob	查询任务	查询一个任务
DescribeMediaJobs	查询任务列表	查询队列中的任务列表

基本操作

创建任务

功能说明

创建一个媒体处理任务。

方法原型

```
public MediaJobResponse createMediaJobs(MediaJobsRequest req);
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
Tag	Request	创建任务的 Tag：Animation 动图任务，Snapshot 截图任务，Transcode 转码任务，SmartCover 智能封面任务	String	是
Input	Request	待操作的媒体信息	Container	是
Operation	Request	操作规则	Container	是

节点名称（关键字）	父节点	描述	类型	必选
QueueId	Request	任务所在的队列 ID	String	是

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
Object	Request.Input	媒体文件的名字	String	是

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
Animation	Request.Operation	指定该任务的参数，同创建动图模板 CreateMediaTemplate 接口中的 Request.Animation	Container	否
Snapshot	Request.Operation	指定该任务的参数，同创建截图模板 CreateMediaTemplate 接口中的 Request.Snapshot	Container	否
Transcode	Request.Operation	指定该任务的参数，同创建转码模板 CreateMediaTemplate 接口中的 Request.Transcode	Container	否
Watermark	Request.Operation	指定该任务的参数，同创建水印模板 CreateMediaTemplate 接口中的 Request.Watermark	Container	否
SmartCover	Request.Operation	当 Tag 为 SmartCover 时有效，目前为空	Container	否
TemplateId	Request.Operation	指定的模板 ID	String	否
Output	Request.Operation	结果输出地址	Container	是

注意：
优先使用 TemplateId，无 TemplateId 时使用对应任务类型的参数。

Container 类型 Animation 的具体数据描述如下：

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	必选
Container	Request.Operation.Animation	同动图模板 CreateMediaTemplate 接口中的 Request.Container	Container	否
Video	Request.Operation.Animation	同动图模板 CreateMediaTemplate 接口中的 Request.Video	Container	否
TimeInterval	Request.Operation.Animation	同动图模板 CreateMediaTemplate 接口中的 Request.TimeInterval	Container	否

Container 类型 Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
Region	Request.Operation.Output	存储桶的地域	String	是
Bucket	Request.Operation.Output	存储结果的存储桶	String	是
Object	Request.Operation.Output	结果文件的名字	String	是

返回结果说明

- 成功：返回 job 对象响应信息。
- 失败：发生错误（如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建任务请求对象
MediaJobsRequest request = new MediaJobsRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
request.setTag("Transcode");
request.getInput().setObject("1.mp4");
request.getOperation().setTemplateId("t0e09a9456d4124542b1f0e44d501*****");
request.getOperation().getOutput().setBucket("examplebucket-1250000000");
request.getOperation().getOutput().setRegion("ap-chongqing");
request.getOperation().getOutput().setObject("2.mp4");
request.setQueueId("p9900025e4ec44b5e8225e70a52170834");
//3.调用接口,获取任务响应对象
MediaJobResponse response = client.createMediaJobs(request);
```

取消任务

功能说明

取消一个未在处理中的任务。

方法原型

```
public Boolean cancelMediaJob(MediaJobsRequest req);
```

参数说明

参数名称	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
jobId	要取消的任务 ID	String	是

返回结果说明

- 成功：返回一个布尔类型，成功则返回 true。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
MediaJobsRequest request = new MediaJobsRequest();
request.setBucketName("examplebucket-1250000000");
request.setJobId("jae776cb4ec3011eab2cdd3817d4*****");
Boolean response = client.cancelMediaJob(request);
```

查询任务

功能说明

根据任务 id 查询任务详情。

方法原型

```
public MediaJobResponse describeMediaJob(MediaJobsRequest req);
```

参数说明

参数名称	描述	类型	必选

参数名称	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
jobId	要取消的任务 id	String	是

返回结果说明

- 成功：返回任务详情响应包装类，类中包含一个 MediaJobObject 任务详情对象。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建任务请求对象
MediaJobsRequest request = new MediaJobsRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
request.setJobId("j29a82fea08ba11ebb54bc9d1c05*****");
//3.调用接口,获取任务响应对象
MediaJobResponse response = client.describeMediaJob(request);
```

查询任务列表

功能说明

查询队列中的任务列表。

方法原型

```
public MediaListJobResponse describeMediaJobs(MediaJobsRequest cIMediaJobsReques
t);
```

参数说明

节点名称（关键字	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
queueId	拉取该队列 ID 下的任务	String	是
tag	任务的Tag：Animation	String	是
orderByTime	Desc 或者 Asc。默认为 Desc	String	否

节点名称（关键字	描述	类型	必选
nextToken	请求的上下文，用于翻页。上次返回的值	String	否
size	拉取的最大任务数。默认为10。最大为100	Integer	否
states	拉取该状态的任务，以,分割支持多状态 All，Submitted，Running，Success， Failed，Pause，Cancel。默认为 All	String	否
startCreationTime	拉取创建时间大于该时间的任务。格式为：`%Y-%m-%dT%H:%m:%S%z`	String	否
endCreationTime	拉取创建时间小于该时间的任务。格式为：`%Y-%m-%dT%H:%m:%S%z`	String	否

返回结果说明

- 成功：返回一个任务集合响应实体。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
MediaJobsRequest request = new MediaJobsRequest();
request.setBucketName("examplebucket-1250000000");
request.setQueueId("p9900025e4ec44b5e8225e70a521*****");
request.setTag("Transcode");
MediaListJobResponse response = client.describeMediaJobs(request);
List<MediaJobObject> jobsDetail = response.getJobsDetail();
```

桶操作

最近更新时间：2022-05-23 12:24:10

简介

本文档提供关于数据万象媒体处理桶的相关 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DescribeMediaBuckets	查询媒体 Bucket	查询当前账号下已经开通媒体处理功能的桶列表

基本操作

查询开通媒体处理功能的桶列表

功能说明

查询当前账号下已经开通媒体处理功能的桶列表。

方法原型

```
public MediaBucketResponse describeMediaBuckets(MediaBucketRequest mediaBucketRequest);
```

参数说明

参数名称	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
bucketNames	bucket 名字，以 `,` 分隔，支持多 bucket，精确搜索	String	否
regions	地区信息，以 `,` 分隔字符串，支持 All，ap-shanghai，ap-beijing	String	否
pageNumber	第几页	String	否
pageSize	每页个数	String	否

返回结果说明

- 成功：返回 bucket 对象信息。

版权所有：腾讯云计算（北京）有限责任公司

- 失败：发生错误（如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建模板请求对象
MediaBucketRequest request = new MediaBucketRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
//3.调用接口,获取桶响应对象
MediaBucketResponse response = client.describeMediaBuckets(request);
```

队列操作

最近更新时间：2022-05-23 12:27:25

简介

本文档提供关于数据万象媒体处理队列的相关 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DescribeMediaQueues	查询队列	查询当前账号下对应的队列信息
UpdateMediaQueue	更新队列	接口用于更新队列，修改回调信息

基本操作

查询队列

功能说明

查询当前账号下对应的队列信息。

方法原型

```
public MediaListQueueResponse describeMediaQueues(MediaQueueRequest mediaQueueRequest);
```

参数说明

参数名称	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
queueIds	队列 ID，以 `,` 符号分割字符串	string	否
state	• Active 表示队列内的作业会被媒体转码服务调度转码执行 • Paused 表示队列暂停，作业不再会被媒体转码调度转码执行，队列内的所有作业状态维持在暂停状态，已经处于转码中的任务将继续转码，不受影响	string	否
pageNumber	第几页	string	否

参数名称	描述	类型	必选
pageSize	每页个数	string	否

返回结果说明

- 成功：返回队列对象集合信息。
- 失败：发生错误（如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
MediaQueueRequest request = new MediaQueueRequest();
request.setBucketName("examplebucket-1250000000");
MediaListQueueResponse response = client.describeMediaQueues(request);
```

更新队列

功能说明

接口用于更新队列，修改回调信息。

方法原型

```
public MediaQueueResponse updateMediaQueue(MediaQueueRequest mediaQueueRequest);
```

参数说明

参数名称	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
Name	模板名称,长度限制100字符	string	是
state	• Active 表示队列内的作业会被媒体转码服务调度转码执行 • Paused 表示队列暂停，作业不再会被媒体转码调度转码执行，队列内的所有作业状态维持在暂停状态，已经处于转码中的任务将继续转码，不受影响	string	是
QueueID	管道 ID	string	是
NotifyConfig	通知渠道,第三方回调 url	Container	是

Container 类型 NotifyConfig 的具体数据描述如下：

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型	必选
Url	回调 url 地址	String	否
Type	回调类型，普通回调：Url	String	否
Event	回调事件，视频转码完成：TransCodingFinish	String	否
State	回调开关，Off，On	String	否

返回结果说明

- 成功：返回队列响应实体，包含有关 queue 的描述。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
MediaQueueRequest request = new MediaQueueRequest();
request.setBucketName("examplebucket-1250000000");
request.setQueueId("p9900025e4ec44b5e8225e70a521*****");
request.getNotifyConfig().setUrl("cloud.tencent.com");
request.setState("Active");
request.setName("testQueue");
MediaQueueResponse response = client.updateMediaQueue(request);
```

视频截帧

最近更新时间：2022-05-23 12:27:25

简介

本文档提供关于数据万象媒体截图的相关 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GenerateSnapshot	获取媒体文件某个时间的截图	获取媒体文件某个时间的截图

基本操作

获取媒体文件某个时间的截图

功能说明

获取媒体文件某个时间的截图。

方法原型

```
public SnapshotResponse generateSnapshot(SnapshotRequest request);
```

参数说明

参数名称	描述	类型	是否必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
Input	媒体文件的位置信息	Container	是
Time	截取哪个时间点的内容，单位为秒	Float	是
Output	截图保存的位置信息	Container	是
Width	截图的宽。默认为0	Int	否
Height	截图的高。默认为0。 Width 和 Height 都为0时，表示使用视频的宽高。 如果单个为0，则以另外一个值按视频宽高比例自动适应	Int	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型	是否必选
Format	截图的格式，支持 jpg 和 png，默认 jpg	String	否
Mode	截帧方式 • keyframe：截取指定时间点之前的最近的一个关键帧 • exactframe：截取指定时间点的帧 默认值为 exactframe	String	否
Rotate	图片旋转方式 • auto：按视频旋转信息进行自动旋转 • off：不旋转 默认值为 auto	String	否

Input 的内容：

参数名称	父节点	描述	类型	是否必选
Object	Request.Input	文件的名称	String	是

Output 的内容：

参数名称）	父节点	描述	类型	是否必选
Region	Request.Output	存储桶所在的地域	String	是
Bucket	Request.Output	文件所属的 COS 存储桶	String	是
Object	Request.Output	文件的名称	String	是

返回结果说明

- 成功：返回截图实体信息。
- 失败：发生错误（如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建截图请求对象
SnapshotRequest request = new SnapshotRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
request.getInput().setObject("1.mp4");
request.getOutput().setBucket("examplebucket-1250000000");
```

```
request.getOutput().setRegion("ap-chongqing");
request.getOutput().setObject("test/1.jpg");
request.setTime("15");
//3.调用接口,获取截图响应对象
SnapshotResponse response = client.generateSnapshot(request);
```

视频元信息获取

最近更新时间：2022-05-23 12:27:25

简介

本文档提供关于数据万象处理媒体信息的相关 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GenerateMediainfo	获取媒体文件的信息	查询 Bucket 中文件的媒体信息详情

基本操作

获取媒体文件的信息

功能说明

查询 Bucket 中文件的媒体信息详情。

方法原型

```
public MediaInfoResponse generateMediainfo(MediaInfoRequest request);
```

参数说明

参数名称	描述	类型	必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
object	input.objet 查询的对象在 Bucket 中的位置信息	String	是

返回结果说明

- 成功：返回媒体对象信息。
- 失败：发生错误（如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见 异常处理。

请求示例

```
//1.创建媒体信息请求对象
MediaInfoRequest request = new MediaInfoRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("examplebucket-1250000000");
request.getInput().setObject("1.mp3");
//3.调用接口,获取媒体信息响应对象
MediaInfoResponse response = client.generateMediainfo(request);
```

版权所有：腾讯云计算（北京）有限责任公司

AI 内容识别
图片标签

最近更新时间：2022-10-10 15:03:59

简介

本文档提供关于图片标签的 API 概览和 SDK 示例代码。

API	操作描述
图片标签	对 COS 存量数据的图片标签识别

图片标签

功能说明

图片标签功能通过借助数据万象的持久化处理接口，实现对 COS 存量数据的图片标签识别，返回图片中置信度较高的主题标签。图片标签识别请求包属于 GET 请求，请求时需要携带签名。

方法原型

```
ImageLabelResponse getImageLabel(ImageLabelRequest request);
```

请求示例

```
//1.创建任务请求对象
ImageLabelRequest request = new ImageLabelRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("demo-123456789");
request.setObjectKey("1.png");
//3.调用接口,获取任务响应对象
ImageLabelResponse response = client.getImageLabel(request);
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

节点名称（关键字）	父节点	描述	类型	是否必选
bucketName	Request	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
objectKey	Request	图片文件在 bucket 中的位置	String	是

返回结果说明

- 成功：成功则返回 ImageLabelResponse 实例，内含结果内容。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见异常处理。

版权所有：腾讯云计算（北京）有限责任公司

汽车识别

最近更新时间：2022-10-10 15:03:59

简介

本文档提供关于汽车识别的 API 概览和 SDK 示例代码。

API	操作描述
车辆车牌检测	对图片中的车辆进行识别

车辆车牌检测

功能说明

车辆车牌检测功能为同步请求方式，您可以通过本接口检测图片中的车辆，识别出车辆的品牌、颜色、位置、车牌位置等信息。该接口属于 GET 请求。

方法原型

```
DetectCarResponse detectCar(DetectCarRequest request);
```

请求示例

```
//1.创建任务请求对象
DetectCarRequest request = new DetectCarRequest();
//2.添加请求参数 参数详情请见api接口文档
//2.1设置请求bucket
request.setBucketName("demobucket-1234567890");
//2.2设置bucket中的图片位置
request.setObjectKey("car.jpg");
DetectCarResponse response = client.detectCar(request);
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

节点名称（关键字）	父节点	描述	类型	是否必选
bucketName	Request	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
objectKey	Request	图片文件在 bucket 中的位置	String	是

返回结果说明

- 成功：成功则返回 DetectCarResponse 任务结果对象。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见异常处理。

以图搜图

最近更新时间：2022-10-19 14:24:36

简介

本文档提供关于以图搜图的 API 概览和 SDK 示例代码。

API	操作描述
开通以图搜图	该接口用于开通 Bucket 搜图功能
添加图库图片	该接口用于添加图库图片
图片搜索接口	该接口用于检索图库中的图片
删除图库图片	该接口用于删除图库图片

开通以图搜图

功能说明

该接口用于开通 Bucket 搜图功能。

方法原型

```
boolean openImageSearch(OpenImageSearchRequest imageSearchRequest);
```

请求示例

```
//1.创建任务请求对象
OpenImageSearchRequest request = new OpenImageSearchRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("demobucket-123456789");
request.setMaxCapacity("100");
request.setMaxQps("10");
//3.调用接口,获取任务响应对象
boolean response = client.openImageSearch(request);
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
bucketName	Request	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
maxCapacity	Request	图库容量限制	String	是
MaxQps	Request	图库容量限制	String	否

返回结果说明

- 成功：成功则返回 true。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见异常处理。

添加图库图片

功能说明

将图片资源添加至图库。

方法原型

```
boolean addGalleryImages(ImageSearchRequest imageSearchRequest);
```

请求示例

```
//1.创建任务请求对象
ImageSearchRequest request = new ImageSearchRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("demobucket-123456789");
request.setObjectKey("1.png");
request.setEntityId("mark1");
//3.调用接口,获取任务响应对象
boolean response = client.addGalleryImages(request);
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

节点名称（关键字）	父节点	描述	类型	是否必选
bucketName	Request	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
objectKey	Request	图片文件在 bucket 中的位置	String	是
entityId	Request	物品 ID，最多支持64个字符。若 EntityId 已存在，则对其追加图片	String	是
customContent	Request	用户自定义的内容，最多支持4096个字符，查询时原样带回	String	否
tags	Request	图片自定义标签，最多不超过10个，json 字符串，格式为 key:value 对	String	否

返回结果说明

- 成功：成功则返回 true。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见异常处理。

图片搜索接口

功能说明

检索图库中的图片。

方法原型

```
ImageSearchResponse searchGalleryImages(ImageSearchRequest imageSearchRequest);
```

请求示例

```
//1.创建任务请求对象
ImageSearchRequest request = new ImageSearchRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("demobucket-123456789");
request.setObjectKey("1.png");
//3.调用接口,获取任务响应对象
ImageSearchResponse response = client.searchGalleryImages(request);
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
bucketName	Request	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
objectKey	Request	图片文件在 bucket 中的位置	String	是
matchThreshold	Request	出参 Score 中，只有超过 MatchThreshold 值的结果才会返回。默认为0	String	否
Offset	Request	起始序号，默认值为0	String	否
limit	Request	返回数量，默认值为10，最大值为100	String	否
filter	Request	针对入库时提交的 Tags 信息进行条件过滤。支持>、>=、<、<=、=、!=，多个条件之间支持 AND 和 OR 进行连接	String	否

返回结果说明

- 成功：成功则返回 ImageSearchResponse 响应对象。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见异常处理。

删除图库图片

功能说明

将图片资源移出图库。

方法原型

```
boolean deleteGalleryImages(ImageSearchRequest imageSearchRequest);
```

请求示例

```
//1.创建任务请求对象
ImageSearchRequest request = new ImageSearchRequest();
//2.添加请求参数 参数详情请见api接口文档
request.setBucketName("demobucket-123456789");
```

```
request.setObjectKey("1.png");
request.setEntityId("mark");
//3.调用接口,获取任务响应对象
boolean response = client.deleteGalleryImages(request);
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
bucketName	Request	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
objectKey	Request	图片文件在 bucket 中的位置	String	是
entityId	Request	物品 ID	String	是

返回结果说明

- 成功：成功则返回 true。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。详情请参见异常处理。

版权所有：腾讯云计算（北京）有限责任公司

异常处理

最近更新时间：2022-01-05 14:34:04

简介

调用 SDK 请求 COS 服务失败时，抛出的异常皆是 RuntimeExcpetion，目前 SDK 常见的异常有 CosClientException，CosServiceException 和 IllegalArgumentException。

客户端异常

客户端异常 CosClientException，是由于客户端原因导致无法和服务端完成正常的交互而导致的失败，如客户端无法连接到服务端，无法解析服务端返回的数据，读取本地文件发生 IO 异常等。CosClientException 继承自 RuntimeException，没有自定义的成员变量，使用方法同 RuntimeException。

服务端异常

服务端异常 CosServiceException，用于指交互正常完成，但是操作失败的场景。例如客户端访问一个不存在 Bucket，删除一个不存在的文件，没有权限进行某个操作， 服务端故障异常等。CosServiceException 包含了服务端返回的状态码，requestid，出错明细等。捕获异常后，建议对整个异常进行打印，异常包含了必须的排查因素。以下是异常成员变量的描述：

request 成员	描述	类型
requestId	请求 ID， 用于表示一个请求， 对于排查问题十分重要	String
traceId	辅助排查问题的 ID，	String
statusCode	response 的 status 状态码， 4xx 是指请求因客户端而失败， 5xx 是服务端异常导致的失败。 请参见 COS 错误码	String
errorType	枚举类， 表示异常的种类， 分为 Client， Service， Unknown	ErrorType
errorCode	请求失败时 body 返回的 Error Code 请参见 COS 错误码	String
errorMessage	请求失败时 body 返回的 Error Message 请参见 COS 错误码	String

使用自助诊断工具

版权所有：腾讯云计算（北京）有限责任公司

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。
   COS 自助诊断工具
   点击自助诊断
   输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**。
4. 稍候片刻，便能看到相应的智能诊断结果。

常见问题

若您在使用 Java SDK 过程中，有相关的疑问，请参见 常见问题 文档的 Java SDK 部分。

版权所有：腾讯云计算（北京）有限责任公司

设置访问域名（CDN/全球加速）

最近更新时间：2022-05-09 14:28:43

简介

本文档提供关于如何使用非默认域名请求 COS 服务。

默认 CDN 加速域名

功能说明

在 COS 控制台为存储桶开启默认 CDN 加速域名后，可在 SDK 代码中设置对应的域名，以便实现默认加速功能。关于如何开启默认加速域名请参见 开启默认 CDN 加速域名。

方法原型

```
void com.qcloud.cos.ClientConfig.setEndPointSuffix(String endPointSuffix)
```

请求示例

```
// 1 使用 CDN 加速域名，无需再发送 COS 的认证信息
COSCredentials cred = new AnonymousCOSCredentials();
// 2 设置 bucket 的地域简称，COS 地域简称请参照 https://intl.cloud.tencent.com/documen
t/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing"));
// 推荐使用 https 协议
clientConfig.setHttpProtocol(HttpProtocol.https);
// 3 设置默认 CDN 加速域名后缀
String cdnSuffix = "file.myqcloud.com";
clientConfig.setEndPointSuffix(cdnSuffix);
// 4 生成 cos 客户端
COSClient cosclient = new COSClient(cred, clientConfig);
```

参数说明

参数名称	描述	类型
endPointSuffix	通过设置域名后缀，使用 CDN 加速域名	String

自定义 CDN 加速域名

功能说明

版权所有：腾讯云计算（北京）有限责任公司

在 COS 控制台为存储桶设置自定义 CDN 加速域名后，可在 SDK 代码中设置对应的域名，以便实现自定义域名的加速功能。关于如何开启自定义 CDN 加速域名，请参见 开启自定义 CDN 加速域名。

方法原型

```
void com.qcloud.cos.ClientConfig.setEndpointBuilder(EndpointBuilder endpointBuild
er)
```

请求示例

```
// 1 使用 CDN 加速域名，无需再发送 COS 的认证信息
COSCredentials cred = new AnonymousCOSCredentials();
// 2 设置 bucket 的地域简称，COS 地域简称请参照 https://intl.cloud.tencent.com/documen
t/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing"));
// 推荐使用 https 协议，如果使用 https 协议，请确保自定义域名有相关 https 证书
clientConfig.setHttpProtocol(HttpProtocol.https);
// 3 设置 CDN 自定义域名
// get service 请求会使用这个域名，这个域名不能自定义
String serviceApiEndpoint = "service.cos.myqcloud.com";
String cdnEndpoint = "xxx.xxx.com";
UserSpecifiedEndpointBuilder endPointBuilder = new UserSpecifiedEndpointBuilder(c
dnEndpoint, serviceApiEndpoint);
clientConfig.setEndpointBuilder(endPointBuilder);
// 4 生成 cos 客户端
COSClient cosclient = new COSClient(cred, clientConfig);
```

参数说明

参数名称	描述	类型
endPointerBuilder	参数用于构建自定义 CDN 加速域名	UserSpecifiedEndpointBuilder

UserSpecifiedEndpointBuilder 说明：

参数名称	描述	类型
generalApiEndpoint	填写用户的自定义 CDN 加速域名	String
getServiceApiEndpoint	只能填写 `service.cos.myqcloud.com`	String

自定义源站域名

功能说明

在 COS 控制台为存储桶设置自定义源站域名后，可在 SDK 代码中设置对应的域名，以便实现访问 COS 资源。关于如何设置自定义源站域名请参见 自定义源站域名。

方法原型

```
void com.qcloud.cos.ClientConfig.setEndpointBuilder(EndpointBuilder endpointBuild
er)
```

请求示例

```
// 1 初始化用户身份信息(secretId, secretKey)
// SECRETID和SECRETKEY请登录访问管理控制台进行查看和管理
String secretId = "SECRETID";
String secretKey = "SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 2 设置 bucket 的地域简称, COS 地域简称请参照 https://intl.cloud.tencent.com/documen
t/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing"));
// 推荐使用 https 协议
clientConfig.setHttpProtocol(HttpProtocol.https);
// 3 设置自定义域名
// get service 请求会使用这个域名, 这个域名不能自定义
String serviceApiEndpoint = "service.cos.myqcloud.com";
String userEndpoint = "xxx.xxx.com";
UserSpecifiedEndpointBuilder endPointBuilder = new UserSpecifiedEndpointBuilder(u
serEndpoint, serviceApiEndpoint);
clientConfig.setEndpointBuilder(endPointBuilder);
// 4 生成 cos 客户端
COSClient cosclient = new COSClient(cred, clientConfig);
```

参数说明

参数名称	描述	类型
endPointerBuilder	参数用于构建自定义源站域名	UserSpecifiedEndpointBuilder

UserSpecifiedEndpointBuilder 说明：

参数名称	描述	类型
generalApiEndpoint	填写自定义源站域名	String
getServiceApiEndpoint	只能填写 `service.cos.myqcloud.com`	String

版权所有：腾讯云计算（北京）有限责任公司

全球加速域名

功能说明

在 COS 控制台为存储桶开启全球加速域名后，可在 SDK 代码中设置对应的域名，以便实现全球加速功能。关于全球加速的功能介绍，请参见 全球加速功能概述。

方法原型

```
void com.qcloud.cos.ClientConfig.setEndPointSuffix(String endPointSuffix)
```

请求示例

```
// 1 初始化用户身份信息(secretId, secretKey)
// SECRETID和SECRETKEY请登录访问管理控制台进行查看和管理
String secretId = "SECRETID";
String secretKey = "SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 2 设置 bucket 的地域简称, COS 地域简称请参照 https://intl.cloud.tencent.com/documen
t/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing"));
// 推荐使用 https 协议
clientConfig.setHttpProtocol(HttpProtocol.https);
// 3 设置全球加速域名后缀
String cosSuffix = "cos.accelerate.myqcloud.com";
clientConfig.setEndPointSuffix(cosSuffix);
// 4 生成 cos 客户端
COSClient cosclient = new COSClient(cred, clientConfig);
```

参数说明

参数名称	描述	类型
endPointSuffix	通过设置域名后缀，使用全球加速域名	String

版权所有：腾讯云计算（北京）有限责任公司

JavaScript SDK 快速入门

最近更新时间：2023-02-23 16:36:33

下载与安装

相关资源

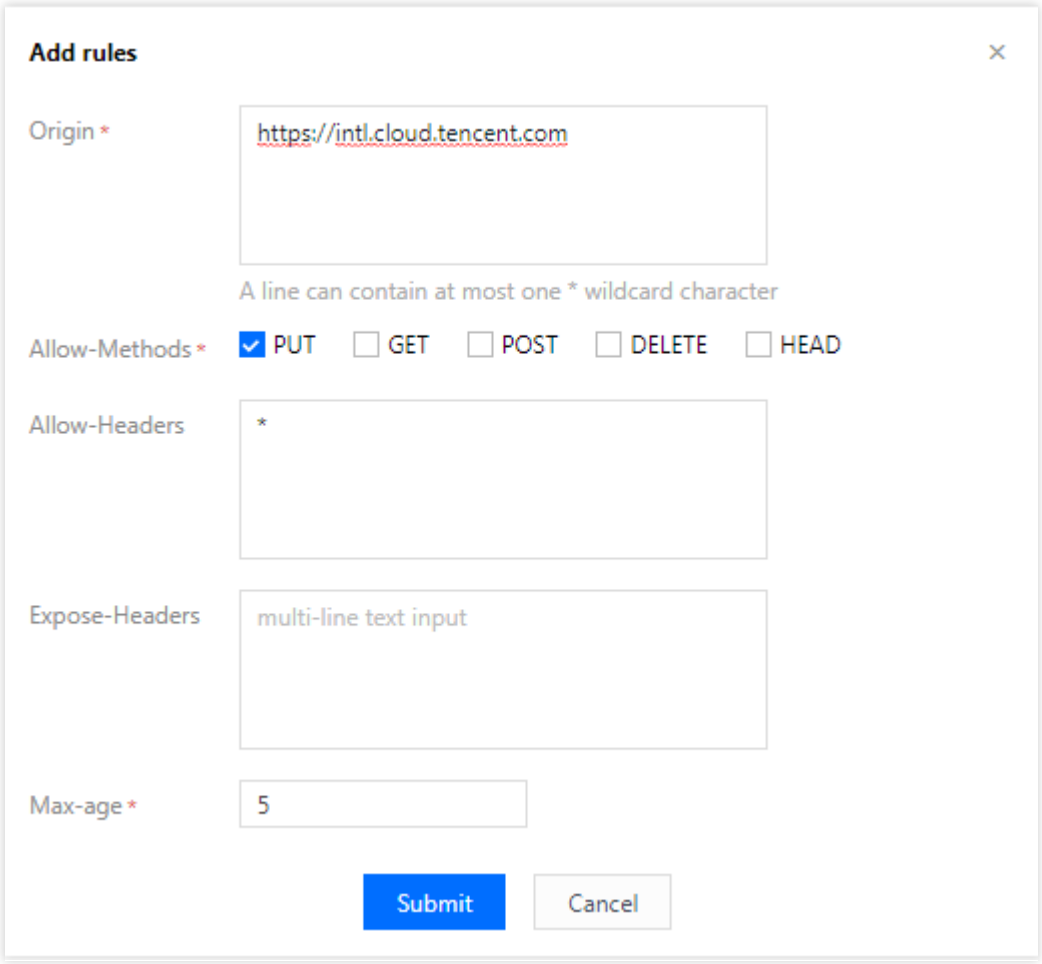
- 对象存储 COS 的 XML JS SDK 源码下载地址：XML JavaScript SDK。
- SDK 快速下载地址：XML JavaScript SDK。
- 演示示例 Demo 下载地址：XML JavaScript SDK Demo。
- SDK 文档中的所有示例代码请参见 SDK 代码示例。
- SDK 更新日志请参见 ChangeLog。
- SDK 常见问题请参见：JavaScript SDK 常见问题。

> 说明：
> 如果您在使用 XML 版本 SDK 时遇到函数或方法不存在等错误，请先将 XML 版本 SDK 升级到最新版再重试。

准备环境

1. JavaScript SDK 需浏览器支持基本的 HTML5 特性（支持 IE10 以上浏览器），以便支持 ajax 上传文件和计算文件 MD5 值。
2. 登录 对象存储控制台，创建存储桶。获取存储桶名称和 地域名称。
3. 登录 访问管理控制台，获取您的项目 SecretId 和 SecretKey。
4. 配置 CORS 规则，AllowHeader 需配成 `*`，ExposeHeaders 需要 ETag、Content-Length 以及其他 js 需要读取的 header 字段，如下图所示。操作详情请参见 设置跨域访问 文档。

版权所有：腾讯云计算（北京）有限责任公司



说明：

- 关于本文中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参见 COS 术语信息。
- 关于跨端框架（例如 uni-app）的使用说明，使用 JavaScript SDK 开发后无法打包成正常使用的移动应用，例如 Android App、iOS App，需要使用对应的 Android SDK、iOS SDK。

安装 SDK

您可以通过以下方式安装 SDK：

script 引入

说明：
建议将sdk下载到本地后引入，选择 地址一或 地址二 下载最新 cos-js-sdk-v5.min.js。

```
<!--src为本地路径 根据自己项目目录结构来调整-->
<script src="../dist/cos-js-sdk-v5.min.js"></script>
```

在 script 标签引用 SDK 时，SDK 占用了全局变量名 COS，通过它的构造函数可以创建 SDK 实例。

webpack 引入方式

通过 `npm i cos-js-sdk-v5 --save` 安装 SDK 依赖，支持 webpack 打包的场景，您可以用 npm 引入作为模块，代码如下:

```
var COS = require('cos-js-sdk-v5');

// 或

import COS from 'cos-js-sdk-v5';
```

开始使用

> 注意：
>
> - 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
> - 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

获取临时密钥

由于固定密钥放在前端会有安全风险，正式部署时我们推荐使用临时密钥的方式，实现过程为：前端首先请求服务端，服务端使用固定密钥调用 STS 服务申请临时密钥（具体内容请参见 临时密钥生成和使用指引 文档），然后返回临时密钥到前端使用。

> 注意：
> 如果站点有登录态，这个获取临时密钥接口，还需要加上登录态校验。

初始化

1. 创建 web.html，填入如下代码，并修改 web.html 中的存储桶名称和 Region。
2. 部署好后端的临时密钥服务，并修改 getAuthorization 里的密钥服务地址。
3. 把 web.html 放在 Web 服务器下，然后在浏览器访问页面，测试文件上传。web.html 文件示例代码如下：

```
<input id="file-selector" type="file">
<script src="dist/cos-js-sdk-v5.min.js"></script>
<script>

// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
var Bucket = 'examplebucket-1250000000'; /* 存储桶，必须字段 */

// 存储桶 region 可以在 COS 控制台指定存储桶的概览页查看 https://console.intl.cloud.tencent.com/cos5/bucket/
// 关于地域的详情见 https://intl.cloud.tencent.com/document/product/436/6224
var Region = 'COS_REGION'; /* 存储桶所在地域，必须字段 */

// 初始化实例
var cos = new COS({
// getAuthorization 必选参数
getAuthorization: function (options, callback) {
// 初始化时不会调用，只有调用 cos 方法（例如 cos.putObject）时才会进入
// 异步获取临时密钥
// 服务端 JS 和 PHP 例子：https://github.com/tencentyun/cos-js-sdk-v5/blob/master/server/
// 服务端其他语言参考 COS STS SDK ：https://github.com/tencentyun/qcloud-cos-sts-sdk
// STS 详细文档指引看：https://intl.cloud.tencent.com/document/product/436/14048

var url = 'http://example.com/server/sts.php'; // url 替换成您自己的后端服务
var xhr = new XMLHttpRequest();
xhr.open('GET', url, true);
xhr.onload = function (e) {
try {
var data = JSON.parse(e.target.responseText);
var credentials = data.credentials;
} catch (e) {
}
if (!data || !credentials) {
return console.error('credentials invalid:\n' + JSON.stringify(data, null, 2))
};
callback({
TmpSecretId: credentials.tmpSecretId,
TmpSecretKey: credentials.tmpSecretKey,
SecurityToken: credentials.sessionToken,
// 建议返回服务器时间作为签名的开始时间，避免用户浏览器本地时间偏差过大导致签名错误
StartTime: data.startTime, // 时间戳，单位秒，如：1580000000
ExpiredTime: data.expiredTime, // 时间戳，单位秒，如：1580000000
});
};
xhr.send();
```

```
}
});

// 接下来可以通过 cos 实例调用 COS 请求。

</script>
```

配置项

使用示例

创建一个 COS SDK 实例，COS SDK 支持以下格式创建：

- 格式一（推荐）：后端通过获取临时密钥给到前端，前端计算签名。

```
var COS = require('cos-js-sdk-v5');
var cos = new COS({
// getAuthorization 必选参数
getAuthorization: function (options, callback) {
// 异步获取临时密钥
// 服务端 JS 和 PHP 例子：https://github.com/tencentyun/cos-js-sdk-v5/blob/master/server/
// 服务端其他语言参考 COS STS SDK ：https://github.com/tencentyun/qcloud-cos-sts-sdk
// STS 详细文档指引看：https://intl.cloud.tencent.com/document/product/436/14048

var url = 'http://example.com/server/sts.php'; // url 替换成您自己的后端服务
var xhr = new XMLHttpRequest();
xhr.open('GET', url, true);
xhr.onload = function (e) {
try {
var data = JSON.parse(e.target.responseText);
var credentials = data.credentials;
} catch (e) {
}
if (!data || !credentials) {
return console.error('credentials invalid:\n' + JSON.stringify(data, null, 2))
};
callback({
TmpSecretId: credentials.tmpSecretId,
TmpSecretKey: credentials.tmpSecretKey,
SecurityToken: credentials.sessionToken,
// 建议返回服务器时间作为签名的开始时间，避免用户浏览器本地时间偏差过大导致签名错误
StartTime: data.startTime, // 时间戳，单位秒，如：1580000000
ExpiredTime: data.expiredTime, // 时间戳，单位秒，如：1580000000
});
};
```

```
xhr.send();
}
});
```

- 格式二（推荐）：细粒度控制权限，后端通过获取临时密钥给到前端，只有在相同请求时，前端才重复使用临时密钥，后端可以通过 Scope 细粒度控制权限。

```
var COS = require('cos-js-sdk-v5');
var cos = new COS({
// getAuthorization 必选参数
getAuthorization: function (options, callback) {
// 服务端例子：https://github.com/tencentyun/qcloud-cos-sts-sdk/blob/master/scope.md
// 异步获取临时密钥
var url = 'http://example.com/server/sts.php'; // url 替换成您自己的后端服务
var xhr = new XMLHttpRequest();
xhr.open('POST', url, true);
xhr.setRequestHeader('Content-Type', 'application/json');
xhr.onload = function (e) {
try {
var data = JSON.parse(e.target.responseText);
var credentials = data.credentials;
} catch (e) {
}
if (!data || !credentials) {
return console.error('credentials invalid:\n' + JSON.stringify(data, null, 2))
};
callback({
TmpSecretId: credentials.tmpSecretId,
TmpSecretKey: credentials.tmpSecretKey,
SecurityToken: credentials.sessionToken,
// 建议返回服务器时间作为签名的开始时间，避免用户浏览器本地时间偏差过大导致签名错误
StartTime: data.startTime, // 时间戳，单位秒，如：1580000000
ExpiredTime: data.expiredTime, // 时间戳，单位秒，如：1580000000
ScopeLimit: true, // 细粒度控制权限需要设为 true，会限制密钥只在相同请求时重复使用
});
};
xhr.send(JSON.stringify(options.Scope));
}
});
```

- 格式三（不推荐）：前端每次请求前都需要通过 getAuthorization 获取签名，后端使用固定密钥或临时密钥计算签名返回至前端。该格式分块上传权限不便控制，不推荐您使用此格式。

版权所有：腾讯云计算（北京）有限责任公司

```
var COS = require('cos-js-sdk-v5');
var cos = new COS({
// getAuthorization 必选参数
getAuthorization: function (options, callback) {
// 异步获取签名

var url = 'http://example.com/server/sts.php'; // url 替换成您自己的后端服务
var method = (options.Method || 'get').toLowerCase();
var query = options.Query || {};
var headers = options.Headers || {};
var pathname = options.Pathname || '/';
var xhr = new XMLHttpRequest();
var data = {
method: method,
pathname: pathname,
query: query,
headers: headers,
};
xhr.open('POST', url, true);
xhr.setRequestHeader('content-type', 'application/json');
xhr.onload = function (e) {
try {
var data = JSON.parse(e.target.responseText);
} catch (e) {
}
if (!data || !data.authorization) return console.error('authorization invalid');
callback({
Authorization: data.authorization,
// SecurityToken: data.sessionToken, // 如果使用临时密钥，需要把 sessionToken 传给 Se
curityToken
});
};
xhr.send(JSON.stringify(data));
},
});
```

- 格式四（不推荐）：前端使用固定密钥计算签名，该格式适用于前端调试，若使用此格式，请避免泄露密钥。

```
var COS = require('cos-js-sdk-v5');

// SECRETID 和 SECRETKEY 请登录 https://console.intl.cloud.tencent.com/cam/capi 进
行查看和管理
var cos = new COS({
SecretId: 'SECRETID',
```

```
SecretKey: 'SECRETKEY',
});
```

构造函数参数说明

参数名	参数描述	类型	是否必填
SecretId	用户的 SecretId	String	否
SecretKey	用户的 SecretKey，建议只在前端调试时使用，避免暴露密钥	String	否
FileParallelLimit	同一个实例下上传的文件并发数，默认值3	Number	否
ChunkParallelLimit	同一个上传文件的分块并发数，默认值3	Number	否
ChunkRetryTimes	分块上传及分块复制时，出错重试次数，默认值2（加第一次，请求共3次）	Number	否
ChunkSize	分块上传时，每片的字节数大小，默认值1048576（1MB）	Number	否
SliceSize	使用 uploadFiles 批量上传时，文件大小大于该数值将使用按分块上传，否则将调用简单上传，单位 Byte，默认值1048576（1MB）	Number	否
CopyChunkParallelLimit	进行分块复制操作中复制分块上传的并发数，默认值20	Number	否
CopyChunkSize	使用 sliceCopyFile 分块复制文件时，每片的大小字节数，默认值10485760（10MB）	Number	否
CopySliceSize	使用 sliceCopyFile 分块复制文件时，文件大小大于该数值将使用分块复制，否则将调用简单复制，默认值10485760（10MB）	Number	否
ProgressInterval	上传进度的回调方法 onProgress 的回调频率，单位 ms，默认值1000	Number	否
Protocol	发请求时用的协议，可选项 `https:`、`http:`，默认判断当前页面是 `http:` 时使用 `http:`，否则使用 `https:`	String	否
Domain	调用操作存储桶和对象的 API 时自定义请求域名。可以使用模板，如 `"{Bucket}.cos.{Region}.myqcloud.com"`，即在调用 API 时会使用参数中传入的 Bucket 和 Region 进行替换。	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
UploadQueueSize	上传队列最长大小，超出的任务如果状态不是 waiting、checking、uploading 会被清理，默认10000	Number	否
ForcePathStyle	强制使用后缀式模式发请求。后缀式模式中 Bucket 会放在域名后的 pathname 里，并且 Bucket 会加入签名 pathname 计算，默认 false	Boolean	否
UploadCheckContentMd5	上传文件时校验 Content-MD5，默认 false。如果开启，上传文件时会对文件内容计算 MD5，大文件耗时较长	Boolean	否
getAuthorization	获取签名的回调方法，如果没有 SecretId、SecretKey 时，这个参数必选。 **注意: 该回调方法在初始化实例时传入，在使用实例调用接口时才会执行并获取签名。**	Function	否
Timeout	超时时间，单位毫秒，默认为0，即不设置超时时间	Number	否
UseAccelerate	是否启用全球加速域名，默认为 false。若改为 true，需要存储桶开启全球加速功能，详情请参见 开启全球加速。	Boolean	否

getAuthorization 回调函数说明的函数说明（使用格式一）

```
getAuthorization: function(options, callback) { ... }
```

getAuthorization 的回调参数说明：

参数名	参数描述	类型
options	获取临时密钥需要的参数对象	Object
- Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
- Region	存储桶所在地域，枚举值请参见 存储桶地域信息	String
callback	临时密钥获取完成后的回传方法	Function

获取完临时密钥后，callback 回传一个对象，回传对象的属性列表如下：

属性名	参数描述	类型	是否必填

属性名	参数描述	类型	是否必填
TmpSecretId	获取回来的临时密钥的 tmpSecretId	String	是
TmpSecretKey	获取回来的临时密钥的 tmpSecretKey	String	是
SecurityToken	获取回来的临时密钥的 sessionToken，对应 header 的 x-cos-security-token 字段	String	是
StartTime	密钥获取的开始时间，即获取时刻的时间戳，单位秒，startTime，如：1580000000，用于签名开始时间，传入该参数可避免前端时间偏差签名过期问题	String	否
ExpiredTime	获取回来的临时密钥的 expiredTime，超时时刻的时间戳，单位秒，如：1580000900	String	是

getAuthorization 回调函数说明（使用格式二）

```
getAuthorization: function(options, callback) { ... }
```

getAuthorization 的函数说明回调参数说明：

参数名	参数描述	类型
options	获取签名需要的参数对象	Object
- Method	当前请求的 Method	String
- Pathname	请求路径，用于签名计算	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，了解更多可参见 对象概述。 **注意:当使用实例请求的接口不是对象操作相关接口时，该参数为空。**	String
- Query	当前请求的 query 参数对象，{key: 'val'} 的格式	Object
- Headers	当前请求的 header 参数对象，{key: 'val'} 的格式	Object
callback	临时密钥获取完成后的回调	Function

getAuthorization 计算完成后，callback 回传参数支持两种格式：
格式一：回传鉴权凭证字符串 Authorization。
格式二：回传一个对象，对象属性列表如下：

属性名	参数描述	类型	是否必填
Authorization	计算得到的签名字符串	String	是
SecurityToken	获取回来的临时密钥的 sessionToken，对应 header 的 x-cos-security-token 字段	String	否

获取鉴权凭证

实例本身鉴权凭证可以通过实例化时传入的参数控制如何或获取，有三种获取方式：

1. 实例化时，传入 SecretId、SecretKey，每次需要签名都由实例内部计算。
2. 实例化时，传入 getAuthorization 回调，每次需要签名通过这个回调计算完，返回签名至实例。
3. 实例化时，传入 getAuthorization 回调，调用回调时返回临时密钥凭证，在临时密钥凭证过期后会再次调用该回调。

开启 beacon 上报

为了持续跟踪和优化 SDK 的质量，给您带来更好的使用体验，我们在 SDK 中引入了 腾讯灯塔 SDK。

> 说明：
> 腾讯灯塔只对 COS 侧的请求性能进行监控，不会上报业务侧数据。

若是想开启该功能，请先确保 SDK 版本升级到1.4.0及以上，然后在初始化中指定 EnableTracker 为 true。

```
new COS({
EnableTracker: true,
})
```

使用方式

回调方式

文档里默认使用回调方式，使用代码如下:

```
// 这里省略初始化过程和上传参数
var cos = new COS({ ... });
cos.uploadFile({ ... }, function(err, data) {
if (err) {
console.log('上传出错', err);
} else {
console.log('上传成功', data);
```

```
}
});
```

Promise

SDK 同样支持 Promise 方式调用，例如上述回调方式的代码等同于以下代码:

```
// 这里省略初始化过程和上传参数
var cos = new COS({ ... });
cos.uploadFile({ ... }).then(data => {
console.log('上传成功', data);
}).catch(err => {
console.log('上传出错', err);
});
```

同步方式

同步方式基于 JavaScript 的 async 和 await(使用时请注意浏览器兼容性)，上述回调方式的代码等同于以下代码:

```
async function upload() {
// 这里省略初始化过程和上传参数
var cos = new COS({ ... });
try {
var data = await cos.uploadFile({ ... });
return { err: null, data: data }
} catch (err) {
return { err: err, data: null };
}
}
// 可以同步拿到请求的返回值,这里举例说明,实际返回的数据格式可以自定义
var uploadResult = await upload();
if (uploadResult.err) {
console.log('上传出错', uploadResult.err);
} else {
console.log('上传成功', uploadResult.data);
}
```

注意：

cos.getObjectUrl 目前只支持回调方式。

使用技巧

通常情况下我们只需要创建一个 COS SDK 实例，然后在需要调用 SDK 方法的地方直接使用这个实例即可，示例代码如下：

```
/* vue 项目举例 */

/* 新建一个 cos.js，导出 cos 实例 */
import COS from 'cos-js-sdk-v5'; // 通过 npm 安装的 SDK
const cos = new COS({
....
});
export default cos;

/* 单页面里 page.vue */
<template>
<input id="fileSelector" type="file" @change="upload" />
</template>
<script>
/* 引入上方新建的 cos.js路径 */
import cos from 'cos';
export default {
data() {},
methods: {
upload(e) {
const file = e.target.files && e.target.files[0];
/* 直接调用 cos sdk 的方法 */
cos.uploadFile({
Bucket: 'examplebucket-1250000000', /* 填写自己的 bucket，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: file, // 上传文件对象
SliceSize: 1024 * 1024 * 5, /* 触发分块上传的阈值，超过5MB 使用分块上传，小于5MB使用简单上传。可自行设置，非必须 */
});
}
},
}
</script>
```

以下是部分常用接口例子，更详细的初始化方法请参见 demo 示例。

查询对象列表

```
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填写自己的 bucket，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Prefix: 'a/', /* 列出目录 a 下所有文件，非必须 */
```

```
}, function(err, data) {
console.log(err || data.Contents);
});
```

上传对象

强烈推荐使用高级上传接口 uploadFile，自动针对小文件使用简单上传，大文件使用分块上传，性能更好。详情请参见 高级上传 文档。
若使用临时密钥方式，需同时授权 `简单上传对象` 和 `分块上传` 的权限。请参考授权指引。
常见上传错误排查，请参考 常见问题。

```
<!-- html 页面 DOM 元素 -->

<!-- 选择要上传的文件 -->
<input id="fileSelector" type="file" />
<!-- 点击按钮上传 -->
<input id="submitBtn" type="submit" />
```

```
function handleFileInUploading(file) {
cos.uploadFile({
Bucket: 'examplebucket-1250000000', /* 填写自己的 bucket，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如:1.jpg，a/b/test.txt，图片.jpg）支持中文，必须字段 */
Body: file, // 上传文件对象
SliceSize: 1024 * 1024 * 5, /* 触发分块上传的阈值，超过5MB使用分块上传，小于5MB使用简单上传。可自行设置，非必须 */
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
if (err) {
console.log('上传失败', err);
} else {
console.log('上传成功');
}
});
}

/* 选择文件 */
document.getElementById('submitBtn').onclick = function (e) {
var file = document.getElementById('fileSelector').files[0];
if (!file) {
document.getElementById('msg').innerText = '未选择上传文件';
return;
```

```
}
handleFileInUploading(file);
};
```

下载对象

浏览器里通过生成预签名 url 并触发浏览器下载的方式实现

```
cos.getObjectUrl({
Bucket: 'examplebucket-1250000000', /* 填写自己的 bucket，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
if (err) return console.log(err);
/* 通过指定 response-content-disposition=attachment 实现强制下载 */
var downloadUrl = data.Url + (data.Url.indexOf('?') > -1 ? '&' : '?') + 'response
-content-disposition=attachment';
/* 可拼接 filename 来实现下载时重命名 */
/* downloadUrl += ';filename=myname'; */
// （推荐使用 window.open()方式）这里是新窗口打开 url，如果需要在当前窗口打开，可以使用隐藏的
iframe 下载，或使用 a 标签 download 属性协助下载
window.open(downloadUrl);
});
```

删除对象

```
cos.deleteObject({
Bucket: 'examplebucket-1250000000', /* 填写自己的 bucket，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

版权所有：腾讯云计算（北京）有限责任公司

存储桶操作

最近更新时间：2022-07-07 14:19:35

简介

本文档提供关于存储桶操作的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

检索存储桶及其权限

功能说明

HEAD Bucket 请求可以确认该存储桶是否存在，是否有权限访问。有以下几种情况：

- 当该存储桶存在且有读取权限时，返回 HTTP 状态码200。
- 当该存储桶无读取权限时，返回 HTTP 状态码403。
- 当该存储桶不存在时，返回 HTTP 状态码404。

使用示例

检索存储桶信息：

```
cos.headBucket({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

判断存储桶是否存在：

```
function doesBucketExist() {
cos.headBucket({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
```

```
if (data) {
console.log('存储桶存在');
} else if (err.statusCode == 403) {
console.log('没有该存储桶读权限');
}
});
}
```

参数说明

参数名	参数描述	类型	必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

删除存储桶

功能说明

删除指定账号下的空存储桶。注意，如果删除成功，则返回的 HTTP 状态码为200或204。

> 说明：

删除存储桶前，请确保存储桶内的数据和未完成上传的分块数据已全部清空，否则会无法删除存储桶。

使用示例

```
cos.deleteBucket({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域。枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

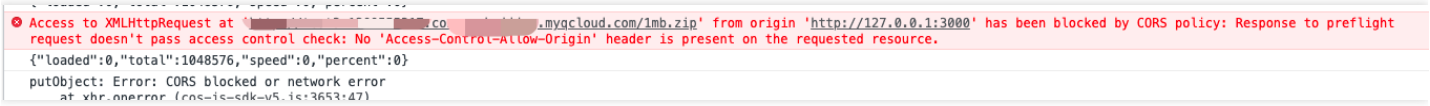
版权所有：腾讯云计算（北京）有限责任公司

JavaScript SDK 常见问题

最近更新时间：2023-02-07 15:06:52

上传时报错 cors error，该如何处理？

报错如下图：



原因是存储桶的跨域访问没有正确设置，请 参考文档 进行跨域设置。

使用临时密钥操作时报错 403，该如何处理？

请检查申请临时密钥时填写的 action 和 allowPrefix 是否正确。

1. 例如调用 cos.putObject()，但是 action 里并没有填写**name/cos:PutObject**，即没有 putObject 权限导致报错 403。
2. 例如操作的 Key 是 `1.jpg`，但是 allowPrefix 填写的是 `test/*`（只允许操作 `test/*` 路径），即没有对应路径的操作权限导致报错 403。

若 aciton 和 allowPrefix 都正确，请参考 临时密钥生成及使用指引 和 访问 COS 时返回 403 错误码。
字段说明：不同语言的 STS SDK，action 和 allowPrefix 所使用的字段不同，例如 STS JAVA SDK 使用的是 allowActions 和 allowPrefixes 字段，请注意留意 STS SDK 中的示例。

JavaScript SDK 报错请求过期：Request has expired (Status Code: 403; Error Code: AccessDenied)，该如何处理？

由于签名过期导致，重新生成签名即可解决；若重新生成签名仍报相同的错误，可以再检查机器的本地时间是否为标准的本地时间。

使用 JavaScript SDK 实现分块上传时，第一次上传成功后，后续请求都是 403，该如何处理？

请求出现 403 报错，可参考 访问 COS 时返回 403 错误码 文档进行排查。如果您使用临时密钥进行分块上传，建议您检查是否被授予 分块上传权限，以及授权路径是否正确。

JavaScript SDK 上传速度达不到满载带宽，该如何处理？

可以使用分块上传接口，通过增加每个分块的大小解决。例如当前设置每个分块大小为 1MB，您可以适当调整分块大小为 5MB 或者其他大小，观察带宽使用情况，详情请参见 分块操作。

JavaScript SDK 如何获取文件上传进度？

JavaScript SDK 的简单上传对象接口和分块上传对象接口会返回进度，详情请参考 分块操作。

版权所有：腾讯云计算（北京）有限责任公司

JavaScript SDK 的 List Multipart Uploads 能否直接获取进度？

当前 List Multipart Uploads 接口需要使用回调函数来返回，暂时无法直接获取。

版权所有：腾讯云计算（北京）有限责任公司

对象操作
上传对象

最近更新时间：2022-12-14 15:24:35

简介

本文档提供关于对象的简单操作、分块操作的 API 概览以及 SDK 示例代码，并举例如何使用。

> 说明：
> 常见上传错误排查，请参考 常见问题。

简单操作

API	操作名	操作描述
PUT Object	简单上传对象	上传一个对象至存储桶
Append Object	追加上传对象	将对象以分块追加的方式上传至存储桶
POST Object	表单上传对象	使用表单请求上传对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传任务
Upload Part	上传分块	分块上传对象
List Parts	查询已上传块	查询特定分块上传操作中已上传的块
Complete Multipart Upload	完成分块上传	完成整个对象的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

简单操作

版权所有：腾讯云计算（北京）有限责任公司

简单上传对象

功能说明

PUT Object 接口可以上传一个对象至指定存储桶中。该操作需要请求者对存储桶有 WRITE 权限。

注意：

- Key（文件名）不能以 `/` 结尾，否则会被识别为文件夹。
- Key（文件名）同名上传默认为覆盖操作。若您未开启版本控制且不想覆盖云上文件时，请确保上传时的Key不重复。
- 每个主账号（即同一个 APPID），存储桶的 ACL 规则数量最多为1000条，对象 ACL 规则数量不限制。如果您不需要进行对象 ACL 控制，请在上传时不要设置，默认继承存储桶权限。
- 上传之后，您可以用同样的 Key 生成预签名链接（下载请指定 method 为 GET，具体接口说明见下文，分享到其他端来进行下载。但注意如果您的文件是私有读权限，那么预签名链接只有一定的有效期。
- 上传进度onProgress依赖于原生的xhr.upload.onprogress方法，如果您发现上传进度不准确，请检查您的项目内是否引用了拦截XHR方法的库，例如 nuysoft/Mock。

使用示例

简单上传文件，适用于小文件上传：

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: fileObject, /* 必须，上传文件对象，可以是input[type="file"]标签选择本地文件后得到的file对象 */
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

上传字符串作为文件内容：

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.txt', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: 'hello!',
}, function(err, data) {
```

```
console.log(err || data);
});
```

上传base64作为文件内容：

```
var base64Url = '.....';
var dataURLtoBlob = function (dataurl) {
var arr = dataurl.split(',');
var mime = arr[0].match(/:(.*?);/)[1];
var bstr = atob(arr[1]);
var n = bstr.length;
var u8arr = new Uint8Array(n);
while (n--) {
u8arr[n] = bstr.charCodeAt(n);
}
return new Blob([u8arr], { type: mime });
};
// 需要转为Blob上传
var body = dataURLtoBlob(base64Url);
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.png', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: body,
}, function(err, data) {
console.log(err || data);
});
```

创建目录a：

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'a/', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: '',
}, function(err, data) {
console.log(err || data);
});
```

上传文件到指定目录a/b：

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'a/b/1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: fileObject, /* 必须，上传文件对象，可以是input[type="file"]标签选择本地文件后得到的
```

版权所有：腾讯云计算（北京）有限责任公司

```
file对象 */
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

上传对象（单链接限速）：

> 说明：
> 关于上传对象的限速说明，请参见 单链接限速。

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: fileObject, /* 必须，上传文件对象，可以是input[type="file"]标签选择本地文件后得到的file对象 */
Headers: {
'x-cos-traffic-limit': 819200, // 限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，如果超出该范围将返回400错误。
},
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
Body	上传文件的内容，可以为字符串，File 对象或者 Blob 对象	String\Fil

参数名	参数描述	类型
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，例如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String
GrantRead	赋予被授权者读取对象的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantReadAcp	赋予被授权者读取对象的访问控制列表（ACL）的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
GrantWriteAcp	赋予被授权者写入对象的访问控制列表（ACL）的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如`'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如`'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，默认值：STANDARD。更多存储类型请参见 存储类型概述	String
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据保存。大小限制2KB	String
UploadAddMetaMd5	当上传时，给对象的元数据信息增加 x-cos-meta-md5 赋值为对象内容的 MD5 值，格式为32位小写字符串。例如`4d00d79b6733c9cc066584a02ed03410`	String
onTaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function
- taskId	上传任务的编号	String
onProgress	进度的回调函数，进度回调响应对象（progressData）	Function
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- progressData.percent	文件上传的百分比，以小数形式呈现，例如，上传50%即为0.5	Number

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String
- Location	上传完的文件访问地址	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数。需要Expose-Headers里设置VersionId字段，参考文档	String

追加上传对象

功能说明

将对象以分块追加的方式上传至存储桶（APPEND Object）。

注意：

- COS Javascript SDK 版本需要大于等于 v1.3.1。

- 对象属性为 appendable 时才能使用本接口追加上传。
- 对象首次使用 APPEND Object 接口上传时，该对象的属性自动为 appendable。
- 可以使用 GET Object 或 HEAD Object 接口获取 x-cos-object-type 响应头来判断对象属性。

使用示例

初次追加上传对象:

```
cos.appendObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'test.txt', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: fileObject, /* 必须，上传文件对象，可以是input[type="file"]标签选择本地文件后得到的file对象 */
Position: 0, // 初次上传为0
}, function(err, data) {
console.log(err || data);
});
```

判断存储桶内的对象是否可追加对象:

```
cos.headObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'test.txt', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
if (err) return console.log(err);
// data.headers没有x-cos-object-type字段需要配置expose-headers，参考文档：https://intl.cloud.tencent.com/document/product/436/13318
var objectType = data.headers['x-cos-object-type'];
console.log(objectType === 'appendable');
});
```

查询可追加对象的Position并追加上传:

```
cos.headObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'test.txt', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
if (err) return console.log(err);
// 首先取到要追加的文件当前长度，即需要上送的Position
var position = data.headers['content-length'];
cos.appendObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
```

```
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'test.txt', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: '66666',
Position: position,
},
function(err, data) {
if (err) return console.log(err);
// 也可以取到下一次上传的position继续追加上传
// data.headers没有x-cos-next-append-position字段需要配置expose-headers，参考文档：ht
tps://intl.cloud.tencent.com/document/product/436/13318
var nextPosition = data.headers['x-cos-next-append-position'];
console.log(nextPosition);
})
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
Body	上传文件的内容，可以为字符串，File 对象或者 Blob 对象	String\File\E
Position	追加操作的起始点，单位为字节。首次追加则设置 Position=0，后续追加则设置 Position 为当前 Object 的 content-length	Number
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，例如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String
GrantRead	赋予被授权者读取对象的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantReadAcp	赋予被授权者读取对象的访问控制列表（ACL）的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantWriteAcp	赋予被授权者写入对象的访问控制列表（ACL）的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，默认值：STANDARD。更多存储类型请参见 存储类型概述	String
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据保存。大小限制2KB	String

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- RequestId	请求的唯一 ID	String

表单上传对象

JS SDK 未提供 POST Object 接口对应的方法，如果需要使用该接口，请参见 Web 端直传实践 里的“方案 B：使用 Form 表单上传”。

分块操作

> 说明：
> 一般不需要关注这些方法，sliceUploadFile 已经封装了以下分块操作，直接调用即可。更推荐使用高级上传 uploadFile 方法。

查询分块上传

功能说明

List Multipart Uploads 用来查询正在进行中的分块上传信息。单次最多列出1000个正在进行中的分块上传。

使用示例

获取前缀为 a 的未完成的 UploadId 列表，示例如下：

```
cos.multipartList({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Prefix: 'a', /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Prefix	对象键前缀匹配，限定返回中只包含指定前缀的对象键。注意使用 prefix 查询时，返回的对象键 中仍会包含 Prefix	String	否
Delimiter	定界符。为一个分隔符号，，用于对对象键进行分组。一般是传 `/` 。所有对象键从 Prefix 或从头（例如未指定 Prefix）到首个 delimiter 之间相同部分的路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix	String	否
EncodingType	规定返回值的编码格式，合法值：url	String	否
MaxUploads	设置最大返回的条目数量，合法取值为1 - 1000，默认1000	String	否
KeyMarker	与 upload-id-marker 一起使用 当 upload-id-marker 未被指定时： - ObjectName 字母顺序大于 key-marker 的条目将被列出 • 当 upload-id-marker 被指定时： - ObjectName 字母顺序大于 key-marker 的条目被列出 - ObjectName 字母顺序等于 key-marker 且 UploadID 大于 upload-id-marker 的条目将被列出	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
UploadIdMarker	与 key-marker 一起使用 • 当 key-marker 未被指定时： - upload-id-marker 将被忽略 • 当 key-marker 被指定时： - ObjectName 字母顺序大于 key-marker 的条目被列出 - ObjectName 字母顺序等于 key-marker 且 UploadID 大于 upload-id-marker 的条目将被列出	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Bucket	分块上传的目标存储桶	String
- Encoding-Type	规定返回值的编码格式，合法值：url	String
- KeyMarker	列出条目从该 key 值开始	String
- UploadIdMarker	列出条目从该 UploadId 值开始	String
- NextKeyMarker	假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点	String
- NextUploadIdMarker	假如返回条目被截断，则返回 UploadId 就是下一个条目的起点	String
- MaxUploads	设置最大返回的条目数量，合法取值为 1 - 1000	String
- IsTruncated	返回条目是否被截断，'true' 或者 'false'	String
- Prefix	对象键前缀匹配，限定返回中只包含指定前缀的对象键。	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- Delimiter	定界符。为一个分隔符号，，用于对对象键进行分组。一般是传 `/` 。所有对象键从 Prefix 或从头（如未指定 Prefix）到首个 delimiter 之间相同部分的路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix	String
- CommonPrefixs	将 prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	ObjectArray
- - Prefix	显示具体的 Common Prefixs	String
- Upload	分块上传的信息集合	ObjectArray
- - Key	对象的名称，即对象键	String
- - UploadId	表示本次分块上传的 ID	String
- - StorageClass	表示分块的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，更多存储类型请参见 存储类型概述	String
- - Initiator	表示本次上传发起者的信息	Object
- - - DisplayName	上传发起者的名称	String
- - - ID	上传发起者 ID，格式：`qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String
- - Owner	表示这些分块持有者的信息	Object
- - - DisplayName	分块持有者的名称	String
- - - ID	分块持有者 ID，格式：`qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String
- - Initiated	分块上传的起始时间	String

初始化分块上传

功能说明

Initiate Multipart Upload 请求实现初始化分块上传，成功执行此请求后会返回 Upload ID ，用于后续的 Upload Part 请求。

版权所有：腾讯云计算（北京）有限责任公司

使用示例

```
cos.multipartInit({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: fileObject, /* 必须，上传文件对象，可以是input[type="file"]标签选择本地文件后得到的file对象 */
}, function(err, data) {
console.log(err || data);
if (data) {
uploadId = data.UploadId;
}
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
CacheControl	RFC 2616 中定义的缓存策略，将作为对象的元数据保存	String
ContentDisposition	RFC 2616 中定义的文件名称，将作为对象的元数据保存	String
ContentEncoding	RFC 2616 中定义的编码格式，将作为对象的元数据保存	String
ContentType	RFC 2616 中定义的内容类型（MIME），将作为对象的元数据保存	String
Expires	RFC 2616 中定义的过期时间，将作为对象的元数据保存	String
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String

参数名	参数描述	类型
GrantRead	赋予被授权者读取对象的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，默认值：STANDARD。更多存储类型请参见 存储类型概述	String
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据返回。大小限制2KB	String
UploadAddMetaMd5	当上传时，给对象的元数据信息增加 x-cos-meta-md5 赋值为对象内容的 MD5 值，格式为 32 位小写字符串。例如：4d00d79b6733c9cc066584a02ed03410	String

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Bucket	分块上传的目标存储桶，由用户自定义字符串和系统生成 APPID 数字串由中划线连接而成，例如 examplebucket-1250000000	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
UploadId	在后续上传中使用的 ID	String

上传分块

功能说明

Upload Part 接口请求实现在初始化以后的分块上传，支持的块的数量为1 - 10000，块的大小为1MB - 5GB。

- 分块上传首先要进行初始化，用 Initiate Multipart Upload 接口初始化分块上传，得到一个 uploadId，该 ID 不但唯一标识这一分块数据，也标识了这分块数据在整个文件内的相对位置。
- 在每次请求 Upload Part 时候，需要携带 partNumber 和 uploadId，partNumber 为块的编号，支持乱序上传。
- 当传入 uploadId 和 partNumber 都相同的时候，后传入的块将覆盖之前传入的块。当 uploadId 不存在时会返回404错误，错误码为 NoSuchUpload。

使用示例

```
cos.multipartUpload({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
UploadId: 'exampleUploadId',
PartNumber: 1,
Body: fileObject, /* 必须，上传文件对象，可以是input[type="file"]标签选择本地文件后得到的file对象 */
}, function(err, data) {
console.log(err || data);
if (data) {
eTag = data.ETag;
}
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

参数名	参数描述	类型	是否必填
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	是
PartNumber	分块的编号	Number	是
UploadId	本次分块上传任务的编号	String	是
Body	上传文件分块的内容，可以为字符串，File 对象或者 Blob 对象	String\File\Blob\ArrayBuffer	是
Expect	RFC 2616 中定义的 HTTP 请求内容长度（字节）。当使用 `Expect: 100-continue` 时，在收到服务端确认后，才会发送请求内容	String	否
ContentMD5	RFC 1864中定义的经过 Base64 编码的128-bit 内容 MD5 校验值，此头部用来校验文件内容是否发生变化	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询已上传块

功能说明

List Parts 用来查询特定分块上传中的已上传的块，即列出指定 UploadId 所属的所有已上传成功的分块。

使用示例

```
cos.multipartListPart({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
UploadId: 'exampleUploadId', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
UploadId	标识本次分块上传的 ID，使用 Initiate Multipart Upload 接口初始化分块上传时得到的 UploadId	String	是
EncodingType	规定返回值的编码方式	String	否
MaxParts	单次返回最大的条目数量，默认1000	String	否
PartNumberMarker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Bucket	分块上传的目标存储桶	String
- Encoding-type	规定返回值的编码方式	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- UploadId	标识本次分块上传的 ID	String
- Initiator	用来表示本次上传发起者的信息	Object
- - DisplayName	上传发起者的名称	String
- - ID	上传发起者 ID，格式：`qcs::cam::uin/<owneruin>:uin/<subuin>`，如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String
- Owner	用来表示这些分块所有者的信息	Object
- - DisplayName	存储桶持有者的名称	String
- - ID	存储桶持有者 ID，一般为用户的 UIN	String
- StorageClass	用来表示这些分块的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，更多存储类型请参见 存储类型概述	String
- PartNumberMarker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
- NextPartNumberMarker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
- MaxParts	单次返回最大的条目数量	String
- IsTruncated	返回条目是否被截断，'true' 或者 'false'	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- Part	分块信息列表	ObjectArray
- - PartNumber	块的编号	String
- - LastModified	块最后修改时间	String
- - ETag	块的 MD5 算法校验值	String
- - Size	块大小，单位 Byte	String

完成分块上传

功能说明

Complete Multipart Upload 接口请求用来实现完成整个分块上传。当使用 Upload Parts 上传完所有块以后，必须调用该 API 来完成整个文件的分块上传。在使用该 API 时，您必须在请求 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。
由于分块上传完后需要合并，而合并需要数分钟时间，因而当合并分块开始的时候，COS 就立即返回200的状态码，在合并的过程中，COS 会周期性的返回空格信息来保持连接活跃，直到合并完成，COS 会在 Body 中返回合并后块的内容。

- 当上传块小于1MB的时候，在调用该 API 时，会返回400 EntityTooSmall。
- 当上传块编号不连续的时候，在调用该 API 时，会返回400 InvalidPart。
- 当请求 Body 中的块信息没有按序号从小到大排列的时候，在调用该 API 时，会返回400 InvalidPartOrder。
- 当 UploadId 不存在的时候，在调用该 API 时，会返回404 NoSuchUpload。

> 注意：
> 建议您及时完成分块上传或者舍弃分块上传，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

使用示例

```
cos.multipartComplete({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
UploadId: 'exampleUploadId', /* 必须 */
Parts: [
{PartNumber: 1, ETag: 'exampleETag'},
]
```

```
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
UploadId	上传任务编号	String	是
Parts	用来说明本次分块上传中块的信息列表	ObjectArray	是
- PartNumber	分块的编号	Number	是
- ETag	每个块文件的 MD5 算法校验值 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- headers	请求返回的头部信息	Object
- Location	上传完的文件访问地址	String
- Bucket	分块上传的目标存储桶	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- ETag	合并后文件的唯一 ID，格式："uuid-<分块数>" 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880-3"`，**注意前后携带双引号**	String

终止分块上传

功能说明

Abort Multipart Upload 用来实现终止一个分块上传操作并删除已上传的块。当您调用 Abort Multipart Upload 时，如果有正在使用这个 UploadId 上传块的请求，则 Upload Parts 会返回失败。当该 UploadId 不存在时，会返回404 NoSuchUpload。

> 注意：
> 建议您及时完成分块上传或者舍弃分块上传，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

使用示例

```
cos.multipartAbort({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
UploadId: 'exampleUploadId' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
UploadId	标识本次分块上传的 ID，使用 Initiate Multipart Upload 接口初始化分块上传时得到的 UploadId	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

高级接口（推荐）

强烈建议您使用高级接口，该类方法是对上面原生方法的封装，实现了分块上传的全过程，支持并发分块上传，支持断点续传，支持上传任务的取消，暂停和重新开始等。

高级上传

功能说明

Upload File实现高级上传，传入参数 SliceSize 可以控制文件大小超出一个数值（默认1MB）时自动使用分块上传（sliceUploadFile），否则使用简单上传(putObject)。

使用示例

```
cos.uploadFile({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: fileObject, /* 必须，上传文件对象，可以是input[type="file"]标签选择本地文件后得到的file对象 */
SliceSize: 1024 * 1024 * 5, /* 触发分块上传的阈值，超过5MB使用分块上传，非必须 */
onTaskReady: function(taskId) { /* 非必须 */
console.log(taskId);
},
onProgress: function (progressData) { /* 非必须 */
console.log(JSON.stringify(progressData));
},
onFileFinish: function (err, data, options) { /* 非必须 */
console.log(options.Key + '上传' + (err ? '失败' : '完成'));
},
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
Body	上传文件的内容，可以为 File 对象 或者 Blob 对象	File\Blob	是
SliceSize	表示文件大小超出一个数值时使用分块上传，单位 Byte，默认值1048576（1MB），小于等于该数值会使用 putObject 上传，大于该数值会使用 sliceUploadFile 上传	Number	否
AsyncLimit	分块的并发量，仅在触发分块上传时有效	Number	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
StorageClass	对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，更多存储类型请参见 存储类型概述	String	否
UploadAddMetaMd5	当上传时，给对象的元数据信息增加 x-cos-meta-md5 赋值为对象内容的 MD5 值，格式为 32 位小写字符串。例如：4d00d79b6733c9cc066584a02ed03410	String	否
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String	否
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String	否
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String	否
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	否
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String	否
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String	否
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否
onTaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function	否
- taskId	上传任务的编号	String	否
onProgress	上传文件的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如，上传50%即为0.5	Number	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
onFileFinish	每个文件完成或错误回调	Function	否
- err	上传的错误信息	Object	否
- data	文件完成的信息	Object	否
- options	当前完成文件的参数信息	Object	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Location	上传完的文件访问地址	String
- Bucket	分块上传的目标存储桶,仅在触发分块上传时返回	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述，仅在触发分块上传时返回	String
- ETag	合并后文件的唯一 ID，格式："uuid-<分块数>" 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880-3"`，**注意前后携带双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数。需要Expose-Headers里设置VersionId字段，参考文档	String

分块上传对象（断点续传）

功能说明

Slice Upload File 可用于实现文件的分块上传，适用于大文件上传。

说明：

- 浏览器未关闭的情况下，可以对上传任务进行暂停、重新开始或取消操作。
- 浏览器刷新后，如果将同一个文件上传到同一个存储桶路径下，则会根据 UploadId 校验内容，校验通过后，续传文件。

使用示例

```
cos.sliceUploadFile({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: fileObject, /* 必须，上传文件对象，可以是input[type="file"]标签选择本地文件后得到的file对象 */
onTaskReady: function(taskId) { /* 非必须 */
console.log(taskId);
},
onHashProgress: function (progressData) { /* 非必须 */
console.log(JSON.stringify(progressData));
},
onProgress: function (progressData) { /* 非必须 */
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
Body	上传文件的内容，可以为 File 对象 或者 Blob 对象	File\Blob	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
SliceSize	分块大小	Number	否
AsyncLimit	分块的并发量	Number	否
StorageClass	对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，更多存储类型请参见 存储类型概述	String	否
UploadAddMetaMd5	当上传时，给对象的元数据信息增加 x-cos-meta-md5 赋值为对象内容的 MD5 值，格式为 32 位小写字符串。例如：4d00d79b6733c9cc066584a02ed03410	String	否
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String	否
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String	否
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String	否
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	否
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String	否
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String	否
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否
onTaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function	否
- taskId	上传任务的编号	String	否
onHashProgress	计算文件 MD5 值的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经校验的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
- progressData.speed	文件的校验速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的校验百分比，以小数形式呈现，例如，校验50% 即为 0.5	Number	否
onProgress	上传文件的进度回调函数，回调参数为进度对象progressData	Function	否
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如，上传50%即为0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Location	上传完的文件访问地址	String
- Bucket	分块上传的目标存储桶	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- ETag	合并后文件的唯一 ID，格式："uuid-<分块数>" 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880-3"`，**注意前后携带双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数。需要Expose-Headers里设置VersionId字段，参考文档	String

批量上传

功能说明

方法一：
批量上传可以直接多次调用 putObject 和 sliceUploadFile，达到批量上传效果。通过实例化参数 FileParallelLimit 控制文件并发数，默认3个并发。

方法二：
可以调用 cos.uploadFiles 实现批量上传，方法参数 SliceSize 可以控制文件。以下是 uploadFiles 方法说明。

方法原型

调用 uploadFiles 操作：

```
cos.uploadFiles({
files: [{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: fileObject1, /* 必须，上传文件对象，可以是input[type="file"]标签选择本地文件后得到的file对象 */
onTaskReady: function(taskId) {
/* taskId可通过队列操作来取消上传cos.cancelTask(taskId)、停止上传cos.pauseTask(taskId)、重新开始上传cos.restartTask(taskId) */
console.log(taskId);
}
}, {
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '2.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: fileObject2, /* 必须，上传文件对象，可以是input[type="file"]标签选择本地文件后得到的file对象 */
onTaskReady: function(taskId) {
/* taskId可通过队列操作来取消上传cos.cancelTask(taskId)、停止上传cos.pauseTask(taskId)、重新开始上传cos.restartTask(taskId) */
console.log(taskId);
```

```
}
}],
SliceSize: 1024 * 1024 * 10, /* 设置大于10MB采用分块上传 */
onProgress: function (info) {
var percent = parseInt(info.percent * 10000) / 100;
var speed = parseInt(info.speed / 1024 / 1024 * 100) / 100;
console.log('进度：' + percent + '%; 速度：' + speed + 'Mb/s;');
},
onFileFinish: function (err, data, options) {
console.log(options.Key + '上传' + (err ? '失败' : '完成'));
},
}, function (err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
files	文件列表，每一项是传给 putObject 和 sliceUploadFile 的参数对象	Object	是
- Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
- Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
- Body	上传文件的内容，可以为 File 对象 或者 Blob 对象	File\Blob	是
- CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String	否
- ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String	否
- ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String	否
- ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	否
- ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String	否
- Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String	否
- Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
- onTaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function	否
-- taskId	上传任务的编号	String	否
SliceSize	表示文件大小多大以上使用分块上传，单位 Byte，默认值1048576（1MB），小于等于该数值会使用 putObject 上传，大于该数值会使用 sliceUploadFile 上传	Number	是
AsyncLimit	分块的并发量，仅在触发分块上传时有效	Number	否
onProgress	所有任务 进度汇总计算出来的上传进度	String	是
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如，上传50%即为0.5	Number	否
onFileFinish	每个文件完成或错误回调	Function	否
- err	上传的错误信息	Object	否
- data	文件完成的信息	Object	否
- options	当前完成文件的参数信息	Object	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- files	每个文件的 error 或 data	ObjectArray
- - error	上传的错误信息	Object
- - data	文件完成的信息	Object
- - options	当前完成文件的参数信息	Object

上传队列

JavaScript SDK 针对 putObject 和 sliceUploadFile 发起的上传任务都有记录队列里，队列相关方法如下。

1. var taskList = cos.getTaskList() 可以获取任务列表。
2. cos.pauseTask()、cos.restartTask()、cos.cancelTask() 操作任务。
3. cos.on('list-update', callback); 可以监听列表和进度变化。

完整的队列使用示例请参见 demo-queue。

取消上传任务

根据 taskId 取消上传任务。

使用示例

```
var taskId = 'xxxxx'; /* 必须 */
cos.cancelTask(taskId);
```

参数说明

参数名	参数描述	类型	是否必填
taskId	文件上传任务的编号，在调用 sliceUploadFile 方法时，其 TaskReady 回调会返回该上传任务的 taskId	String	是

暂停上传任务

根据 taskId 暂停上传任务。

使用示例

```
var taskId = 'xxxxx'; /* 必须 */
cos.pauseTask(taskId);
```

版权所有：腾讯云计算（北京）有限责任公司

参数说明

参数名	参数描述	类型	是否必填
taskId	文件上传任务的编号，在调用 sliceUploadFile 方法时，其 TaskReady 回调会返回该上传任务的 taskId	String	是

重启上传任务

根据 taskId 重新开始上传任务，可以用于开启用户手动停止的（调用 pauseTask 停止）或者因为上传错误而停止的上传任务。

使用示例

```
var taskId = 'xxxxx'; /* 必须 */
cos.restartTask(taskId);
```

参数说明

参数名	参数描述	类型	是否必填
taskId	文件上传任务的编号，在调用 sliceUploadFile 方法时，其 TaskReady 回调会返回该上传任务的 taskId	String	是

版权所有：腾讯云计算（北京）有限责任公司

下载对象

最近更新时间：2023-02-07 15:03:05

简介

> 注意：
> 该接口用于读取对象内容，如果需要发起浏览器下载文件，可以通过 cos.getObjectUrl 获取 url 再触发浏览器下载，具体参见 预签名 URL 文档。

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地

功能说明

GET Object 接口请求可以获取存储桶里指定文件的内容，得到文件内容是字符串格式。
下载文件到本地请参见 预签名 URL 。

下载单个对象

使用示例

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
onProgress: function (progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data.Body);
});
```

指定 Range 获取文件内容：

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
```

版权所有：腾讯云计算（北京）有限责任公司

```
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Range: 'bytes=1-3', /* 非必须 */
onProgress: function (progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data.Body);
});
```

返回blob格式文件内容：

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'exampleobject', /* 必须 */
DataType: 'blob', /* 非必须 */
onProgress: function (progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data.Body);
});
```

下载对象（单链接限速）：

> 说明：
> 关于下载对象的限速说明，请参见 单链接限速。

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Headers: {
'x-cos-traffic-limit': 819200, // 限速值设置范围为819200 - 838860800，即100KB/s - 10
0MB/s，如果超出该范围将返回400错误。
},
onProgress: function (progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data.Body);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
DataType	返回文件内容格式，枚举值：string、blob、arraybuffer，默认值：string	String	否
ResponseContentType	设置响应头部中的 Content-Type 参数	String	否
ResponseContentLanguage	设置返回头部中的 Content-Language 参数	String	否
ResponseExpires	设置返回头部中的 Content-Expires 参数	String	否
ResponseCacheControl	设置返回头部中的 Cache-Control 参数	String	否
ResponseContentDisposition	设置返回头部中的 Content-Disposition 参数	String	否
ResponseContentEncoding	设置返回头部中的 Content-Encoding 参数	String	否
Range	RFC 2616 中定义的字节范围，范围值必须使用 bytes=first-last 格式，first 和 last 都是基于0开始的偏移量。例如 bytes=0-9 表示下载对象的开头10个字节的数据，如果不指定，则表示下载整个对象	String	否
IfModifiedSince	当对象在指定时间后被修改，则返回对象，否则返回304（Not Modified）	String	否
IfUnmodifiedSince	当对象在指定时间后未被修改，则返回对象，否则返回412（precondition failed）	String	否
IfMatch	当对象的 ETag 与指定的值一致，则返回对象，否则返回412（precondition failed）	String	否
IfNoneMatch	当对象的 ETag 与指定的值不一致，则返回对象，否则返回304（not modified）	String	否
VersionId	指定要下载的对象的版本 ID	String	否
onProgress	进度的回调函数，进度回调响应对象（progressData）属性如下	Function	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
- progressData.loaded	已经下载的对象部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个对象的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	对象的下载速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	对象下载的百分比，以小数形式呈现，例如，下载50%即为0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，304，403，404等	Number
- headers	请求返回的头部信息	Object
- CacheControl	RFC 2616 中定义的缓存指令，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- ContentDisposition	RFC 2616 中定义的文件名称，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- ContentEncoding	RFC 2616 中定义的编码格式，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- Expires	RFC 2616 中定义的缓存失效时间，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- x-cos-storage-class	对象的存储类型，枚举值：STANDARD、STANDARD_IA，更多存储类型请参见 存储类型概述。 **注意：如果没有返回该头部，则说明文件存储类型为 STANDARD（标准存储）**	String
- x-cos-meta-*	用户自定义的元数据	String
- NotModified	如果请求时带有 IfModifiedSince 则返回该属性，如果文件未被修改，则为 true，否则为 false	Boolean
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String
- Body	返回的文件内容，默认为 String 形式	String

批量下载对象

> 注意：
> 不推荐使用，触发浏览器下载后，无法用代码控制开始、停止。当下载的对象过多时，可能造成较差的体验。

使用示例

按前缀下载多个对象（下载指定目录下的文件）：

```
var getObjects = function (marker) {
cos.getBucket({
Bucket: config.Bucket,
Region: config.Region,
Prefix: 'abc', /* 要下载的目录，或要下载的前缀 */
Marker: marker,
MaxKeys: 1000,
}, function (listError, listResult) {
if (listError) return console.log('list error:', listError);
var nextMarker = listResult.NextMarker;
```

```
listResult.Contents.forEach(function (item) {
cos.getObjectUrl({
Bucket: config.Bucket,
Region: config.Region,
Key: item.Key,
}, function(err, data) {
if (err) return console.log(err);
setTimeout(() => {
var downloadUrl = data.Url + (data.Url.indexOf('?') > -1 ? '&' : '?') + 'response
-content-disposition=attachment'; // 补充强制下载的参数
window.open(downloadUrl); // 这里是新窗口打开 url，如果需要在当前窗口打开，可以使用隐藏的
iframe 下载，或使用 a 标签 download 属性协助下载
}, 500);
});
});
// 如果没有处理完需要继续查询下一页文件列表
if (listResult.IsTruncated === 'true') getMultipleObjects(nextMarker);
});
}
getObjects();
```

列出对象

最近更新时间：2022-02-18 16:08:50

简介

本文档提供关于列出对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象

功能说明

查询存储桶下的部分或者全部对象。

> 注意：
> 在 COS 中，列出的对象键为 project/，表示是一个内容为空的对象，展示方式上模拟了传统文件夹。相关介绍可参见 文件夹和目录。

使用示例

示例一：列出目录 a 的所有文件。

```
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a/', /* Prefix表示列出的object的key以prefix开始，非必须 */
}, function(err, data) {
console.log(err || data.Contents);
});
```

返回值格式：

```
{
"Name": "examplebucket-1250000000",
"Prefix": "",
"Marker": "a/",
"MaxKeys": "1000",
"Delimiter": "",
"IsTruncated": "false",
```

```
"Contents": [{
"Key": "a/3mb.zip",
"LastModified": "2018-10-18T07:08:03.000Z",
"ETag": "\"05a9a30179f3db7b63136f30aa6aacae-3\"",
"Size": "3145728",
"Owner": {
"ID": "1250000000",
"DisplayName": "1250000000"
},
"StorageClass": "STANDARD"
}],
"statusCode": 200,
"headers": {}
}
```


示例二：列出目录 a 的文件，不深度遍历。

```
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a/', /* Prefix表示列出的object的key以prefix开始，非必须 */
Delimiter: '/', /* Deliter表示分隔符，设置为/表示列出当前目录下的object，设置为空表示列出所有的object，非必须 */
}, function(err, data) {
console.log(err || data.CommonPrefixes);
});
```

返回值格式：


```
{
"Name": "examplebucket-1250000000",
"Prefix": "a/",
"Marker": "",
"MaxKeys": "1000",
"Delimiter": "/",
"IsTruncated": "false",
"CommonPrefixes": [{
"Prefix": "a/1/"
}],
"Contents": [{
"Key": "a/3mb.zip",
"LastModified": "2018-10-18T07:08:03.000Z",
"ETag": "\"05a9a30179f3db7b63136f30aa6aacae-3\"",
"Size": "3145728",
"Owner": {
"ID": "1250000000",
```


版权所有：腾讯云计算（北京）有限责任公司

```
"DisplayName": "1250000000"
},
"StorageClass": "STANDARD"
}],
"statusCode": 200,
"headers": {}
}
```

示例三：列出目录下所有文件。

```
var listFolder = function(marker) {
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a/', /* Prefix表示列出的object的key以prefix开始，非必须 */
Marker: marker,
MaxKeys: 1000,
}, function(err, data) {
if (err) {
return console.log('list error:', err);
} else {
console.log('list result:', data.Contents);
if (data.IsTruncated === 'true') listFolder(data.NextMarker);
else return console.log('list complete');
}
});
};
listFolder();
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Prefix	对象键前缀匹配，限定返回中只包含指定前缀的对象键	String	否
Delimiter	定界符。为一个分隔符号，用于对对象键进行分组。一般是传 `/` 。所有对象键从 Prefix 或从头（如未指定 Prefix）到首个 delimiter 之间相同部分的路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Marker	起始对象键标记，列出从 Marker 开始 MaxKeys 条目，默认顺序为 UTF-8 字典序	String	否
MaxKeys	单次返回最大的条目数量，默认1000，最大为1000	String	否
EncodingType	规定返回值的编码方式，可选值：url，代表返回的对象键为 URL 编码（百分号编码）后的值，例如“腾讯云”将被编码为 `%E8%85%BE%E8%AE%AF%E4%BA%91`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- headers	请求返回的头部信息	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- Name	存储桶的名称，格式为 BucketName-APPID，例如 examplebucket-1250000000	String
- Prefix	对象键前缀匹配，从该标记之后（不含）按照 UTF-8 字典序返回对象键条目	String
- Marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 Marker 开始	String
- MaxKeys	单次响应请求内返回结果的最大的条目数量	String
- Delimiter	定界符	String
- IsTruncated	响应请求条目是否被截断，值为 'true' 或者 'false'	String
- NextMarker	假如返回条目被截断，则返回 NextMarker 表示下一个条目的起点	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- CommonPrefixes	将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	ObjectArray
- - Prefix	单条 Common Prefix 的前缀	String
- EncodingType	返回值的编码方式，作用于 Delimiter，Marker，Prefix，NextMarker，Key	String
- Contents	对象元数据信息列表	ObjectArray
- - Key	对象键，即对象的名称	String
- - ETag	根据对象内容计算出的 MD5 算法校验值，例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- - Size	对象大小，单位 Byte	String
- - LastModified	对象最后修改时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	String
- - Owner	对象持有者信息	Object
- - - ID	对象持有者的完整 ID，格式为 `qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]`，例如 `qcs::cam::uin/100000000001:uin/100000000001`，其中100000000001为 uin	String
- - - DisplayName	对象持有者的名称	String
- - StorageClass	对象的存储类型，枚举值为：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE，详情参见 存储类型 文档	String

删除对象

最近更新时间：2022-01-13 12:40:58

简介

本文档提供关于对象的删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

删除单个对象

功能说明

DELETE Object 接口请求可以在 COS 的存储桶中将一个对象（Object）删除。该操作需要请求者对存储桶有 WRITE 权限。

使用示例

```
cos.deleteObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
VersionId	要删除的对象版本 ID 或 DeleteMarker 版本 ID	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，204，403，404等，**如果删除成功或者文件不存在则返回204或200，如果找不到指定的 Bucket，则返回404**	Number
- headers	请求返回的头部信息	Object

删除多个对象

功能说明

DELETE Multiple Objects 接口请求实现在指定存储桶中批量删除对象，单次请求最大支持批量删除1000个对象。对于响应结果，COS 提供 Verbose 和 Quiet 两种模式： Verbose 模式将返回每个对象的删除结果，Quiet 模式只返回报错的对象信息。

使用示例

删除多个文件：

```
cos.deleteMultipleObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Objects: [
{ Key: '1.jpg' }, /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
{ Key: '2.txt' }, /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
]
}, function(err, data) {
```

```
console.log(err || data);
});
```

按前缀删除多个对象（删除指定目录a下的文件）：

```
var deleteFiles = function (marker) {
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a/',
Marker: marker,
MaxKeys: 1000,
}, function (listError, listResult) {
if (listError) return console.log('list error:', listError);
var nextMarker = listResult.NextMarker;
var objects = listResult.Contents.map(function (item) {
return {Key: item.Key}
});
cos.deleteMultipleObject({
Bucket: bucket,
Region: region,
Objects: objects,
}, function (delError, deleteResult) {
if (delError) {
console.log('delete error', delError);
console.log('delete stop');
} else {
console.log('delete result', deleteResult);
if (listResult.IsTruncated === 'true') deleteFiles(nextMarker);
else console.log('delete complete');
}
});
});
}
deleteFiles();
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

参数名	参数描述	类型	是否必填
Quiet	布尔值，这个值决定了是否启动 Quiet 模式。值为 true 启动 Quiet 模式，值为 false 则启动 Verbose 模式，默认值为 false	Boolean	否
Objects	要删除的对象列表	ObjectArray	是
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
- VersionId	要删除的对象版本 ID 或 DeleteMarker 版本 ID	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200，204，403，404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，204，403，404等	Number
- headers	请求返回的头部信息	Object
- Deleted	说明本次删除成功的对象信息列表	ObjectArray
- - Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- - VersionId	如果参数传入了 VersionId，返回也会带上 VersionId，表示刚操作的对象版本或 DeleteMarker 版本	String
- - DeleteMarker	如果开启了版本控制，并且参数没有 VersionId，本次删除不会真正抹去文件内容，只新增一个 DeleteMarker 代表可见的文件已删除，枚举值：true、false	String
- - DeleteMarkerVersionId	当返回的 DeleteMarker 为 true 时，返回刚新增的 DeleteMarker 的 VersionId	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- Error	说明本次删除失败的对象信息列表	ObjectArray
- - Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- - Code	删除失败的错误码	String
- - Message	删除错误信息	String

复制与移动对象

最近更新时间：2023-01-30 11:45:28

简介

本文档提供关于对象的复制、移动操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object - Copy	设置对象复制（修改对象属性）	复制文件到目标路径

分块操作

API	操作名	操作描述
Initiate Multipart Upload	初始化分块复制	初始化分块复制任务
Upload Part - Copy	复制分块	将其他对象复制为一个分块
Complete Multipart Upload	完成分块复制	完成整个对象的分块复制

简单操作

复制对象

功能说明

PUT Object - Copy 请求创建一个已存在 COS 的对象的副本，即将一个对象从源路径（对象键）复制到目标路径（对象键）。在复制的过程中，对象元数据和访问控制列表（ACL）可以被修改。
用户可以通过该接口创建副本、修改对象元属性（源对象和目标文件的属性相同）、移动或重命名对象（先复制，再单独调用删除接口）。

> 注意：
> 建议对象大小1MB - 5GB，超过5GB的对象请使用高级接口复制对象 Slice Copy File。

使用示例

```
/* 把a/1.jpg复制一份到b/1.jpg */
cos.putObjectCopy({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'b/1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/a/1.jpg', /*
必须 */
/* CopySource中的Key含中文时，需要自行转义 */
// CopySource: `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/${encodeUR
IComponent('a/中文文件名.jpg')}`,
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
Region	存储桶所在地域，枚举值请参见 地域和访问域名
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述
CopySource	源对象 URL 路径，可以通过 URL 参数 ?versionId=<versionId> 参数指定指定历史版本
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，例如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**
GrantRead	赋予被授权者读取对象的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`

参数名	参数描述
GrantWrite	赋予被授权者写入对象的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`
MetadataDirective	是否拷贝元数据，枚举值：Copy，Replaced，默认值 Copy。假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制；假如标记为 Replaced，按 Header 信息修改元数据。**当目标路径和原路径一致，即用户试图修改元数据时，必须为 Replaced**
CopySourceIfModifiedSince	当对象在指定时间后被修改，则执行操作，否则返回412。**可与 CopySourceIfNoneMatch 一起使用，与其他条件联合使用返回冲突**
CopySourceIfUnmodifiedSince	当对象在指定时间后未被修改，则执行操作，否则返回412。**可与 CopySourceIfMatch 一起使用，与其他条件联合使用返回冲突**
CopySourceIfMatch	当对象的 Etag 和给定一致时，则执行操作，否则返回412。**可与 CopySourceIfUnmodifiedSince 一起使用，与其他条件联合使用返回冲突**
CopySourceIfNoneMatch	当对象的 Etag 和给定不一致时，则执行操作，否则返回412。**可与 CopySourceIfModifiedSince 一起使用，与其他条件联合使用返回冲突**
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA 等，默认值：STANDARD。更多存储类型请参见 存储类型概述
x-cos-meta-*	其他自定义的文件头部

回调函数说明

版权所有：腾讯云计算（北京）有限责任公司

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- ETag	文件的 MD5 算法校验值，例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- LastModified	返回对象最后被修改时间，例如2017-06-23T12:33:27.000Z	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

移动对象

功能说明

移动对象主要包括两个操作：复制源对象到目标位置，删除源对象。

由于 COS 通过存储桶名称（Bucket）和对象键（ObjectKey）来标识对象。移动对象也就意味着修改这个对象的标识，COS目前没有提供修改对象唯一标识名的单独接口，但是可以通过组合复制对象加上删除对象的基本操作，来达到修改对象标识的目的，从而实现移动对象。

使用示例

```
/* 把a/1.jpg移动到b/1.jpg */
/* 也可替换为高级复制cos.sliceCopyFile实现 */
cos.putObjectCopy({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'b/1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/a/1.jpg', /*
```

版权所有：腾讯云计算（北京）有限责任公司

```
必须 */
}, function(err, data) {
if (err) return console.log(err);
/* 删除a/1.jpg
cos.deleteObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'a/1.jpg' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
});
```

修改对象存储类型

功能说明

修改对象存储类型通过复制对象（源和目的是同一个对象）时设置StorageClass实现。

使用示例

```
/* 把根目录下的1.jpg设置为归档存储类型 */
/* 也可替换为高级复制cos.sliceCopyFile实现 */
cos.putObjectCopy({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/1.jpg', /* 必
须 */
StorageClass: 'ARCHIVE', /* 设置为归档存储 */
}, function(err, data) {
console.log(err || data);
});
```

分块操作

初始化分块复制

功能说明

Initiate Multipart Upload 请求实现初始化分块复制，成功执行此请求后会返回 Upload ID ，用于后续的 Upload Part - Copy 请求。

使用示例

```
cos.multipartInit({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
if (data) {
uploadId = data.UploadId;
}
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
CacheControl	RFC 2616 中定义的缓存策略，将作为对象的元数据保存	String
ContentDisposition	RFC 2616 中定义的文件名称，将作为对象的元数据保存	String
ContentEncoding	RFC 2616 中定义的编码格式，将作为对象的元数据保存	String
ContentType	RFC 2616 中定义的内容类型（MIME），将作为对象的元数据保存	String
Expires	RFC 2616 中定义的过期时间，将作为对象的元数据保存	String
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
GrantRead	赋予被授权者读取对象的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，默认值：STANDARD。更多存储类型请参见 存储类型概述	String
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据返回。大小限制2KB	String
UploadAddMetaMd5	当上传时，给对象的元数据信息增加 x-cos-meta-md5 赋值为对象内容的 MD5 值，格式为 32 位小写字符串。例如：4d00d79b6733c9cc066584a02ed03410	String

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Bucket	分块上传的目标存储桶，由用户自定义字符串和系统生成 APPID 数字串由中划线连接而成，例如 examplebucket-1250000000	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
UploadId	在后续上传中使用的 ID	String

复制分块

功能说明

Upload Part - Copy 接口请求实现将一个对象的分块内容从源路径复制到目标路径。

> 注意：
> 使用上传分块对象，必须先初始化分块上传。在初始化分块上传的响应中，会返回一个唯一的描述符（upload ID），您需要在分块上传请求中携带此 ID。

使用示例

```
cos.uploadPartCopy({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject',
/* 必须 */
UploadId: 'exampleUploadId', /* 必须 */
PartNumber: '1', /* 必须 */
}, function(err, data) {
console.log(err || data);
if (data) {
eTag = data.ETag;
}
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

参数名	参数描述	类型	是否必填
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
CopySource	源对象 URL 路径，可以通过 URL 参数 ?versionId=<versionId> 参数指定指定历史版本	String	是
partNumber	分块拷贝的块号	String	是
uploadId	使用上传分块文件，必须先初始化分块上传。在初始化分块上传的响应中，会返回一个唯一的描述符（upload ID），您需要在分块上传请求中携带此 ID	String	是
CopySourceRange	源对象的字节范围，范围值必须使用 bytes=first-last 格式，first 和 last 都是基于0开始的偏移量。例如 bytes=0-9 表示您希望拷贝源对象的开头10个字节的数据，如果不指定，则表示拷贝整个对象	String	否
CopySourceIfMatch	当对象的 Etag 和给定一致时，则执行操作，否则返回412，可与 x-cos-copy-source-If-Unmodified-Since 一起使用，与其他条件联合使用返回冲突	String	否
CopySourceIfNoneMatch	当对象的 Etag 和给定不一致时，则执行操作，否则返回412，可与 x-cos-copy-source-If-Modified-Since 一起使用，与其他条件联合使用返回冲突	String	否
CopySourceIfUnmodifiedSince	当对象在指定时间后未被修改，则执行操作，否则返回412，可与 x-cos-copy-source-If-Match 一起使用，与其他条件联合使用返回冲突	String	否
CopySourceIfModifiedSince	当对象在指定时间后被修改，则执行操作，否则返回412，可与 x-cos-copy-source-If-None-Match 一起使用，与其他条件联合使用返回冲突	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- ETag	文件的 MD5 算法校验值，例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- LastModified	返回对象最后修改时间，GMT 格式	String

完成分块复制

功能说明

Complete Multipart Upload 接口请求用来实现完成整个分块复制。当使用 Copy Parts 复制完所有块以后，必须调用该 API 来完成整个文件的分块复制。在使用该 API 时，您必须在请求 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。
由于分块复制完后需要合并，而合并需要数分钟时间，因而当合并分块开始的时候，COS 就立即返回200的状态码，在合并的过程中，COS 会周期性的返回空格信息来保持连接活跃，直到合并完成，COS 会在 Body 中返回合并后块的内容。

- 当上传块小于1MB的时候，在调用该 API 时，会返回400 EntityTooSmall。
- 当上传块编号不连续的时候，在调用该 API 时，会返回400 InvalidPart。
- 当请求 Body 中的块信息没有按序号从小到大排列的时候，在调用该 API 时，会返回400 InvalidPartOrder。
- 当 UploadId 不存在的时候，在调用该 API 时，会返回404 NoSuchUpload。

> 注意：
> 建议您及时完成分块复制或者舍弃分块复制，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

使用示例

```
cos.multipartComplete({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
```

```
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg, a/b/test.txt），必须字段 */
UploadId: 'exampleUploadId', /* 初始化分块任务时拿到，必须 */
Parts: [
{PartNumber: '1', ETag: 'exampleETag'},
]
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
UploadId	上传任务编号	String	是
Parts	用来说明本次分块上传中块的信息列表	ObjectArray	是
- PartNumber	分块的编号	String	是
- ETag	每个块文件的 MD5 算法校验值 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Location	上传完的文件访问地址	String
- Bucket	分块上传的目标存储桶	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- ETag	合并后文件的唯一 ID，格式："uuid-<分块数>" 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880-3"`，**注意前后携带双引号**	String

高级接口（推荐）

强烈建议您使用高级接口，该类方法是对上面原生方法的封装，实现了分块复制的全过程。

分块复制对象

功能说明

Slice Copy File 可用于实现通过分块复制将一个文件从源路径复制到目标路径。在拷贝的过程中，对象元属性和 ACL 可以被修改。用户可以通过该接口实现文件移动，文件重命名，修改文件属性和创建副本。

方法原型

调用 Slice Copy File 操作：

```
/* 把a/1.jpg复制到b/1.jpg */
cos.sliceCopyFile({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'b/1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/a/1.jpg', /*
必须 */
onProgress:function (progressData) { /* 非必须 */
console.log(JSON.stringify(progressData));
}
},function (err,data) {
```

```
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
CopySource	源对象 URL 路径，可以通过 URL 参数 ?versionId=<versionId> 参数指定指定历史版本	String	是
ChunkSize	分块复制时，每片的大小字节数，默认值 1048576（1MB）	Number	否
SliceSize	表示文件大小超出一个数值时使用分块复制，单位 Byte，默认值5G。小于等于该数值会使用 putObjectCopy 上传，大于该数值会使用 sliceCopyFile 上传	Number	否
onProgress	上传文件的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如：上传50%即为0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Location	创建对象的外网访问域名	String
- Bucket	分块上传的目标存储桶	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- ETag	合并后文件的 MD5 算法校验值 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

恢复归档对象

最近更新时间：2022-10-19 14:17:22

简介

本文档提供关于恢复归档对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
POST Object restore	恢复归档对象	将归档类型的对象取回访问

恢复归档对象

功能说明

POST Object restore 接口可以对一个通过 COS 归档为 archive 类型的对象进行恢复，恢复出的可读取对象是临时的，您可以设置需要保持可读，以及随后删除该临时副本的时间。您可以用 Days 参数来指定临时对象的过期时间，若超出该时间且期间您没有发起任何复制、延长等操作，该临时对象将被系统自动删除。临时对象仅为 archive 类型对象的副本，被归档的源对象在此期间将始终存在。

使用示例

```
cos.restoreObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
RestoreRequest: {
Days: 1,
CASJobParameters: {
Tier: 'Expedited'
}
},
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
RestoreRequest	用于恢复数据的容器	Object	是
- Days	设置临时副本的过期时间	Number	是
- CASJobParameters	归档存储工作参数的容器	Object	是
- - Tier	恢复数据时，Tier 可以指定为 COS 提供的三种恢复模式，分别为 Standard（标准取回模式，恢复任务在3 - 5小时内完成）、Expedited（快速取回模式，恢复任务在15分钟内可完成）以及 Bulk（批量取回模式，恢复任务在5 - 12小时内完成）	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

查询对象元数据

最近更新时间：2022-01-13 12:40:58

简介

本文档提供关于查询对象元数据操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

查询对象元数据

功能说明

查询对象的元数据信息。

使用示例

```
cos.headObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
IfModifiedSince	当对象在指定时间后被修改，返回对应对象的元数据信息，否则返回304	String	否

版权所有：腾讯云计算（北京）有限责任公司

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，304等，如果在指定时间后未被修改，则返回304	Number
- headers	请求返回的头部信息	Object
- x-cos-object-type	用来表示对象是否可以被追加上传，枚举值：normal、appendable，默认 normal 不显示在返回中	String
- x-cos-storage-class	对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，默认值：STANDARD 不显示在返回中。更多存储类型请参见 存储类型概述	String
- x-cos-meta-*	用户自定义的 meta	String
- NotModified	对象是否在指定时间后未被修改	Boolean
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

检查对象是否存在

最近更新时间：2022-07-07 14:08:31

简介

本文档提供快捷查询存储桶中某个对象是否存在的示例代码。示例代码实际调用了 HEAD Object COS API，是该接口的简化版。

HEAD Object 除了检查对象是否存在，主要功能为返回对象元数据。如需查看包含了 HEAD Object 完整功能的 SDK 接口，请参见 查询对象元数据。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

功能说明

检查存储桶中是否存在某个对象,可通过封装cos.headObject方法实现。

使用示例

```
function doesObjectExist() {
cos.headObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
if (data) {
console.log('对象存在');
} else if (err.statusCode == 404) {
console.log('对象不存在');
} else if (err.statusCode == 403) {
console.log('没有该对象读权限');
}
});
}
```

参数说明

参数名	参数描述	类型	是否必填

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
IfModifiedSince	当对象在指定时间后被修改，返回对应对象的元数据信息，否则返回304	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，304等，如果在指定时间后未被修改，则返回304	Number
- headers	请求返回的头部信息	Object
- x-cos-object-type	用来表示对象是否可以被追加上传，枚举值：normal、appendable，默认 normal 不显示在返回中	String
- x-cos-storage-class	对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，默认值：STANDARD 不显示在返回中。更多存储类型请参见 存储类型概述	String
- x-cos-meta-*	用户自定义的 meta	String
- NotModified	对象是否在指定时间后未被修改	Boolean

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

对象访问 URL

最近更新时间：2023-05-23 16:14:05

简介

本文档提供获取已上传到存储的对象访问 URL 的代码示例。

获取对象访问 URL

功能说明

查询对象访问的 URL，该接口不会判断对象是否真实存在。

说明：

- 如何使生成的对象 URL 在浏览器中打开是预览，而不是下载：在获取的 URL 后拼接参数 response-content-disposition=inline。
- 如何使生成的对象 URL 在浏览器中打开是下载，而不是预览：在获取的 URL 后拼接参数 response-content-disposition=attachment。
- 下载时如何重命名文件：在获取的 URL 后拼接参数 filename，例如 response-content-disposition=attachment;filename=自定义文件名。
- 若使用临时密钥生成预签名 URL，请确保申请临时密钥的 action 添加了 `"name/cos:GetObject"` 权限。

使用示例

获取在浏览器默认下载的 URL：

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', // 填入您自己的存储桶，必须字段
Region: 'COS_REGION', // 存储桶所在地域，例如 ap-beijing，必须字段
Key: '头像.jpg', // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段
Sign: true, // 获取带签名的对象 URL

},
function (err, data) {
if (err) return console.log(err);
```

```
/* url为对象访问 url */
var url = data.Url;
/* 复制 downloadUrl 的值到浏览器打开会自动触发下载 */
var downloadUrl =
url +
(url.indexOf('?') > -1 ? '&' : '?') +
'response-content-disposition=attachment;filename=图片.jpg'; // 补充强制下载的参数并重命名下载后的文件
}
);
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
Sign	是否返回带有签名的 Url，默认为 true，当对象为私有读时，获取到不带签名的 url 依然没有权限访问	Boolean	否
Protocol	可选填为 `http:` 或 `https:`，默认为 `http:`（带冒号）	String	否
Domain	存储桶访问域名，默认为 {BucketName-APPID}.cos.{Region}.myqcloud.com	String	否
Method	操作方法，例如 GET，POST，DELETE，HEAD 等 HTTP 方法，默认为 GET	String	否
Query	签名中要签入的请求参数，{key: 'val'} 的格式	Object	否
Headers	签名中要签入的请求头部，{key: 'val'} 的格式	Object	否
Expires	签名几秒后失效，默认为 900 秒	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- Url	计算得到的 Url	String

生成预签名链接

最近更新时间：2023-02-23 16:36:33

简介

JavaScript SDK 提供获取对象 URL、获取请求预签名 URL 接口。

说明：

- 建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。申请临时密钥的 action 需添加 `"name/cos:GetObject"` 权限。
- 如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。

计算签名

COS XML API 的请求里，私有资源操作都需要鉴权凭证 Authorization，用于判断当前请求是否合法。

鉴权凭证使用方式有两种：

1. 放在 header 参数里使用，字段名：authorization。
2. 放在 url 参数里使用，字段名：sign。

COS.getAuthorization 方法用于计算鉴权凭证（Authorization），用以验证请求合法性的签名信息。

注意：
该方法推荐只在前端调试时使用，项目上线不推荐使用前端计算签名的方法，有暴露密钥的风险。

使用示例

获取对象下载的鉴权凭证：

```
// SECRETID 和 SECRETKEY 请登录 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
var Authorization = COS.getAuthorization({
```

```
SecretId: 'SECRETID', // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
SecretKey: 'SECRETKEY', // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
Method: 'get',
Key: 'exampleobject',
Expires: 60,
Query: {},
Headers: {},
});
```

参数说明

参数名	参数描述	类型	是否必填
SecretId	用户的 SecretId	String	是
SecretKey	用户的 SecretKey	String	是
Method	操作方法，例如 GET，POST，DELETE，HEAD 等 HTTP 方法	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，**如果请求操作是对文件的，则为文件名，且为必须参数**。如果操作是对于存储桶，则为空	String	否
Query	签名中要签入的请求参数，{key: 'val'} 的格式	Object	否
Headers	签名中要签入的请求头部，{key: 'val'} 的格式	Object	否
Expires	签名几秒后失效，默认为 900 秒	Number	否

返回值说明

返回值是计算得到的鉴权凭证字符串 authorization。

获取请求预签名 URL

下载请求示例

示例一：获取不带签名的对象的 Url

```
var url = cos.getObjectUrl({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
```

```
Sign: false,
});
```

示例二：获取带签名的对象的 Url

```
var url = cos.getObjectUrl({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
});
```

示例三：通过 callback 获取带签名 Url

> 说明：
> 如果签名过程是异步获取，需要通过 callback 获取带签名 Url。

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: false,
},
function (err, data) {
console.log(err || data.Url);
}
);
```

示例四：指定链接有效时间

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: true,
Expires: 3600, // 单位秒
},
function (err, data) {
console.log(err || data.Url);
}
);
```

版权所有：腾讯云计算（北京）有限责任公司

示例五：获取对象的 Url 并下载对象

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: true,
},
function (err, data) {
if (err) return console.log(err);
var downloadUrl =
data.Url +
(data.Url.indexOf('?') > -1 ? '&' : '?') +
'response-content-disposition=attachment'; // 补充强制下载的参数
/* 可拼接 filename 来实现下载时重命名 */
/* downloadUrl += ';filename=myname'; */
// （推荐使用 window.open()方式）这里是新窗口打开 url，如果需要在当前窗口打开，可以使用隐藏的 iframe 下载，或使用 a 标签 download 属性协助下载
window.open(downloadUrl);
}
);
```

示例六：生成预签名 URL，并在签名中携带 Query 和 Header

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: true,
/* 传入的请求参数需与实际请求相同，能够防止用户篡改此 HTTP 请求的参数 */
Query: {
'imageMogr2/thumbnail/200x/': '',
},
/* 传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的 HTTP 请求头部 */
Headers: {
host: 'xxx', /* 指定 host 访问，非指定的 host 访问会报错403 */
},
},
function (err, data) {
console.log(err || data.Url);
}
);
```

上传请求示例

示例一：获取预签名 Put Object 上传 Url

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Method: 'PUT',
Sign: true,
},
function (err, data) {
if (err) return console.log(err);
console.log(data.Url);

/* 获取到 Url 后，前端可以这样 ajax 上传 */
var xhr = new XMLHttpRequest();
xhr.open('PUT', data.Url, true); /* PUT 和 getObjectUrl 填写的 Method 对应 */
xhr.onload = function (e) {
console.log('上传成功', xhr.status, xhr.statusText);
};
xhr.onerror = function (e) {
console.log('上传出错', xhr.status, xhr.statusText);
};
xhr.send(file); /* file 是要上传的文件对象 */
}
);
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，**如果请求操作是对文件的，则为文件名，且为必须参数**。如果操作是对于存储桶，则为空	String	是
Sign	是否返回带有签名的 Url，默认为 true	Boolean	否
Protocol	可选填为 `http:` 或 `https:`，默认为 `http:`（带冒号）	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Domain	存储桶访问域名，默认为 {BucketName-APPID}.cos.{Region}.myqcloud.com	String	否
Method	操作方法，例如 GET，POST，DELETE，HEAD 等 HTTP 方法，默认为 GET	String	否
Query	签名中要签入的请求参数，{key: 'val'} 的格式	Object	否
Headers	签名中要签入的请求头部，{key: 'val'} 的格式	Object	否
Expires	签名几秒后失效，默认为 900 秒	Number	否

返回值说明

返回值是一个字符串，有以下两种情况：

1. 如果签名计算可以同步计算（例如，实例化传入了 SecretId 和 SecretKey），则默认返回带签名的 Url。
2. 否则返回不带签名的 Url。

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- Url	计算得到的 Url	String

服务端加密

最近更新时间：2022-03-04 11:09:28

简介

本文档提供关于如何使用在上传对象时开启服务端加密。服务端加密的密钥分为三种：

- COS 托管加密密钥
- 客户提供的加密密钥
- KMS 托管加密密钥

使用 COS 托管加密密钥的服务端加密（SSE-COS）保护数据

由腾讯云 COS 托管主密钥和管理数据。COS 会帮助您在数据写入数据中心时自动加密，并在您取用该数据时自动解密。目前支持使用 COS 主密钥对数据进行 AES-256 加密。

JavaScript SDK 通过调用 API 时传入 `ServerSideEncryption` 参数完成。

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: 'hello!',
ServerSideEncryption: 'AES256',
}, function(err, data) {
console.log(err || data);
});
```

使用客户提供的加密密钥的服务端加密 （SSE-C）保护数据

加密密钥由用户自己提供，用户在上传对象时，COS 将使用用户提供的加密密钥对用户的数据进行 AES-256 加密。JavaScript SDK 通过调用 API 时传入 `SSECustomerKey` 参数完成。

注意：

- 该加密所运行的服务需要使用 HTTPS 请求。
- customerKey：用户提供的密钥，传入一个32字节的字符串的base64编码，字符串支持数字、字母、字符的组合，不支持中文。
- 如果上传源文件（putObject）使用了该参数，那么在使用 headObject（查询）、getObject（下载）、multipartInit（初始化分块上传）、multipartUpload（上传分块）、putObjectCopy（复制）时对源对象操作

版权所有：腾讯云计算（北京）有限责任公司

的时候也要调用该方法。

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: 'hello!',
SSECustomerAlgorithm: 'AES256',
SSECustomerKey: 'MDEyMzQ1Njc4OUFCQ0RFRjAxMjM0NTY3ODlBQkNERUY',
SSECustomerKeyMD5: 'U5L61r7jcwdNvT7frmUG8g==',
}, function(err, data) {
console.log(err || data);
});
```

使用 KMS 托管加密密钥的服务端加密（SSE-KMS）保护数据

SSE-KMS 加密即使用 KMS 托管密钥的服务端加密。KMS 是腾讯云推出的一款安全管理类服务，使用经过第三方认证的硬件安全模块 HSM（Hardware Security Module）来生成和保护密钥。它能够帮助用户轻松创建和管理密钥，满足用户多应用多业务的密钥管理需求以及满足监管和合规要求。关于如何开通 KMS 服务请参考：服务端加密概述。

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: 'hello!',
ServerSideEncryption: 'cos/kms', /* 上传或复制对象（包括简单上传/复制与分块上传/复制）时必选，下载对象时不能指定此头部 */
// SSEKMSKeyId: 用于指定 KMS 的用户主密钥 CMK，如不指定，则使用 COS 默认创建的 CMK，更多详细信息可参见 SSE-KMS 加密
SSEKMSKeyId: 'xxxx', /* 这里填入自己的key，非必须 */
// SSEContext: 用于指定加密上下文，值为 JSON 格式加密上下文键值对的 Base64 编码。
SSEContext: 'eyJhIjoiYXNkZmEiLCJiIjoiMTIzMzIxIn0=', /* 这里指定加密上下文，需要自行使用Base64编码，非必须 */
}, function(err, data) {
console.log(err || data);
});
```

预请求跨域配置

最近更新时间：2022-01-10 15:03:55

简介

本文档提供关于预请求跨域配置操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
Options Object	预请求跨域配置	用预请求来确认是否可以发送真正的跨域请求

功能说明

OPTIONS Object 接口实现对对象进行跨域访问配置的预请求。即在发送跨域请求之前会发送一个 OPTIONS 请求并带上特定的来源域，HTTP 方法和 HEADER 信息等给 COS，以决定是否可以发送真正的跨域请求。当 CORS 配置不存在时，请求返回403 Forbidden。**用户可以通过 PUT Bucket cors 接口来开启存储桶的 CORS 支持。**

使用示例

```
cos.optionsObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Origin: 'https://www.qq.com', /* 必须 */
AccessControlRequestMethod: 'PUT', /* 必须 */
AccessControlRequestHeaders: 'origin,accept,content-type' /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Origin	模拟跨域访问的请求来源域名	String	是
AccessControlRequestMethod	模拟跨域访问的请求 HTTP 方法	String	是
AccessControlRequestHeaders	模拟跨域访问的请求头部	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- headers	请求返回的头部信息	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- AccessControlAllowOrigin	模拟跨域访问的请求来源域名，中间用逗号间隔，当来源不允许的时候，此 Header 不返回，例如 `*`	String
- AccessControlAllowMethods	模拟跨域访问的请求 HTTP 方法，中间用逗号间隔，当请求方法不允许的时候，此 Header 不返回，例如 PUT，GET，POST，DELETE，HEAD	String
- AccessControlAllowHeaders	模拟跨域访问的请求头部，中间用逗号间隔，当模拟任何请求头部不允许的时候，此 Header 不返回该请求头部。例如：accept，content-type，origin，authorization	String
- AccessControlExposeHeaders	跨域支持返回头部，中间用逗号间隔。例如：ETag	String
- AccessControlMaxAge	设置 OPTIONS 请求得到结果的有效期。例如：3600	String
- OptionsForbidden	OPTIONS 请求是否被禁止，如果返回的 HTTP 状态码为403，则为 true	Boolean

版权所有：腾讯云计算（北京）有限责任公司

单链接限速

最近更新时间：2022-01-10 15:03:56

简介

本文档提供关于调用上传下载接口时对链接进行限速。

使用说明

限速值设置范围为**819200 - 838860800**，单位默认为 bit/s，即100KB/s - 100MB/s，如果超出该范围将返回400错误。

> 说明：
> 关于下载对象的限速说明，请参见 单链接限速。

示例一：上传时对单链接限速

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
StorageClass: 'STANDARD',
Body: fileObject, // 上传文件对象
Headers: {
'x-cos-traffic-limit': 819200, // 限速值设置范围为819200 - 838860800，即100KB/s - 10
0MB/s，如果超出该范围将返回400错误。
},
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

示例二：下载时对单链接限速

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
```

```
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Headers: {
'x-cos-traffic-limit': 819200, // 限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，如果超出该范围将返回400错误。
},
}, function(err, data) {
console.log(err || data.Body);
});
```

文件处理
文件压缩

最近更新时间：2023-03-28 14:38:57

简介

本文档提供关于数据万象文件压缩的 API 概览以及 SDK 示例代码。

> 注意：
> COS Javascript SDK 版本需要大于等于 v1.3.1。

API	操作描述
提交多文件打包压缩任务	创建一个多文件打包压缩任务
查询多文件打包压缩结果	查询指定的多文件打包压缩任务结果

提交多文件打包压缩任务

功能说明

多文件打包压缩功能可以将您的多个文件，打包为 zip 等压缩包格式，以提交任务的方式进行多文件打包压缩，异步返回打包后的文件。

示例代码

```
function postFileCompressTask() {
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com/file_jobs';
var url = 'https://' + host;
var body = COS.util.json2xml({
Request: {
Tag: 'FileCompress', // 必须
```

```
Operation: {
FileCompressConfig: {
Flatten: '0', // 文件打包时，是否需要去除源文件已有的目录结构.0:不需要;1:需要
Format: 'zip', // 打包压缩的类型，有效值：zip、tar、tar.gz
// UrlList、Prefix、Key 三者仅能选择一个，不能都为空，也不会同时生效
// UrlList: '', // 索引文件的对象地址
Prefix: 'testCompress/', // 目录前缀
// Key: [], // 支持对存储桶中的多个文件进行打包，个数不能超过 1000，总大小不超过50G，否则会导致任务失败
},
Output: {
Bucket: config.Bucket, // 保存压缩后文件的存储桶
Region: config.Region, // 保存压缩后文件的存储桶地域
Object: 'testCompress/compressed.zip', // 压缩后文件的文件名
},
UserData: '',
},
// QueueId: '', // 任务所在的队列 ID
// CallBack: 'http://callback.demo.com', // 任务回调的地址
// CallBackFormat: 'JSON', // 任务回调格式
// CallBackType: 'Url', // 任务回调类型，Url 或 TDMQ，默认 Url
}
});
cos.request({
Method: 'POST',
Key: 'file_jobs',
Url: url,
Body: body,
ContentType: 'application/xml',
},
function(err, data){
console.log(err || data);
});
}
postFileCompressTask();
```

参数说明

Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	表示任务的类型，多文件打包压缩默认为：FileCompress。	String	是

节点名称（关键字）	父节点	描述	类型	是否必选
Operation	Request	包含文件打包压缩的处理规则。	Container	是
QueueId	Request	任务所在的队列 ID。	String	否
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式。	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型。	String	否
CallBack	Request	任务回调的地址，优先级高于队列的回调地址。	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情请参见 CallBackMqConfig	Container	否

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
FileCompressConfig	Request.Operation	指定文件打包压缩的处理规则。	Container	是
UserData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否
Output	Request.Operation	指定打包压缩后的文件保存的地址信息。	Container	是

Container 类型 FileCompressConfig 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

节点名称（关键字）	父节点	描述	类型	是否必选
Flatten	Request.Operation.FileCompressConfig	文件打包时，是否需要去除源文件已有的目录结构，有效值： • 0：不需要去除目录结构，打包后压缩包中的文件会保留原有的目录结构； • 1：需要，打包后压缩包内的文件会去除原有的目录结构，所有文件都在同一层级。例如：源文件 URL 为 https://domain/source/test.mp4，则源文件路径为 source/test.mp4，如果为 1，则 ZIP 包中该文件路径为 test.mp4；如果为0， ZIP 包中该文件路径为 source/test.mp4。	String	是
Format	Request.Operation.FileCompressConfig	打包压缩的类型，有效值：zip、tar、tar.gz。	String	是
UrlList	Request.Operation.FileCompressConfig	• 支持将需要打包的文件整理成索引文件，后台将根据索引文件内提供的文件url，打包为一个压缩包文件。索引文件需要保存在当前存储桶中，本字段需要提供索引文件的对象地址，例如：/test/index.csv。 • 索引文件格式：仅支持 CVS 文件，一行一条URL（仅支持本存储桶文件），如有多列字段，默认取第一列作为URL。最多不超过10000个文件, 总大小不超过50G, 否则会导致任务失败。	String	否

节点名称（关键字）	父节点	描述	类型	是否必选
Prefix	Request.Operation.FileCompressConfig	支持对存储桶中的某个前缀进行打包，如果需要对某个目录进行打包，需要加/，例如test目录打包，则值为：test/。最多不超过10000个文件，总大小不超过50G，否则会导致任务失败。	String	否
Key	Request.Operation.FileCompressConfig	支持对存储桶中的多个文件进行打包，个数不能超过 1000，总大小不超过50G，否则会导致任务失败。	String 数组	否

注意：
UrlList、Prefix、Key 三者仅能选择一个，不能都为空，也不会同时生效。如果填了多个，会按优先级 UrlList > Prefix > Key 取最高优先级执行。

Container 类型 Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域。	String	是
Bucket	Request.Operation.Output	保存压缩后文件的存储桶。	String	是
Object	Request.Operation.Output	压缩后文件的文件名	String	是

返回结果说明

详情请参见 提交多文件打包压缩任务。

查询多文件打包压缩结果

功能说明

查询一个文件处理任务,根据任务 ID 查询任务详情。

示例代码

```
function getFileCompressTask() {
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
var jobId = 'xxxxxx'; // 提交文件压缩任务后会返回当前任务的 jobId
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com/file_jobs/' +
jobId;
var url = 'https://' + host;
cos.request({
Method: 'GET',
Key: 'file_jobs/' + jobId,
Url: url,
},
function(err, data){
console.log(err || data);
});
}
getFileCompressTask();
```

参数说明

参数名称	描述	类型	是否必选
jobId	要查询的任务 ID	String	是

返回结果说明

详情请参见 查询多文件打包压缩结果。

文件哈希值计算

最近更新时间：2023-03-28 14:38:57

简介

本文档提供关于数据万象文件哈希值计算的 API 概览以及 SDK 示例代码。

> 注意：
> COS Javascript SDK 版本需要大于等于 v1.3.1。

API	操作描述
提交哈希值计算任务	创建一个文件处理哈希值计算任务
查询哈希值计算结果	查询一个文件处理哈希值计算结果任务

提交哈希值计算任务

功能说明

以提交任务的方式进行文件哈希值计算，异步返回计算得到的哈希值。

示例代码

```
function postFileHashTask() {
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com/file_jobs';
var url = 'https://' + host;
var body = COS.util.json2xml({
Request: {
Tag: 'FileHashCode', // 必须
Input: {
Object: 'test/1.pdf', // 文件名，取值为文件在当前存储桶中的完整名称
},
Operation: {
```

```
FileHashCodeConfig: {
Type: 'MD5', // 哈希值的算法类型，有效值：MD5、SHA1、SHA256
AddToHeader: 'false', // 是否将计算得到的哈希值添加至文件自定义header，有效值：true、false，默认值为 false。
},
// UserData: '', // 透传用户信息，可打印的 ASCII 码，长度不超过1024
},
// QueueId: '', // 任务所在的队列 ID
// CallBack: 'http://callback.demo.com', // 任务回调的地址
// CallBackFormat: 'JSON', // 任务回调格式
// CallBackType: 'Url', // 任务回调类型，Url 或 TDMQ，默认 Url
}
});
cos.request({
Method: 'POST',
Key: 'file_jobs',
Url: url,
Body: body,
ContentType: 'application/xml',
},
function(err, data){
console.log(err || data);
});
}
postFileHashTask();
```

参数说明

Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	表示任务的类型，哈希值计算任务类型为：FileHashCode。	String	是
Operation	Request	包含哈希值计算的处理规则。	Container	是
QueueId	Request	任务所在的队列 ID。	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式。	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型。	String	否
CallBack	Request	任务回调的地址，优先级高于队列的回调地址。	String	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情请参见 CallBackMqConfig	Container	否

Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	文件名，取值为文件在当前存储桶中的完整名称。	String	是

Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
FileHashCodeConfig	Request.Operation	指定哈希值计算的处理规则。	Container	是
UserData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否

FileHashCodeConfig 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Type	Request.Operation.FileHashCodeConfig	哈希值的算法类型，有效值：MD5、SHA1、SHA256。	String	是
AddToHeader	Request.Operation.FileHashCodeConfig	是否将计算得到的哈希值添加至文件自定义 header, 有效值：true、false，默认值为 false。自定义header根据 `Type` 的值变化，例如 `Type` 值为MD5 时，自定义 heade r为 x-cos-meta-md5。	String	否

返回结果说明

详情请参见 提交哈希值计算任务。

查询哈希值计算结果

功能说明

查询一个文件处理任务，根据任务 ID 查询任务详情。

示例代码

```
function getFileHashTask() {
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
var jobId = 'xxxxxx'; // 提交文件哈希值计算任务后会返回当前任务的 jobId
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com/file_jobs/' +
jobId;
var url = 'https://' + host;
cos.request({
Method: 'GET',
Key: 'file_jobs/' + jobId,
Url: url,
},
function(err, data){
console.log(err || data);
});
}
getFileHashTask();
```

参数说明

参数名称	描述	类型	是否必选
jobId	要查询的任务 ID	String	是

返回结果说明

详情请参见 查询哈希值计算结果。

文件解压

最近更新时间：2023-03-28 14:38:57

简介

本文档提供关于数据万象文件解压的 API 概览以及 SDK 示例代码。

> 注意：
> COS Javascript SDK 版本需要大于等于 v1.3.1。

API	操作描述
提交文件解压任务	创建一个文件处理解压任务
查询文件解压结果	查询一个文件处理解压任务

提交文件解压任务

功能说明

以提交任务的方式进行文件解压缩，异步返回压缩包内被解压出来的文件。

示例代码

```
function postFileUnCompressTask() {
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com/file_jobs';
var url = 'https://' + host;
var body = COS.util.json2xml({
Request: {
Tag: 'FileUncompress', // 必须
Input: {
Object: 'compressed.zip', // 文件名，取值为文件在当前存储桶中的完整名称
},
Operation: {
```

版权所有：腾讯云计算（北京）有限责任公司

```
FileUncompressConfig: {
Prefix: '', // 指定解压后输出文件的前缀，不填则默认保存在存储桶根路径
PrefixReplaced: '0', // 指定解压后的文件路径是否需要替换前缀，默认0
},
Output: {
Bucket: config.Bucket, // 保存解压后文件的存储桶
Region: config.Region, // 保存解压后文件的存储桶地域
},
},
// QueueId: '', // 任务所在的队列 ID
// CallBack: 'http://callback.demo.com', // 任务回调的地址
// CallBackFormat: 'JSON', // 任务回调格式
// CallBackType: 'Url', // 任务回调类型，Url 或 TDMQ，默认 Url
}
});
cos.request({
Method: 'POST',
Key: 'file_jobs',
Url: url,
Body: body,
ContentType: 'application/xml',
},
function(err, data){
console.log(err || data);
});
}
postFileUnCompressTask();
```

参数说明

Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	表示任务的类型，文件解压默认为：FileUncompress。	String	是
Input	Request	包含待操作的文件信息。	Container	是
Operation	Request	包含文件解压的处理规则。	Container	是
QueueId	Request	任务所在的队列 ID。	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式。	String	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型。	String	否
CallBack	Request	任务回调的地址，优先级高于队列的回调地址。	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	文件名，取值为文件在当前存储桶中的完整名称。	String	是

Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
FileUncompressConfig	Request.Operation	指定文件解压的处理规则。	Container	是
UserData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否
Output	Request.Operation	指定解压后的文件保存的存储桶信息。	Container	是

FileUncompressConfig 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Prefix	Request.Operation.FileUncompressConfig	指定解压后输出文件的前缀，不填则默认保存在存储桶根路径。	String	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
PrefixReplaced	Request.Operation.FileUncompressConfig	指定解压后的文件路径是否需要替换前缀，有效值： • 0：不添加额外的前缀，解压缩将保存在Prefix指定的路径下（不会保留压缩包的名称，仅将压缩包内的文件保存至指定的路径）。 • 1：以压缩包本身的名称作为前缀，解压缩将保存在Prefix指定的路径下。 • 2：以压缩包完整路径作为前缀，此时如果不指定Prefix，就是解压到压缩包所在的当前路径（包含压缩包本身名称）。 • 默认值为0。	String	否

说明：
假设压缩包名为 test.zip，压缩包内的文件为“图片.jpg”，存储在了存储桶 A 的 123 目录下，则压缩包的完整路径为：123/test.zip。
对该压缩包进行解压缩，指定解压缩到存储桶A，并指定 `Prefix` 为 456，则不同的 `PrefixReplaced` 值对应解压后的文件保存情况如下：

- 0：“图片.jpg”会保存在 456 目录下，“图片.jpg”的完整路径为 456/图片.jpg；
- 1：“图片.jpg”会以 test 为前缀保存在 456 目录下，“图片.jpg”的完整路径为 456/test/图片.jpg；
- 2：“图片.jpg”会以 123/test 为前缀保存在 456 目录下，“图片.jpg”的完整路径为 456/123/test/图片.jpg。

Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域。	String	是

节点名称（关键字）	父节点	描述	类型	是否必选
Bucket	Request.Operation.Output	保存解压后文件的存储桶。	String	是

返回结果说明

详情请参见 提交文件解压任务。

查询文件解压结果

功能说明

查询一个文件处理任务,根据任务 ID 查询任务详情。

示例代码

```
function getFileUnCompressTask() {
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
var jobId = 'xxxxxx'; // 提交文件解压任务后会返回当前任务的 jobId
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com/file_jobs/' +
jobId;
var url = 'https://' + host;
cos.request({
Method: 'GET',
Key: 'file_jobs/' + jobId,
Url: url,
},
function(err, data){
console.log(err || data);
});
}
getFileUnCompressTask();
```

参数说明

参数名称	描述	类型	是否必选
jobId	要查询的任务 ID	String	是

返回结果说明

详情请参见 查询文件解压结果。

版权所有：腾讯云计算（北京）有限责任公司

异地容灾
版本控制

最近更新时间：2020-04-27 18:44:16

简介

本文档提供关于版本控制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

设置版本控制

功能说明

PUT Bucket versioning 接口实现启用或者暂停存储桶的版本控制功能。

1. 如果您从未在存储桶上启用过版本控制，则 GET Bucket versioning 请求不返回版本状态值。
2. 开启版本控制功能后，只能暂停，不能关闭。
3. 设置版本控制状态值为 Enabled 或者 Suspended，表示开启版本控制和暂停版本控制。
4. 设置存储桶的版本控制功能，您需要有存储桶的写权限。

请求示例

```
cos.putBucketVersioning({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
VersioningConfiguration: {
Status: "Enabled"
}
}, function (err, data) {
console.log(err || data);
});
```

版权所有：腾讯云计算（北京）有限责任公司

参数说明

参数名称	描述	类型	是否必填
Bucket	开启或暂停版本控制的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
VersioningConfiguration	定义存储桶的版本控制配置信息	Object	是
- Status	版本控制是否打开的状态，枚举值：Enabled、Suspended。 • Enabled 表示打开 • Suspended 表示暂停	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询版本控制

功能说明

查询存储桶的版本控制信息。

请求示例

```
cos.getBucketVersioning({
Bucket: 'examplebucket-1250000000', /* 必须 */
```

```
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function (err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询版本控制的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- VersioningConfiguration	存储桶的版本控制配置信息，若从未开启过，则为空对象 '{}'	Object
- - Status	版本控制是否打开的状态，枚举值：Enabled、Suspended	String

版权所有：腾讯云计算（北京）有限责任公司

跨地域复制

最近更新时间：2020-05-07 18:02:46

简介

本文档提供关于跨地域复制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

设置跨地域复制

功能说明

设置存储桶的跨地域复制规则。

> 存储桶必须开启版本控制才能使用跨地域复制功能。

请求示例

```
cos.putBucketReplication({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
ReplicationConfiguration: { /* 必须 */
Role: "qcs::cam::uin/100000000001:uin/100000000001",
Rules: [{
ID: "1",
Status: "Enabled",
Prefix: "sync/",
Destination: {
Bucket: "qcs::cos:ap-beijing::destinationbucket-1250000000",
StorageClass: "Standard",
}
}]
```

```
}
}, function (err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型
Bucket	源存储桶，格式：BucketName-APPID	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
ReplicationConfiguration	定义跨地域复制规则	Object
- Role	复制过程以什么角色的身份 格 式：`qcs::cam::uin/100000000001:uin/100000000011` 其中100000000001是主账号，100000000011是子账号	Object
- Rules	复制具体规则列表	ObjectArray
- - ID	标注具体 Rule 的 ID	String
- - Status	标识 Rule 是否生效，枚举值：Enabled、Disabled	String
- - Prefix	前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空	String
- - Destination	目标存储桶信息	Object
- - - Bucket	目标存储桶的名称 格式：`qcs::cos:<Region>::<BucketName-APPID>` 例如：`qcs::cos:ap-beijing::destinationbucket-1250000000`	Object
- - - StorageClass	复制后的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD	Object

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询跨地域复制

功能说明

查询存储桶的跨地域复制规则。

请求示例

```
cos.getBucketReplication({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"ReplicationConfiguration": {
"Role": "qcs::cam::uin/100000000001:uin/100000000001",
"Rules": {
"ID": "1",
"Status": "Enabled",
"Prefix": "sync/",
"Destination": {
"Bucket": "qcs::cos:ap-beijing::destinationbucket-1250000000",
"StorageClass": "Standard"
}
}
},
"statusCode": 200,
```

版权所有：腾讯云计算（北京）有限责任公司

```
"headers": {}
}
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- ReplicationConfiguration	跨地域复制规则	Object
- - Role	复制过程以什么角色的身份 格 式：`qcs::cam::uin/100000000001:uin/100000000011` 其中100000000001是主账号，100000000011是子账号	Object
- - Rules	复制具体规则列表	ObjectArray
- - - ID	标注具体 Rule 的 ID	String
- - - Status	标识 Rule 是否生效，枚举值：Enabled、Disabled	String
- - - Prefix	前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空	String
- - - Destination	目标存储桶信息	Object
- - - - Bucket	目标存储桶的名称 格式：`qcs::cos:<Region>::<BucketName-APPID>` 例如：`qcs::cos:ap-beijing::destinationbucket-1250000000`	Object

参数名	参数描述	类型
- - - - StorageClass	复制后的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD	Object

删除跨地域复制

功能说明

删除存储桶的跨地域复制规则。

请求示例

```
cos.deleteBucketReplication({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

数据管理
生命周期

最近更新时间：2020-04-27 18:43:33

简介

本文档提供关于生命周期的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket lifecycle	设置生命周期	设置存储桶的生命周期管理的配置
GET Bucket lifecycle	查询生命周期	查询存储桶生命周期管理的配置
DELETE Bucket lifecycle	删除生命周期	删除存储桶生命周期管理的配置

设置生命周期

功能说明

设置指定存储桶的生命周期配置信息（PUT Bucket lifecycle）。

请求示例

示例一：上传30天后，当前版本沉降至低频存储。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "1",
"Status": "Enabled",
"Filter": {},
"Transition": {
"Days": "30",
"StorageClass": "STANDARD_IA"
}
}],
}, function(err, data) {
console.log(err || data);
});
```

示例二：指定目录前缀 `dir/`，上传90天后，当前版本沉降至归档存储。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "2",
"Filter": {
"Prefix": "dir/",
},
"Status": "Enabled",
"Transition": {
"Days": "90",
"StorageClass": "ARCHIVE"
}
}],
}, function(err, data) {
console.log(err || data);
});
```

示例三：上传180天后，清理过期文件删除标记。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "3",
"Status": "Enabled",
"Filter": {},
"Expiration": {
"Days": "180"
}
}],
}, function(err, data) {
console.log(err || data);
});
```

示例四：上传30天后，删除碎片。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "4",
"Status": "Enabled",
```

版权所有：腾讯云计算（北京）有限责任公司

```
"Filter": {},
"AbortIncompleteMultipartUpload": {
"DaysAfterInitiation": "30"
}
}],
}, function(err, data) {
console.log(err || data);
});
```

示例五：历史版本生成30天后沉降至归档存储。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "5",
"Status": "Enabled",
"Filter": {},
"NoncurrentVersionTransition": {
"NoncurrentDays": "30",
"StorageClass": 'ARCHIVE'
}
}],
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	设置生命周期的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Rules	指定生命周期规则列表	ObjectArray	是
- ID	规则的唯一标识 ID	String	是
- Status	规则的开启状态，枚举值：Enabled、Disabled	String	是
- Filter	指定过滤条件	Object	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
- - Prefix	规则要匹配上的对象键前缀	String	否
- Transition	规则转换属性，表示对象存储级别何时转换为 Standard_IA 或 Archive	Object	否
- - Days	指明规则对应的动作在对象最后的修改日期过后多少天执行，该字段有效值是非负整数，最大支持3650天	Number	是
- - StorageClass	表示规则对应的动作执行后变更对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD	String	否
- NoncurrentVersionTransition	指明历史版本对象转换属性	ObjectArray	否
- - NoncurrentDays	指明历史版本对象在该值确定的生效天数后进行转换	Number	是
- - StorageClass	表示规则对应的动作执行后变更对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE	String	是
- Expiration	规则过期属性	Object	否
- - ExpiredObjectDeleteMarker	删除过期对象删除标记，枚举值 true，false，与 Days 不可并存	Boolean	是
- - Days	指明规则对应的动作在对象最后的修改日期过后多少天进行删除操作，与 ExpiredObjectDeleteMarker 不可并存	Number	是
- AbortIncompleteMultipartUpload	表示删除碎片	Object	否
- - DaysAfterInitiation	碎片在该值所确定的生效天数后被删除，按文件上传时间开始算，必须为正整数	Number	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询生命周期

功能说明

GET Bucket lifecycle 接口可以查询存储桶生命周期管理的配置。

请求示例

```
cos.getBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"Rules": [{
"ID": "1",
"Filter": "",
"Status": "Enabled",
"Transition": {
"Days": "30",
"StorageClass": "STANDARD_IA"
}
}, {
"ID": "2",
"Filter": {
```

```
"Prefix": "dir/"
},
"Status": "Enabled",
"Transition": {
"Days": "90",
"StorageClass": "ARCHIVE"
}
}, {
"ID": "3",
"Filter": "",
"Status": "Enabled",
"Expiration": {
"Days": "180"
}
}, {
"ID": "4",
"Filter": "",
"Status": "Enabled",
"AbortIncompleteMultipartUpload": {
"DaysAfterInitiation": "30"
}
}, {
"ID": "5",
"Filter": "",
"Status": "Enabled",
"NoncurrentVersionTransition:": {
"NoncurrentDays": "30",
"StorageClass": "ARCHIVE"
}
}],
"statusCode": 200,
"headers": {}
}
```


参数说明

参数名称	描述	类型	是否必填
Bucket	查询生命周期的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Rules	指定生命周期规则列表	ObjectArray
- - ID	规则的唯一标识 ID	String
- - Status	规则的开启状态，枚举值：Enabled、Disabled	String
- - Filter	指定过滤条件	Object
- - - Prefix	规则要匹配上的对象键前缀	String
- - Transition	规则转换属性，表示对象存储级别何时转换为 Standard_IA 或 Archive	ObjectArray
- - - Days	指明规则对应的动作在对象最后的修改日期过后多少天执行，该字段有效值是非负整数，最大支持3650天	Number
- - - StorageClass	表示规则对应的动作执行后变更对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE	String
- - NoncurrentVersionTransition	指明历史版本对象转换属性	ObjectArray
- - - NoncurrentDays	指明历史版本对象在该值确定的生效天数后进行转换	Number
- - - StorageClass	表示规则对应的动作执行后变更对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE	String
- - Expiration	规则过期属性	Object
- - - ExpiredObjectDeleteMarker	删除过期对象删除标记，枚举值 true，false，与 Days 不可并存	Boolean

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- - - Days	指明规则对应的动作在对象最后的修改日期过后多少天进行删除操作，与 ExpiredObjectDeleteMarker 不可并存	Number
- - AbortIncompleteMultipartUpload	表示删除碎片	Object
- - - DaysAfterInitiation	碎片在该值所确定的生效天数后被删除，按文件上传时间开始算，必须为正整数	Number

删除生命周期

功能说明

DELETE Bucket lifecycle 接口可以删除存储桶生命周期管理的配置。

请求示例

```
cos.deleteBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	被删除生命周期配置的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object

参数名称	描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

存储桶标签

最近更新时间：2020-04-27 18:43:34

简介

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket tagging	设置存储桶标签	为已存在的存储桶设置标签
GET Bucket tagging	查询存储桶标签	查询指定存储桶下已有的存储桶标签
DELETE Bucket tagging	删除存储桶标签	删除指定的存储桶标签

设置存储桶标签

功能说明

为已存在的存储桶设置标签。

请求示例

```
cos.putBucketTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Tagging: {
Tags: [
{"Key": "k1", "Value": "v1"},
{"Key": "k2", "Value": "v2"}
]
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	设置存储桶标签的存储桶，格式：BucketName-APPID	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Tagging	标签信息	Object	是
- Tags	标签信息	ObjectArray	是
- - Key	标签名	String	是
- - Value	标签值	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询存储桶标签

功能说明

查询指定存储桶下已有的存储桶标签。

请求示例

```
cos.getBucketTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"Tags": [
{"Key": "k1", "Value": "v1"},
{"Key": "k2", "Value": "v2"}
],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	查询存储桶标签的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Tags	标签信息	ObjectArray
- - Key	标签名	String
- - Value	标签值	String

版权所有：腾讯云计算（北京）有限责任公司

删除存储桶标签

功能说明

删除指定的存储桶标签。

请求示例

```
cos.deleteBucketTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	描述	类型	是否必填
Bucket	删除存储桶标签的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

静态网站

最近更新时间：2020-04-27 18:43:34

简介

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket website	设置静态网站	为已存在的存储桶设置静态网站配置信息
GET Bucket website	查询静态网站	查询指定存储桶的静态网站配置信息
DELETE Bucket website	删除静态网站	删除指定存储桶的静态网站配置信息

设置静态网站

功能说明

为已存在的存储桶配置静态网站。

请求示例

```
cos.putBucketWebsite({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
WebsiteConfiguration: {
IndexDocument: {
Suffix: "index.html"
},
ErrorDocument: {
Key: "error.html"
},
RedirectAllRequestsTo: {
Protocol: "https"
},
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置静态网站的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
WebsiteConfiguration	静态网站配置，包括索引文档、错误文档、协议转换和重定向规则	Object	是
- IndexDocument	索引文档	Object	是
- - Suffix	指定索引文档	String	是
- ErrorDocument	错误文档	Object	否
- - Key	指定通用错误返回	String	否
- RedirectAllRequestsTo	重定向所有请求	Object	否
- - Protocol	指定全站重定向的协议，只能设置为 https	String	否
- RoutingRules	设置重定向规则，最多设置100条	ObjectArray	否
- - Condition	指定重定向发生的条件，前缀匹配重定向和错误码重定向只能指定一个	Object	否
- - - HttpErrorCodeReturnedEquals	指定重定向错误码，只支持配置4XX返回码，优先级高于ErrorDocument	String	否
- - - KeyPrefixEquals	指定前缀重定向的路径，替换指定的 folder/	String	否
- - Redirect	指定满足重定向 conditon 时重定向的具体替换规则	Object	否
- - - ReplaceKeyWith	替换整个 Key 为指定的内容	String	否
- - - ReplaceKeyPrefixWith	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询静态网站配置

功能说明

查询与指定存储桶关联的静态网站配置信息。

请求示例

```
cos.getBucketWebsite({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"WebsiteConfiguration": {
"IndexDocument": {
"Suffix": "index.html"
},
"ErrorDocument": {
"Key": "error.html"
},
"RedirectAllRequestsTo": {
"Protocol": "https"
},
},
```

```
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询静态网站配置的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
- WebsiteConfiguration	静态网站配置，包括索引文档、错误文档、协议转换和重定向规则	Object
- - IndexDocument	索引文档	Object
- - - Suffix	指定索引文档	String
- - ErrorDocument	错误文档	Object
- - - Key	指定通用错误返回	String
- - RedirectAllRequestsTo	重定向所有请求	Object
- - - Protocol	指定全站重定向的协议，只能设置为 https	String

参数名称	描述	类型
- - RoutingRules	设置重定向规则，最多设置100条	ObjectArray
- - - Condition	指定重定向发生的条件，前缀匹配重定向和错误码重定向只能指定一个	Object
- - - - HttpErrorCodeReturnedEquals	指定重定向错误码，只支持配置4XX 返回码，优先级高于 ErrorDocument	String
- - - - KeyPrefixEquals	指定前缀重定向的路径，替换指定的 folder/	String
- - - Redirect	指定满足重定向 conditon 时重定向的具体替换规则	Object
- - - - ReplaceKeyWith	替换整个 Key 为指定的内容	String
- - - - ReplaceKeyPrefixWith	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String

删除静态网站配置

功能说明

删除指定存储桶中的静态网站配置。

请求示例

```
cos.deleteBucketWebsite({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	被删除静态网站配置的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

对象标签

最近更新时间：2022-10-25 14:26:55

简介

本文档提供关于对象标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Object tagging	设置对象标签	为已存在的对象设置标签
GET Object tagging	查询对象标签	查询指定对象下已有的对象标签
DELETE Object tagging	删除对象标签	删除指定的对象标签

设置对象标签

功能说明

为已存在的对象设置标签。

请求示例

```
cos.putObjectTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.png', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Tags: [
{"Key": "k1", "Value": "v1"},
{"Key": "k2", "Value": "v2"}
],
// VersionId: 'MTg0NDUwOTMyODM1MDg2MTA1xxx' /* 对象版本 ID，非必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	设置对象标签的存储桶，格式：BucketName-APPID	String	是

参数名	参数描述	类型	是否必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
Tags	标签信息	ObjectArray	是
- Key	标签名	String	是
- Value	标签值	String	是
VersionId	当启用版本控制时，指定要操作的对象版本 ID，如不指定则将添加标签到最新版本的对象	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询对象标签

功能说明

查询指定对象下已有的对象标签。

请求示例

```
cos.getObjectTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
```

```
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.png', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"Tags": [
{"Key": "k1", "Value": "v1"},
{"Key": "k2", "Value": "v2"}
],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	查询对象标签的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- headers	请求返回的头部信息	Object
- Tags	标签信息	ObjectArray
- - Key	标签名	String
- - Value	标签值	String

删除对象标签

功能说明

删除指定的对象标签。

请求示例

```
cos.deleteObjectTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.png', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	描述	类型	是否必填
Bucket	删除对象标签的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

访问管理
跨域访问

最近更新时间：2020-04-27 18:43:35

简介

本文档提供关于跨域访问的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域名访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域名访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域名访问配置信息

设置跨域配置

1. 如果要在前端修改跨域访问配置，需要该存储桶本身支持跨域，您可以在控制台进行跨域访问配置，详情请参见 快速入门 文档。
2. 在修改跨域访问配置时，请注意不要影响到当前的 Origin 下的跨域请求。

功能说明

设置存储桶的跨域名访问权限，您可以通过传入 XML 格式的配置文件来实现配置，文件大小限制为64KB。默认情况下，存储桶的持有者直接有权限使用该 API 接口，存储桶持有者也可以将权限授予其他用户。

请求示例

```
cos.putBucketCors({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
CORSRules: [{
"AllowedOrigin": ["*"],
"AllowedMethod": ["GET", "POST", "PUT", "DELETE", "HEAD"],
"AllowedHeader": ["*"],
"ExposeHeader": ["ETag", "x-cos-acl", "x-cos-version-id", "x-cos-delete-marker",
"x-cos-server-side-encryption"],
```

```
"MaxAgeSeconds": "5"
}]
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置跨域配置的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
CORSRules	说明跨域资源共享配置的所有信息列表	ObjectArray	否
- ID	配置规则的 ID	String	否
- AllowedMethods	允许的 HTTP 操作，枚举值：GET、PUT、HEAD、POST、DELETE 等	StringArray	是
- AllowedOrigins	允许的访问来源，支持通配符 `*`，格式为：`协议://域名[:端口]`，例如：`http://www.qq.com`	StringArray	是
- AllowedHeaders	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 `*`	StringArray	否
- ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	StringArray	否
- MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询跨域配置

功能说明

GET Bucket cors 接口实现查询存储桶的跨域名访问配置信息。（CORS 是一个 W3C 标准，全称是"跨域资源共享"（Cross-origin Resource Sharing））。默认情况下，存储桶的持有者直接有权限使用该 API 接口，存储桶持有者也可以将权限授予其他用户。

请求示例

```
cos.getBucketCors({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"CORSRules": [{
"MaxAgeSeconds": "5",
"AllowedOrigins": ["*"],
"AllowedHeaders": ["*"],
"AllowedMethods": ["GET", "POST", "PUT", "DELETE", "HEAD"],
"ExposeHeaders": ["ETag", "Content-Length", "x-cos-acl", "x-cos-version-id", "x-cos-request-id", "x-cos-delete-marker", "x-cos-server-side-encryption"]
}],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	查询跨域配置的存储桶，格式：BucketName-APPID	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- CORSRules	说明跨域资源共享配置的所有信息列表	ObjectArray
- - AllowedMethods	允许的 HTTP 操作，枚举值：GET、PUT、HEAD、POST、DELETE	StringArray
- - AllowedOrigins	允许的访问来源，支持通配符 `*` ，格式为： `协议://域名[:端口]` 例如： `http://www.qq.com`	StringArray
- - AllowedHeaders	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 `*`	StringArray
- - ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	StringArray
- - MaxAgeSeconds	设置 OPTIONS 跨域信息缓存秒数	String
- - ID	配置规则的 ID	String

删除跨域配置

功能说明

删除存储桶的跨域名访问配置信息。

1. 删除当前存储桶的跨域访问配置信息，会导致所有请求跨域失败，请谨慎操作。
2. 不推荐在浏览器端使用该方法。

请求示例

```
cos.deleteBucketCors({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	被删除跨域配置的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

存储桶策略

最近更新时间：2020-04-27 18:43:36

简介

本文档提供关于存储桶策略的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket policy	设置存储桶策略	设置指定存储桶的权限策略
GET Bucket policy	查询存储桶策略	查询指定存储桶的权限策略
DELETE Bucket policy	删除存储桶策略	删除指定存储桶的权限策略

设置存储桶策略

功能说明

设置指定存储桶的权限策略。

请求示例

```
cos.putBucketPolicy({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Policy: {
"version": "2.0",
"Statement": [{
"Effect": "allow",
"Principal": {
"qcs": ["qcs::cam::uin/100000000001:uin/100000000001"]
},
"Action": [
"name/cos:PutObject",
"name/cos:InitiateMultipartUpload",
"name/cos:ListMultipartUploads",
"name/cos:ListParts",
"name/cos:UploadPart",
"name/cos:CompleteMultipartUpload"
],
"Resource": ["qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"],
```

版权所有：腾讯云计算（北京）有限责任公司

```
}]
},
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	必
Bucket	设置存储桶策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Policy	权限策略，详情请参见 访问管理策略语法	Object	是
- version	版本号，固定2.0	String	是
- statement	权限策略生命列表	ObjectArray	是
- - effect	效力，枚举值：allow、deny	String	是
- - principal	身份信息	ObjectArray	是
- - - qcs	身份信息标识字符串 格 式：`qcs::cam::uin/100000000001:uin/100000000011` 其中100000000001 是主账号，100000000011是子账号	String	是
- - action	策略生效的相关 Action 列表，支持通配符 `*`	StringArray	是
- - resource	相关的资源标识字符串列表 格式：`qcs::cos:<Region>:uid/<AppId>:<ShortBucketName>/*` 例如：`qcs::cos:ap-beijing:uid/1250000000:examplebucket/*`	StringArray	是
- - condition	约束条件，可以不填，具体说明请参见 condition 说明	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询存储桶策略

功能说明

查询指定存储桶的权限策略。

请求示例

```
cos.getBucketPolicy({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"Policy": {
"version": "2.0",
"Statement": [{
"Action": [
"name/cos:PutObject",
"name/cos:InitiateMultipartUpload",
"name/cos:ListMultipartUploads",
"name/cos:ListParts",
"name/cos:UploadPart",
"name/cos:CompleteMultipartUpload"
],
```

版权所有：腾讯云计算（北京）有限责任公司

```
"Effect": "allow",
"Principal": {
"qcs": ["qcs::cam::uin/100000000001:uin/100000000001"]
},
"Resource": ["qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*"],
"Sid": "costs-153983319700000307620-46518-39"
}]
},
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询存储桶权限策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- Policy	权限策略，详情请参见 访问管理策略语法	Object
- - version	版本号，固定2.0	String
- - Statement	权限策略生命列表	ObjectArray
- - - effect	效力，枚举值：allow、deny	String
- - - principal	身份信息	ObjectArray
- - - - qcs	身份信息标识字符串 格式：`qcs::cam::uin/100000000001:uin/100000000011` 其中100000000001是主账号，100000000011是子账号	String
- - - action	策略生效的相关 Action 列表，支持通配符 `*`	StringArray

参数名	参数描述	类型
- - - resource	相关的资源标识字符串列表 格式：`qcs::cos:<Region>:uid/&ltAppId>:<ShortBucketName>/*` 例如：`qcs::cos:ap-beijing:uid/1250000000:examplebucket/*`	StringArray
- - - condition	约束条件，可以不填，具体说明请参见 condition 说明	ObjectArray

删除存储桶策略

功能说明

DELETE Bucket policy 请求删除指定存储桶的权限策略。

> 只有 Bucket 所有者有权限发起该请求。如果权限策略不存在，将返回204 No Content。

请求示例

```
cos.deleteBucketPolicy({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	被删除存储桶权限策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

自定义域名

最近更新时间：2020-05-07 18:02:46

简介

本文档提供关于自定义域名的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket domain	设置自定义域名	设置存储桶的自定义域名信息
GET Bucket domain	查询自定义域名	查询存储桶的自定义域名信息
DELETE Bucket domain	删除自定义域名	删除存储桶的自定义域名信息

设置自定义域名

功能说明

为已存在的存储桶绑定自定义域名。

请求示例

```
cos.putBucketDomain({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
DomainRule: [{
Status: "DISABLED",
Name: "www.example.com",
Type: "REST"
},
{
Status: "DISABLED",
Name: "www.example.net",
Type: "WEBSITE",
}]
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置自定义域名的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
DomainRule	自定义域名配置	Object	是
- Status	域名上线/下线状态，枚举值： ENABLED、DISABLED	String	是
- Name	用户的自定义域名	String	是
- Type	绑定的源站类型，枚举值：REST、WEBSITE	String	是
- ForcedReplacement	替换已存在的配置，枚举值：CNAME、TXT。填写则强制校验域名所有权后，再下发配置。	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询自定义域名

功能说明

查询与指定存储桶关联的自定义域名信息。

请求示例

```
cos.getBucketDomain({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"DomainRule": [{
"Status": "DISABLED",
"Name": "www.example.com",
"Type": "REST"
}, {
"Status": "DISABLED",
"Name": "www.example.net",
"Type": "WEBSITE"
}],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询自定义域名的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
- DomainRule	自定义域名配置	Object
- - Status	域名上线/下线状态，枚举值： ENABLED、DISABLED	String
- - Name	用户的自定义域名	String
- - Type	绑定的源站类型，枚举值：REST、WEBSITE	String
- - ForcedReplacement	替换已存在的配置，枚举值：CNAME、TXT。填写则强制校验域名所有权后，再下发配置。	String

删除自定义域名

功能说明

删除指定存储桶中的自定义域名配置。

请求示例

```
cos.deleteBucketDomain({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	被删除自定义域名的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

访问控制

最近更新时间：2021-06-24 10:38:35

简介

本文档提供关于存储桶、对象的访问控制列表（ACL）的 API 概览以及 SDK 示例代码。

存储桶 ACL

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶的访问权限控制列表（ACL）
GET Bucket acl	查询存储桶 ACL	查询指定存储桶的访问权限控制列表（ACL）

对象 ACL

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表
GET Object acl	查询对象 ACL	查询对象的访问控制列表

存储桶 ACL

设置存储桶 ACL

功能说明

PUT Bucket acl 接口用来设置指定存储桶访问权限控制列表（ACL）。

使用示例

设置存储桶公有读：

```
cos.putBucketAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
ACL: 'public-read'
}, function(err, data) {
console.log(err || data);
});
```

为某个用户赋予存储桶所有权限：

```
cos.putBucketAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
GrantFullControl: 'id="qcs::cam::uin/100000000001:uin/100000000001",id="qcs::ca
m::uin/100000000011:uin/100000000011"' // 100000000001是 uin
}, function(err, data) {
console.log(err || data);
});
```

通过 AccessControlPolicy 修改存储桶权限：

```
cos.putBucketAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
AccessControlPolicy: {
"Owner": { // AccessControlPolicy 里必须有 owner
"ID": 'qcs::cam::uin/100000000001:uin/100000000001' // 100000000001 是 Bucket 所属
用户的 Uin
},
"Grants": [{
"Grantee": {
"ID": "qcs::cam::uin/100000000011:uin/100000000011", // 100000000011 是 Uin
},
"Permission": "WRITE"
}]
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
ACL	定义存储桶的访问控制列表（ACL）属性。枚举值请参见 ACL 概述 文档中存储桶的预设 ACL 部分，例如 private，public-read 等，默认为 private	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
GrantRead	赋予被授权者读取存储桶的权限，格式：id="[OwnerUin]" 可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权 时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权 时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantWrite	赋予被授权者写入存储桶的权限，格式：id="[OwnerUin]" 可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权 时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权 时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantReadAcp	赋予被授权者读取存储桶的访问控制列表（ACL）和存储桶策略（Policy）的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权 时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权 时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantWriteAcp	赋予被授权者写入存储桶的访问控制列表（ACL）和存储桶策略（Policy）的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权 时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权 时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
GrantFullControl	赋予被授权者操作存储桶的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权 时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权 时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
AccessControlPolicy	跨域资源共享配置的所有信息列表	Object
- Owner	存储桶持有者的信息	Object
- - ID	存储桶持有者的完整 ID，格式为 `qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]` 例如 `qcs::cam::uin/100000000001:uin/100000000001`，其中100000000001为 uin	String
- Grants	被授权者信息与权限信息列表	ObjectA
- - Permission	授予的权限信息，可选项 READ、WRITE、READ_ACP、WRITE_ACP、FULL_CONTROL。枚举值详情参见 ACL 概述 文档中存储桶的操作部分	String
- - Grantee	被授权者的信息	Object
- - - ID	被授权者的完整 ID，格式为 `qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]` 例如 `qcs::cam::uin/100000000001:uin/100000000001`，其中100000000001为 uin	String
- - - DisplayName	被授权者的名称，一般填写成和 ID 一致的字符串	String
- - - URI	预设用户组，请参见 ACL 概述 文档中预设用户组部分，例如 `http://cam.qcloud.com/groups/global/AllUsers` 或 `http://cam.qcloud.com/groups/global/AuthenticatedUsers`	String

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询存储桶 ACL

功能说明

GET Bucket acl 接口用来查询存储桶的访问控制列表（ACL）。该 API 的请求者需要对存储桶有写入 ACL 权限。

使用示例

```
cos.getBucketAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"GrantFullControl": "",
"GrantWrite": "",
"GrantRead": "",
"GrantReadAcp": "id=\"qcs::cam::uin/100000000011:uin/100000000011\"",
"GrantWriteAcp": "id=\"qcs::cam::uin/100000000011:uin/100000000011\"",
"ACL": "private",
"Owner": {
"ID": "qcs::cam::uin/100000000001:uin/100000000001",
"DisplayName": "qcs::cam::uin/100000000001:uin/100000000001"
},
"Grants": [{
"Grantee": {
"ID": "qcs::cam::uin/100000000011:uin/100000000011",
```

```
"DisplayName": "qcs::cam::uin/100000000011:uin/100000000011"
},
"Permission": "READ"
}],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名	参数描述	类型	必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Numbe
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Numbe
- headers	请求返回的头部信息	Object
- ACL	定义存储桶的访问控制列表（ACL）属性。枚举值请参见 ACL 概述 文档中存储桶的预设 ACL 部分，例如 private，public-read 等，默认为 private	String
- GrantRead	具有读取存储桶权限的被授权者 ID 信息	String
- GrantWrite	具有写入存储桶权限的被授权者 ID 信息	String
- GrantReadAcp	具有读取存储桶的访问控制列表（ACL）和存储桶策略（Policy）权限的被授权者 ID 信息	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- GrantWriteAcp	具有写入存储桶的访问控制列表（ACL）和存储桶策略（Policy）权限的被授权者 ID 信息	String
- GrantFullControl	具有存储桶所有权限的被授权者 ID 信息	String
- Owner	存储桶持有者的信息	Object
- - DisplayName	存储桶持有者的名称	String
- - ID	存储桶持有者的完整 ID 格式为 `qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主帐号，<OwnerUin> 和 <SubUin> 是同一个值	String
- Grants	被授权者信息与权限信息列表	ObjectA
- - Permission	指明授予被授权者的权限信息，枚举值：READ、WRITE、READ_ACP、WRITE_ACP、FULL_CONTROL	String
- - Grantee	被授权者的信息	Object
- - - DisplayName	被授权者的名称	String
- - - ID	被授权者的完整 ID， 如果是主账号，格式 为 `qcs::cam::uin/<owneruin>:uin/<owneruin>` 或 `qcs::cam::anyone:anyone` （指代所有用户） 如果是子帐号，格式为 `qcs::cam::uin/<owneruin>:uin/<subuin>`	String
- - - URI	预设用户组，请参见 ACL 概述 文档中预设用户组部分，例如 `http://cam.qcloud.com/groups/global/AllUsers` 或 `http://cam.qcloud.com/groups/global/AuthenticatedUsers`	String

对象 ACL

设置对象 ACL

功能说明

PUT Object acl 接口用来设置特定存储桶中某个对象的访问控制列表（ACL）。

> 注意：

每个主账号（即同一个 APPID），存储桶的 ACL、Policy 和 CAM 关联的策略数量总和最多为1000条，对象访问控制列表（ACL）规则数量不限制。如果您不需要进行对象访问控制列表（ACL）控制，请不要设置，默认继承存储桶权限。

使用示例

```
cos.putObjectAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'exampleobject', /* 必须 */
ACL: 'public-read', /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

为某个用户赋予对象的所有权限：

```
cos.putObjectAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'exampleobject', /* 必须 */
GrantFullControl: 'id="100000000001"' // 100000000001是主账号 uin
}, function(err, data) {
console.log(err || data);
});
```

通过 AccessControlPolicy 赋予对象写权限：

```
cos.putObjectAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'exampleobject', /* 必须 */
AccessControlPolicy: {
"Owner": { // AccessControlPolicy 里必须有 owner
"ID": 'qcs::cam::uin/100000000001:uin/100000000001' // 100000000001 是 Bucket 所属用户的 UIN 账号
},
"Grants": [{
"Grantee": {
"ID": "qcs::cam::uin/100000000011:uin/100000000011", // 100000000011 是 Bucket 所属用户下的子账号 UIN
},
"Permission": "WRITE"
}]
```

```
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String
GrantRead	赋予被授权者读取对象的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
AccessControlPolicy	设置对象的访问控制列表（ACL）属性信息	Object
- Owner	对象持有者的信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- - ID	对象持有者 ID，格式：`qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String
- - DisplayName	对象持有者的名称	String
- Grants	被授权者信息与权限信息列表	ObjectArr
- - Permission	指明授予被授权者的权限信息，枚举值：READ、WRITE、READ_ACP、WRITE_ACP、FULL_CONTROL	String
- - Grantee	被授权者的信息	Object
- - - DisplayName	被授权者的名称	String
- - - ID	被授权者的 ID，格式：`qcs::cam::uin/<owneruin>:uin/<;SubUin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，204，403，404等	Number
- headers	请求返回的头部信息	Object

查询对象 ACL

功能说明

GET Object acl 接口用来查询某个 Bucket 下的某个 Object 的访问权限。只有 Bucket 的持有者才有权限操作。

使用示例

```
cos.getObjectAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'exampleobject', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String
- Owner	标识资源的所有者	Object
- - ID	对象持有者 ID，格式为：`qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主账号 ID，<OwnerUin> 和 <SubUin>是同一个值	String
- - DisplayName	对象持有者的名称	String
- Grants	被授权者信息与权限信息列表	ObjectArray
- - Permission	指明授予被授权者的权限信息，枚举值：READ、READ_ACP、WRITE_ACP、FULL_CONTROL	String
- - Grantee	说明被授权者的信息	Object
- - - DisplayName	用户的名称	String
- - - ID	用户的 ID，格式为：`qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主账号 ID，<OwnerUin> 和 <SubUin> 是同一个值	String

防盗链

最近更新时间：2022-01-10 12:32:58

简介

本文档提供关于存储桶 Referer 白名单或者黑名单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket referer	设置存储桶 Referer	设置存储桶 Referer 白名单或者黑名单
GET Bucket referer	查询存储桶 Referer	查询存储桶 Referer 白名单或者黑名单

设置存储桶 Referer

功能说明

设置指定存储桶的 Referer 白名单或者黑名单（PUT Bucket referer）。

请求示例

```
cos.putBucketReferer({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
RefererConfiguration: {
Status: 'Enabled',
RefererType: 'White-List',
DomainList: {
Domains: [
'*.qq.com',
'*.qcloud.com',
]
},
EmptyReferConfiguration: 'Allow',
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置存储桶策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
RefererConfiguration	防盗链配置信息，详情请参见 PUT Bucket referer	Object	是
- Status	是否开启防盗链，枚举值：Enabled、Disabled	String	是
- RefererType	防盗链类型，枚举值：Black-List、White-List	String	是
- DomainList	生效域名列表， 支持多个域名且为前缀匹配， 支持带端口的域名和 IP， 支持通配符*，做二级域名或多级域名的通配	Object	是
- - Domains	生效域名,支持两种写法：单条'*.qq.com'或多条['.qq.com', '*.qcloud.com']	String\Array	是
- EmptyReferConfiguration	是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询存储桶 Referer

功能说明

查询指定存储桶 Referer 白名单或者黑名单（GET Bucket referer）。

请求示例

```
cos.getBucketReferer({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"RefererConfiguration": {
"Status": "Enabled",
"RefererType": "White-List",
"DomainList": {
"Domains": [
"*.qq.com",
"*.qcloud.com"
]
},
"EmptyReferConfiguration": "Allow"
},
"statusCode": 200,
"headers": {},
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询存储桶权限策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- RefererConfiguration	防盗链配置信息，详情请参见 GET Bucket referer	Object
- - Status	是否开启防盗链，枚举值：Enabled、Disabled	String
- - RefererType	防盗链类型，枚举值：Black-List、White-List	String
- - DomainList	生效域名列表， 支持多个域名且为前缀匹配， 支持带端口的域名和IP， 支持通配符*，做二级域名或多级域名的通配	Object
- - - Domains	生效域名	Array
- - EmptyReferConfiguration	是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny	String

数据校验
CRC64 校验

最近更新时间：2023-11-27 11:03:35

简介

数据在客户端和服务器间传输时可能会出现错误，COS 除了可以通过 MD5 和自定义属性 验证数据完整性外，还可以通过 CRC64 检验码来进行数据校验。

COS 会对新上传的对象进行 CRC64 计算，并将结果作为对象的属性进行存储，随后在返回的响应头部中携带 x-cos-hash-crc64ecma，该头部表示上传对象的 CRC64 值，根据 ECMA-182标准 计算得到。对于 CRC64 特性上线前就已经存在于 COS 的对象，COS 不会对其计算 CRC64 值，所以获取此类对象时不会返回其 CRC64 值。

操作说明

目前支持 CRC64 的 API 如下：

- 简单上传接口
  - PUT Object 和 POST Object：用户可在返回的响应头中获得文件 CRC64 校验值。
- 分块上传接口
  - Upload Part：用户可以根据 COS 返回的 CRC64 值与本地计算的数值进行比较验证。
  - Complete Multipart Upload：如果每个分块都有 CRC64 属性，则会返回整个对象的 CRC64 值，如果某些分块不具备 CRC64 值，则不返回。
- 执行 Upload Part - Copy 时，会返回对应的 CRC64 值。
- 执行 PUT Object - Copy 时，如果源对象存在 CRC64 值，则返回 CRC64，否则不返回。
- 执行 HEAD Object 和 GET Object 时，如果对象存在 CRC64，则返回。用户可以根据 COS 返回的 CRC64 值和本地计算的 CRC64 进行比较验证。

SDK 示例

功能说明

用于在对象上传和下载的时候对对象数据做 CRC64 一致性校验。

请求示例

这里只用简单上传举例，其它接口也是同样使用方式。

版权所有：腾讯云计算（北京）有限责任公司

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'exampleobject', /* 必须 */
StorageClass: 'STANDARD',
Body: fileObject, // 上传文件对象
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
if (err) {
console.log(err);
} else {
// 需要给Expose-Headers添加字段x-cos-hash-crc64ecma字段才可正确返回
// 参考文档：https://cloud.tencent.com/document/product/436/13318
var crc64 = data.headers['x-cos-hash-crc64ecma'];
console.log(crc64);
}
});
```

图片处理
基础图片处理

最近更新时间：2023-02-07 15:03:05

简介

本文档提供关于基础图片处理的相关的 API 概览以及 SDK 示例代码。

服务	功能	说明
基础图片处理服务	缩放	等比缩放、设定目标宽高缩放等多种方式
	裁剪	普通裁剪、缩放裁剪、内切圆、人脸智能裁剪
	旋转	自适应旋转、普通旋转
	格式转换	格式转换、GIF 格式优化、渐进显示
	质量变换	针对 JPG 和 WEBP 图片进行质量变换
	高斯模糊	对图片进行模糊处理
	锐化	对图片进行锐化处理
	添加水印	图片水印、文字水印
	获取图片信息	基本信息、EXIF 信息、主色调
	去除元信息	包括 EXIF 信息
	快速缩略模板	快速实现图片格式转换、缩略、剪裁等功能，生成缩略图

水印的文字内容需要经过 URL 安全的 Base64 编码，可引入 `js-base64` 来解决。

示例代码如下：

```
1. 项目内安装依赖
npm install --save js-base64
2. 代码里引入 encode 方法
import { encode } from 'js-base64';
// 文字水印内容
var text = '腾讯云万象优图';
// 有效的文字水印
```

```
var safeBase64 = encode(text).replaceAll('+', '-').replaceAll('/', '_').replaceAl
l('=', '');
// 可使用如下Pic-Operations
'Pic-Operations': `{"is_pic_info": 1, "rules": [{"fileid": "waterMask.jpg", "rul
e": "watermark/2/text/${safeBase64}"}]}`
```

上传时使用图片处理

下面示例展示了如何在上传图片时自动实现图片处理。

图片上传完成后，COS 会存储原始图片和已处理过的图片。后续用户可以通过普通的下载请求获取处理结果。

示例代码

```
<!-- html 页面 DOM 元素 -->
<!-- 选择要上传的文件 -->
<input id="fileSelector" type="file" />
<!-- 点击按钮上传 -->
<input id="submitBtn" type="submit" />
```

```
function handleFileInUploading(file) {
cos.putObject(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: file.name,
Body: file,
Headers: {
// 通过 imageMogr2 接口使用图片缩放功能：指定图片宽度为 200，宽度等比压缩
'Pic-Operations':
'{"is_pic_info": 1, "rules": [{"fileid": "desample_photo.jpg", "rule": "imageMogr
2/thumbnail/200x/"}]}',
},
},
function (err, data) {
console.log(err || data);
},
);
}
document.getElementById('submitBtn').onclick = function (e) {
var file = document.getElementById('fileSelector').files[0];
if (!file) {
document.getElementById('msg').innerText = '未选择上传文件';
return;
```

```
}
handleFileInUploading(file);
};
```

对云上数据进行图片处理

下面示例展示了如何在对已存储在 COS 的图片进行相应处理操作，并将结果存入到 COS。

示例代码

```
<!-- html 页面 DOM 元素 -->
<!-- 点击按钮请求对云上数据进行图片处理 -->
<input id="submitBtn" type="submit" />
```

```
function handleFileInBucket() {
cos.request(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleImage',
Method: 'POST',
Action: 'image_process',
Headers: {
// 通过 imageMogr2 接口使用图片缩放功能：指定图片宽度为 200，宽度等比压缩
'Pic-Operations':
'{"is_pic_info": 1, "rules": [{"fileid": "desample_photo.jpg", "rule": "imageMogr
2/thumbnail/200x/"}]}',
},
},
function (err, data) {
console.log(err || data);
},
);
}
document.getElementById('submitBtn').onclick = function (e) {
handleFileInBucket();
};
```

下载时使用图片处理

下面示例展示了如何在下载图片时实现图片处理。

示例代码

```
<!-- html 页面 DOM 元素 -->
<!-- 点击按钮下载文件并在下载时使用图片处理 -->
<input id="submitBtn" type="submit" />
```

```
function getObject() {
cos.getObject(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleobject',
QueryString: `imageMogr2/thumbnail/200x/`,
},
function (err, data) {
console.log(err || data);
},
);
}
```

生成带图片处理参数的签名 URL

示例代码

```
// 生成带图片处理参数的文件签名 URL，过期时间设置为 30 分钟。
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleobject',
Query: {
`imageMogr2/thumbnail/200x/`: ''
},
Expires: 1800,
Sign: true,
},
function (err, data) {
if (data) {
// 使用浏览器打开url预览或触发浏览器下载
console.log(data.URL);
}
},
);
```

```
// 生成带图片处理参数的文件 URL，不带签名。
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleobject',
Query: {
`imageMogr2/thumbnail/200x/`: ''
},
Sign: false,
},
function (err, data) {
if (data) {
// 使用浏览器打开url预览或触发浏览器下载
console.log(data.URL);
}
},
);
```

盲水印

最近更新时间：2022-10-09 16:16:28

简介

本文档提供关于盲水印的相关的 API 概览以及 SDK 示例代码。

关于盲水印 API 文档请参见 盲水印。

添加盲水印

功能说明

盲水印支持在上传时添加以及下载时添加。

示例代码一：上传时添加盲水印

```
<!-- html 页面 DOM 元素 -->
<!-- 选择要上传的文件 -->
<input id="fileSelector" type="file" />
<!-- 点击按钮上传 -->
<input id="submitBtn" type="submit" />
```

```
function handleFileInUploading(file) {
cos.putObject(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: file.name,
Body: file,
Headers: {
// 万象持久化接口，上传时持久化
'Pic-Operations':
'{"is_pic_info": 1, "rules": [{"fileid": "desample_photo.jpg", "rule": "watermar
k/3/type/1/image/aHR0cDovL2V4YW1wbGVzLTEyNTEwMDAwMDQucGljc2gubXlxY2xvdWQuY29tL3No
dWl5aW4uanBn"}]}',
},
},
function (err, data) {
console.log(err || data);
},
);
```

```
}
document.getElementById('submitBtn').onclick = function (e) {
var file = document.getElementById('fileSelector').files[0];
if (!file) {
document.getElementById('msg').innerText = '未选择上传文件';
return;
}
handleFileInUploading();
};
```

示例代码二：下载时添加盲水印

```
<!-- html 页面 DOM 元素 -->
<!-- 点击按钮下载文件并在下载时使用图片处理 -->
<input id="submitBtn" type="submit" />
```

```
function getObject() {
cos.getObject(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleobject',
QueryString: `watermark/3/type/1/image/aHR0cDovL2V4YW1wbGVzLTEyNTEwMDAwMDQucGljc2
gubXlxY2xvdWQuY29tL3NodWl5aW4uanBn`,
},
function (err, data) {
console.log(err || data);
},
);
}
```

内容审核
图片审核

最近更新时间：2022-12-14 15:23:11

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于图片审核的 API 概览和 SDK 示例代码。

> 注意：
> COS Javascript SDK 版本需要大于等于 v1.3.1。

API	操作描述
图片同步审核	对对象存储（Cloud Object Storage，COS）存量数据进行涉黄、违法违规以及广告引导类图片的扫描
图片批量审核	对多个图片进行批量审核

图片同步审核

功能说明

图片审核的存量扫描功能通过借助数据万象的持久化处理接口，实现对 COS 存量数据的涉黄、违法违规以及广告引导类图片的扫描。

使用示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function getImageAuditing() {
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'GET',
Key: '1.png', /* 存储桶内的图片文件，必须字段 */
Query: {
'ci-process': 'sensitive-content-recognition', /** 固定值，必须 */
'biz-type': '', /** 审核类型，非必须 */
'detect-type': 'porn,ads', /** 审核策略，不填写则使用默认策略，非必须 */
'detect-url': '', /** 审核任意公网可访问的图片链接，非必须 */
'interval': 5, /** 审核 GIF 动图时，每隔interval帧截取一帧，非必须 */
'max-frames': 5, /** 审核 GIF 动图时，最大截帧数，非必须 */
'large-image-detect': '0', /** 是否需要压缩图片后再审核，非必须 */
'dataid': '123', /** 自定义图片标识，非必须 */
},
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

Query 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
biz-type	Query	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置公共审核策略。	String	否
detect-type	Query	审核类型审核类型，porn（涉黄识别）、ads（广告识别）。可选择多种识别类型用 `,` 连接。	String	否
object-key	Query	图片文件在 bucket 中的位置。	String	是

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
detect-url	Query	您可以通过填写 `detect-url` 审核任意公网可访问的图片链接。 • 不填写 `detect-url` 时，后台会默认审核 `ObjectKey` • 填写 `detect-url` 时，后台会审核 `detect-url` 链接，无需再填写 `ObjectKey` `detect-url` 示例：http://www.example.com/abc.jpg	String	否
interval	Query	审核 GIF 动图时，可使用该参数进行截帧配置，代表截帧的间隔。例如值设为5，则表示从第1帧开始截取，每隔5帧截取一帧，默认值5	Int	否
max-frames	Query	针对 GIF 动图审核的最大截帧数量，需大于0。例如值设为5，则表示最大截取5帧，默认值为5。	Int	否
large-image-detect	Query	对于超过大小限制的图片是否进行压缩后再审核，取值为：0（不压缩），1（压缩）。默认为0。 注：压缩最大支持32M的图片，且会收取压缩费用。	String	否
dataid	Query	图片标识，该字段在结果中返回原始内容，长度限制为512字节。	String	否

返回结果说明

详情请参见 图片同步审核。

图片批量审核

功能说明

图片批量审核接口为同步请求方式，您可以通过本接口对多个图片文件进行内容审核。该接口属于 POST 请求。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function postImagesAuditing() {
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
```

```
var url = 'https://' + host + '/image/auditing';
var body = COS.util.json2xml({
Request: {
Input: [{
Object: '1.png', /* 需要审核的图片，存储桶里的路径 */
}, {
Object: 'a/6.png', /* 需要审核的图片，存储桶里的路径 */
}],
Conf: {
BizType: '', /* 不填写代表默认策略 */
DetectType: 'Porn' /* 只审核是否涉黄 */
}
}
});
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'POST',
Url: url,
Key: '/image/auditing', /** 固定值，必须 */
ContentType: 'application/xml', /** 固定值，必须 */
Body: body
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

request的数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Request	无	图片批量审核的具体配置项。	Container	是

Container 类型 Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的内容，如有多个图片，请传入多个 Input 结构。	Container Array	是
Conf	Request	审核规则配置。	Container	是

版权所有：腾讯云计算（北京）有限责任公司

Container 类型 Input 的具体数据描述如下，使用其中一种：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	存储在 COS 存储桶中的图片文件名称，例如在目录 test 中的文件 image.jpg，则文件名称为 test/image.jpg。Object 和 Url 只能选择其中一种。	String	否
Url	Request.Input	图片文件的链接地址，例如 http://a-1250000.cos.ap-shanghai.myqcloud.com/image.jpg。Object 和 Url 只能选择其中一种。	String	否
Interval	Request.Input	截帧频率，GIF 图检测专用，默认值为5，表示从第一帧（包含）开始每隔5帧截取一帧	Int	否
MaxFrames	Request.Input	最大截帧数量，GIF 图检测专用，默认值为5，表示只截取 GIF 的5帧图片进行审核，必须大于0	Int	否
DataId	Request.Input	图片标识，该字段在结果中返回原始内容，长度限制为512字节	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以 `,` 分隔，例如：Porn,Ads。	String	是
BizType	Request.Conf	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置公共审核策略。	String	否

注意：

- 通过 Object 进行审核为内网操作，不会产生额外的外网流量。
- 通过 Url 进行审核，会产生图片所在源站对应的外网流量。

返回结果说明

详情请参见 图片批量审核。

版权所有：腾讯云计算（北京）有限责任公司

视频审核

最近更新时间：2022-12-14 15:23:11

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于视频审核的 API 概览和 SDK 示例代码。

> 注意：
> COS Javascript SDK 版本需要大于等于 v1.3.1。

API	操作描述
提交视频审核任务	用于提交一个视频审核任务
查询视频审核任务结果	用于查询指定的视频审核任务

提交视频审核任务

功能说明

用于提交一个视频审核任务。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
```

```
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function postVideoAuditing() {
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/video/auditing';
var body = COS.util.json2xml({
Request: {
Input: {
Object: '1.mp4', /* 需要审核的视频文件，存储桶里的路径 */
},
Conf: {
BizType: '',
DetectType: 'Porn',
Snapshot: {
Count: 1000, // 视频截帧数量
},
DetectContent: 1, // 是否审核视频声音，0-只审核视频不审核声音；1-审核视频+声音
}
}
});
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'POST',
Url: url,
Key: '/video/auditing', /** 固定值，必须 */
ContentType: 'application/xml', /** 固定值，必须 */
Body: body
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的视频。	Container	是
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下：

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	当前 COS 存储桶中的视频文件名称，例如在目录 test 中的文件 video.mp4，则文件名称为 test/video.mp4。	String	是
Url	Request.Input	视频文件的链接地址，例如 http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/test.mp4。Object 和 Url 只能选择其中一种。	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。	String	否
Snapshot	Request.Conf	视频画面的审核通过视频截帧能力截取出一定量的截图，通过对截图逐一审核而实现的，该参数用于指定视频截帧的配置。	Container	是
Callback	Request.Conf	回调地址，以 `http://` 或者 `https://` 开头的地址。	String	否
CallbackVersion	Request.Conf	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。	String	否
BizType	Request.Conf	审核策略，不带审核策略时使用默认策略。	String	否
DetectContent	Request.Conf	用于指定是否审核视频声音，当值为0时：表示只审核视频画面截图；值为1时：表示同时审核视频画面截图和视频声音。默认值为0。	Integer	否

Container 类型 Snapshot 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Mode	Request.Conf.Snapshot	截帧模式。Interval 表示间隔模式；Average 表示平均模式；Fps 表示固定帧率模式。 • Interval 模式：TimeInterval，Count 参数生效。当设置 Count，未设置 TimeInterval 时，表示截取所有帧，共 Count 张图片。 • Average 模式：Count 参数生效。表示整个视频，按平均间隔截取共 Count 张图片。 • Fps 模式：TimeInterval 表示每秒截取多少帧，Count 表示共截取多少帧。	String	否
Count	Request.Conf.Snapshot	视频截帧数量，范围为(0, 10000]。	String	否
TimeInterval	Request.Conf.Snapshot	视频截帧频率，范围为(0, 60]，单位为秒，支持 float 格式，执行精度精确到毫秒。	Float	否

返回结果说明

详情请参见 提交视频审核任务。

查询视频审核任务结果

功能说明

用于查询视频审核任务执行情况和结果。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function getVideoAuditingResult() {
var jobId = 'av14d9ca15af3a11eca0d6525400d88xxx'; // jobId可以通过提交视频审核任务返回
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/video/auditing/' + jobId;
cos.request({
```

```
Bucket: config.Bucket,
Region: config.Region,
Method: 'GET',
Key: '/video/auditing/' + jobId,
Url: url,
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID	String	是

返回结果说明

详情请参见 查询视频审核任务结果。

音频审核

最近更新时间：2022-12-14 15:23:11

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于音频审核的 API 概览和 SDK 示例代码。

> 注意：
> COS Javascript SDK 版本需要大于等于 v1.3.1。

API	操作描述
提交音频审核任务	用于提交一个音频审核任务
查询音频审核任务结果	用于查询指定的音频审核任务

提交音频审核任务

功能说明

用于提交一个音频审核任务。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
```

```
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function postAudioAuditing() {
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/audio/auditing';
var body = COS.util.json2xml({
Request: {
Input: {
Object: '1.mp3', /* 需要审核的音频文件，存储桶里的路径 */
},
Conf: {
BizType: '',
DetectType: 'Porn',
}
}
});
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'POST',
Url: url,
Key: '/audio/auditing', /** 固定值，必须 */
ContentType: 'application/xml', /** 固定值，必须 */
Body: body
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的内容。	Container	是
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	存储在 COS 存储桶中的音频文件名称，例如在目录 test 中的文件 audio.mp3，则文件名称为 test/audio.mp3。Object 和 Url 只能选择其中一种。	String	否
Url	Request.Input	音频文件的链接地址，例如 http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/audio.mp3。Object 和 Url 只能选择其中一种。	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告）、Illegal（违法）、Abuse（谩骂），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。	String	否
Callback	Request.Conf	审核结果以回调形式发送至您的回调地址，支持以 `http://` 或者 `https://` 开头的地址，例如：`http://www.callback.com` 。	String	否
CallbackVersion	Request.Conf	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。	string	否
BizType	Request.Conf	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置公共审核策略。	String	否

返回结果说明

详情请参见 提交音频审核任务。

查询音频审核任务

功能说明

用于查询音频审核任务执行情况和结果。

版权所有：腾讯云计算（北京）有限责任公司

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function getAudioAuditingResult() {
var jobId = 'sa0c28d41daff411ecb23352540078cxxx'; // jobId可以通过提交音频审核任务返回
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/audio/auditing/' + jobId;
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'GET',
Key: '/audio/auditing/' + jobId,
Url: url,
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID	String	是

返回结果说明

详情请参见 查询音频审核任务结果。

文本审核

最近更新时间：2022-12-14 15:23:11

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于文本审核的 API 概览和 SDK 示例代码。

> 注意：
> COS Javascript SDK 版本需要大于等于 v1.3.1。

API	操作描述
提交文本审核任务	用于提交一个文本审核任务
查询文本审核任务结果	用于查询指定的文本审核任务

提交文本审核任务

功能说明

用于提交一个文本审核任务。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
```

```
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function postTextAuditing() {
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/text/auditing';
var body = COS.util.json2xml({
Request: {
Input: {
Object: 'hello.txt', /* 需要审核的文本文件，存储桶里的路径 */
},
Conf: {
BizType: '',
DetectType: 'Porn',
}
}
});
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'POST',
Url: url,
Key: '/text/auditing', /** 固定值，必须 */
ContentType: 'application/xml', /** 固定值，必须 */
Body: body
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Request	无	文本审核的具体配置项。	Container	是

Container 类型 Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的内容。	Container	是
Conf	Request	审核规则配置。	Container	是

版权所有：腾讯云计算（北京）有限责任公司

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	当前 COS 存储桶中的文本文件名称，例如在目录 test 中的文件 img.jpg，则文件名称为 test/img.jpg，文本文件仅支持UTF8编码和 GBK 编码的内容，且文件大小不得超过1MB。	String	否
Content	Request.Input	当传入的内容为纯文本信息，需要先经过 base64 编码，文本编码前的原文长度不能超过10000个 utf8 编码字符。若超出长度限制，接口将会报错。	String	否

注意：

- 不可同时输入 Object 和 Content。
- 当选择 Object 时，审核结果为异步返回，可通过 查询文本审核任务结果 API 接口获取返回结果。
- 当选择 Content 时，审核结果为同步返回，可通过查看响应体 TextAuditingResponse 获取返回结果。
- 目前仅支持中文、英文、阿拉伯数字的检测与审核。

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告）、Illegal（违法）、Abuse（谩骂），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。	String	否
Callback	Request.Conf	审核结果可以回调形式发送至您的回调地址，支持以 `http://` 或者 `https://` 开头的地址，例如： `http://www.callback.com` 。	String	否
BizType	Request.Conf	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置公共审核策略。	String	否

返回结果说明

详情请参见 提交文本审核任务。

版权所有：腾讯云计算（北京）有限责任公司

查询文本审核任务

功能说明

用于查询文本审核任务执行情况和结果。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function getTextAuditingResult() {
var jobId = 'st8d88c664aff511ecb23352540078cxxx'; // jobId可以通过提交文本审核任务返回（Input传入Object）
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/text/auditing/' + jobId;
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'GET',
Key: '/text/auditing/' + jobId,
Url: url,
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID	String	是

返回结果说明

详情请参见 查询文本审核任务结果。

网页审核

最近更新时间：2022-12-14 15:23:11

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于网页审核的 API 概览和 SDK 示例代码。

> 注意：
> COS Javascript SDK 版本需要大于等于 v1.3.1。

API	操作描述
提交网页审核任务	用于提交一个网页审核任务
查询网页审核任务结果	用于查询指定的网页审核任务

提交网页审核任务

功能说明

用于提交一个网页审核任务。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
```

```
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function postWebpageAuditing() {
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/webpage/auditing';
var body = COS.util.json2xml({
Request: {
Input: {
Url: 'https://cloud.tencent.com/', // 存在cos里的资源，审核结果异步返回，可以调用查询文本审核结果api查询
},
Conf: {
BizType: '',
DetectType: 'Porn,Ads',
}
}
});
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'POST',
Url: url,
Key: '/webpage/auditing', /** 固定值，必须 */
ContentType: 'application/xml', /** 固定值，必须 */
Body: body
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Request	无	网页审核的具体配置项。	Container	是

Container 类型 Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的网页信息。	Container	是

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Url	Request.Input	网页文件的链接地址，例如 `http://www.test.com` 。	String	是
DataId	Request.Input	该字段在审核结果中会返回原始内容，长度限制为512字节。您可以使用该字段对待审核的数据进行唯一业务标识。	String	否
UserInfo	Request.Input	用户业务字段。	Container	否

Container 节点 UserInfo 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
TokenId	Request.Input.UserInfo	用户业务 TokenId，长度不超过128字节。	String	否
Nickname	Request.Input.UserInfo	用户业务 Nickname，长度不超过128字节。	String	否
DeviceId	Request.Input.UserInfo	用户业务 DeviceId，长度不超过128字节。	String	否
AppId	Request.Input.UserInfo	用户业务 AppId，长度不超过128字节。	String	否
Room	Request.Input.UserInfo	用户业务 Room，长度不超过128字节。	String	否
IP	Request.Input.UserInfo	用户业务 IP，长度不超过128字节。	String	否
Type	Request.Input.UserInfo	用户业务 Type，长度不超过128字节。	String	否

Container 类型 Conf 的具体数据描述如下：

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。	String	是
Callback	Request.Conf	回调地址，以 `http://` 或者 `https://` 开头的地址。	String	否
ReturnHighlightHtml	Request.Conf	指定是否需要高亮展示网页内的违规文本，查询及回调结果时会根据此参数决定是否返回高亮展示的 html 内容。取值为 true 或者 false，默认为 false。	Boolean	否

返回结果说明

详情请参见 提交网页审核任务。

查询网页审核任务结果

功能说明

用于查询网页审核任务执行情况和结果。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function getWebpageAuditingResult() {
var jobId = 'shce868019aff611ecb1155254009a4xxx'; // jobId可以通过提交网页审核任务返回
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/webpage/auditing/' + jobId;
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'GET',
Key: '/webpage/auditing/' + jobId,
Url: url,
},
function(err, data){
```

```
console.log(err || data);
});
}
```

参数说明

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID	String	是

返回结果说明

详情请参见 查询网页审核任务结果。

设置访问域名（CDN/全球加速）

最近更新时间：2022-01-06 16:16:08

简介

本文档提供关于如何使用非默认域名请求 COS 服务。

相关参数说明

通过初始化参数，来控制请求域名，相关的参数说明如下，更多初始化参数说明请查看 配置项。

参数名	参数描述	类型	是否必填
Domain	调用操作存储桶和对象的 API 时自定义请求域名。可以使用模版，如 `"{Bucket}.cos.{Region}.myqcloud.com"`，即在调用 API 时会使用参数中传入的 Bucket 和 Region 进行替换。	String	否
Protocol	发请求时用的协议，可选项 `https:`、`http:`，默认判断当前页面是 `http:` 时使用 `http:`，否则使用 `https:`	String	否

默认 CDN 加速域名

关于如何开启默认加速域名请参考 开启默认 CDN 加速域名。

以下代码展示了如何使用默认加速域名访问 COS 服务。

```
var cos = new COS({
Domain: '{Bucket}.file.myqcloud.com', // 自定义加速域名，Domain 参数支持模板，这个例子里 {Bucket} 会在请求时按传入的 Bucket 自动替换
Protocol: 'https:', // 请求协议： 'https:' 或 'http:'
});
```

自定义 CDN 加速域名

关于如何开启 CDN 自定义加速域名请参考 开启自定义 CDN 加速域名。

以下代码展示了如何使用自定义加速域名访问 COS 服务。

```
var cos = new COS({
Domain: 'example-cdn-domain.com', // 自定义加速域名
Protocol: 'https:', // 请求协议： 'https:' 或 'http:'
});
```

版权所有：腾讯云计算（北京）有限责任公司

自定义源站域名

关于如何设置自定义源站域名请参考 自定义源站域名。

以下代码展示了如何使用自定义源站域名访问 COS 服务。

```
var cos = new COS({
Domain: 'example-cos-domain.com', // 自定义源站域名
Protocol: 'https:', // 请求协议： 'https:' 或 'http:'
});
```

全球加速域名

关于全球加速功能请参考 全球加速功能概述。

以下代码展示了如何使用全球加速域名访问 COS 服务。

```
var cos = new COS({
UseAccelerate: true, // 指定 true，使用全球加速域名请求
Protocol: 'https:', // 请求协议： 'https:' 或 'http:'
});
```

版权所有：腾讯云计算（北京）有限责任公司

异常处理

最近更新时间：2022-01-05 14:31:57

简介

调用 SDK 接口请求 COS 服务失败时，将在回调函数里返回错误信息。

错误处理示例

```
cos.headBucket({
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
if (err) {
console.log(err.error);
}
});
```

客户端异常

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- error	请求报错信息	Object/String

服务端异常

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- error	请求报错信息	Object/String
- - Code	请求失败时 body 返回的 Error Code，详情请参见 错误码 文档	String
- - Message	请求失败时 body 返回的 Error Message，详情请参见 错误码 文档	String
- - RequestId	服务器请求日志里的唯一 ID，可用于遇到问题 联系我们 排查错误原因	String

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。
   COS 自助诊断工具
   点击自助诊断
   输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**。
4. 稍侯片刻，便能看到相应的智能诊断结果。

Node.js SDK
快速入门

最近更新时间：2023-03-09 17:10:43

下载与安装

相关资源

- 对象存储服务的 XML JS SDK 资源 github 地址：XML Node.js SDK。
- SDK 快速下载地址：XML Node.js SDK。
- 演示示例 Demo 下载地址：XML Node.js SDK Demo。
- SDK 文档中的所有示例代码请参见 SDK 代码示例。
- SDK 更新日志请参见 ChangeLog。

> 说明：
> 如果您在使用 SDK 时遇到函数或方法不存在等错误，请先将 SDK 升级到最新版再重试。

环境依赖

1. 使用 SDK 需要您的运行环境包含 nodejs 以及 npm，其中 nodejs 版本要求 ≥ 6。
2. 登录 对象存储控制台 创建存储桶后，获取存储桶名称和 地域名称。
3. 登录 访问管理控制台 获取您的项目 SecretId 和 SecretKey。

> 说明：
> 关于本文中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参见 COS 术语信息。

安装 SDK

通过 npm 安装环境 SDK： npm 地址。

```
npm i cos-nodejs-sdk-v5 --save
```

开始使用

初始化

注意：

- 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

使用临时密钥初始化(推荐)

临时密钥生成和使用请参见 临时密钥生成及使用指引。Node.js SDK 支持通过传入临时密钥进行初始化，请参考以下示例代码：

```
var request = require('request');
var COS = require('cos-nodejs-sdk-v5');
var cos = new COS({
getAuthorization: function (options, callback) {
// 初始化时不会调用，只有调用 cos 方法（例如 cos.putObject）时才会进入
// 异步获取临时密钥
request({
url: 'https://example.com/sts',
data: {
// 可从 options 取需要的参数
}
}, function (err, response, body) {
try {
var data = JSON.parse(body);
var credentials = data.credentials;
} catch(e) {}
if (!data || !credentials) return console.error('credentials invalid');
callback({
TmpSecretId: credentials.tmpSecretId, // 临时密钥的 tmpSecretId
TmpSecretKey: credentials.tmpSecretKey, // 临时密钥的 tmpSecretKey
SecurityToken: credentials.sessionToken, // 临时密钥的 sessionToken
ExpiredTime: data.expiredTime, // 临时密钥失效时间戳，是申请临时密钥时，时间戳加 duratio
nSeconds
});
});
}
});
```

使用永久密钥初始化(不推荐)

使用永久密钥初始化时，请注意保管好密钥，防止泄露。

请先在访问管理控制台中的 API 密钥管理 页面获取 SecretId、SecretKey。

将 SecretId、SecretKey、Bucket 和 Region 修改为您实际开发环境下的值，测试上传文件，请参考以下示例代码：

```
// SECRETID 和 SECRETKEY 请登录 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
var COS = require('cos-nodejs-sdk-v5');
var cos = new COS({
SecretId: process.env.SecretId, // 推荐使用环境变量获取；用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考https://cloud.tencent.com/document/product/598/37140
SecretKey: process.env.SecretKey, // 推荐使用环境变量获取；用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考https://cloud.tencent.com/document/product/598/37140
});
```

以下是部分常用接口例子，更详细的初始化方法请参见 demo 示例。

配置项

构造函数参数说明

参数名	参数描述	类型	是否必填
SecretId	用户的 SecretId	String	是
SecretKey	用户的 SecretKey	String	是
FileParallelLimit	同一个实例下上传的文件并发数，默认值3	Number	否
ChunkParallelLimit	同一个上传文件的分块并发数，默认值3	Number	否
ChunkRetryTimes	分块上传及分块复制时，出错重试次数，默认值2（加第一次，请求共3次）	Number	否
ChunkSize	分块上传时，每块的字节数大小，单位为 Byte，默认值1048576（1MB）	Number	否
SliceSize	使用 uploadFiles 批量上传时，文件大小大于该数值将使用按分片上传，否则将调用简单上传，单位 Byte，默认值1048576（1MB）	Number	否
CopyChunkParallelLimit	进行分块复制操作中复制分块上传的并发数，默认值20	Number	否
CopyChunkSize	使用 sliceCopyFile 分块复制文件时，每片的大小字节数，单位为 Byte，默认值10485760（10MB）	Number	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
CopySliceSize	使用 sliceCopyFile 分片复制文件时，文件大小大于该数值将使用分片复制，否则将调用简单复制，单位为 Byte，默认值10485760（10MB）	Number	否
ProgressInterval	上传进度的回调方法 onProgress 的回调频率，单位 ms，默认值1000	Number	否
Protocol	发请求时用的协议，可选项 `https:` 、 `http:` ，默认判断当前页面是 `http:` 时使用 `http:` ，否则使用 `https:`	String	否
ServiceDomain	调用 getService 方法时，请求的域名，例如 `service.cos.myqcloud.com`	String	否
Domain	调用操作存储桶和对象的 API 时自定义请求域名。可以使用模板， 例如 `"{Bucket}.cos.{Region}.myqcloud.com"` ，即在调用 API 时会使用参数中传入的 Bucket 和 Region 进行替换	String	否
UploadQueueSize	上传队列最长大小，超出队列大小并失败/已完成/已取消状态的任务会被清理，默认1000	Number	否
ForcePathStyle	强制使用后缀式模式发请求。后缀式模式中 Bucket 会放在域名后的 pathname 里，并且 Bucket 会加入签名 pathname 计算，默认 false	Boolean	否
UploadCheckContentMd5	强制上传文件也校验 Content-MD5，会对文件请求 Body 计算 md5 放在 header 的 Content-MD5 字段里，默认 false	Boolean	否
Timeout	超时时间，单位毫秒，默认为0，即不设置超时时间	Number	否
KeepAlive	多个请求同用 TCP 连接，默认 true，若请求并发量大建议 打开	Boolean	否
StrictSsl	严格校验 HTTPS 证书，默认 true	Boolean	否
Proxy	请求时使用 HTTP 代理，例如： `http://127.0.0.1:8080`	String	否
getAuthorization	获取签名的回调方法，如果没有 SecretId、SecretKey 时，这个参数必选	Function	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
UseAccelerate	是否启用全球加速域名，默认为 false。若改为 true，需要存储桶开启全球加速功能，详情请参见 开启全球加速。	Boolean	否

getAuthorization 回调函数说明的函数说明（使用格式一）

```
getAuthorization: function(options, callback) { ... }
```

getAuthorization 的函数说明：

参数名	参数描述	类型
options	获取临时密钥需要的参数对象	Object
- Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
- Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
callback	临时密钥获取完成后的回传方法	Function

获取完临时密钥后，callback 回传一个对象，回传对象的属性列表如下：

属性名	参数描述	类型	是否必填
TmpSecretId	获取回来的临时密钥的 tmpSecretId	String	是
TmpSecretKey	获取回来的临时密钥的 tmpSecretKey	String	是
SecurityToken	获取回来的临时密钥的 sessionToken，对应 header 的 x-cos-security-token 字段	String	是
StartTime	密钥获取的开始时间，即获取时刻的时间戳，单位秒，startTime，如：1580000000，用于签名开始时间，传入该参数可避免前端时间偏差签名过期问题	String	否
ExpiredTime	获取回来的临时密钥的 expiredTime，超时时刻的时间戳，单位秒，如：1580000900	String	是

getAuthorization 回调函数说明（使用格式二）

```
getAuthorization: function(options, callback) { ... }
```

getAuthorization 的函数说明：

参数名	参数描述	类型
options	获取签名需要的参数对象	Object
- Method	当前请求的 Method	String
- Pathname	请求路径，用于签名计算	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，了解更多请参见 对象概述	String
- Query	当前请求的 query 参数对象，{key: 'val'} 的格式	Object
- Headers	当前请求的 header 参数对象，{key: 'val'} 的格式	Object
callback	临时密钥获取完成后的回调	Function

getAuthorization 计算完成后，callback 回传参数支持两种格式：
格式一：回传鉴权凭证字符串 Authorization。
格式二：回传一个对象，对象属性列表如下：

属性名	参数描述	类型	是否必填
Authorization	计算得到的签名字符串	String	是
SecurityToken	获取回来的临时密钥的 sessionToken，对应 header 的 x-cos-security-token 字段	String	否

获取鉴权凭证

实例本身鉴权凭证可以通过实例化时传入的参数控制如何或获取，有三种获取方式：

1. 实例化时，传入 SecretId、SecretKey，每次需要签名都由实例内部计算。
2. 实例化时，传入 getAuthorization 回调，每次需要签名通过这个回调计算完返回签名给实例。
3. 实例化时，传入 getSTS 回调，每次需要临时密钥通过这个回调回去完返回给实例，在每次请求时实例内部使用临时密钥计算得到签名。

使用方式

回调方式

文档里默认使用回调方式，使用代码如下:

```
// 这里省略初始化过程和上传参数
var cos = new COS({ ... });
cos.uploadFile({ ... }, function(err, data) {
if (err) {
console.log('上传出错', err);
} else {
console.log('上传成功', data);
}
});
```

Promise

sdk同样支持 Promise 方式调用，例如上述回调方式的代码等同于以下代码:

```
// 这里省略初始化过程和上传参数
var cos = new COS({ ... });
cos.uploadFile({ ... }).then(data => {
console.log('上传成功', data);
}).catch(err => {
console.log('上传出错', err);
});
```

同步方式

同步方式基于 JavaScript 的 async 和 await，上述回调方式的代码等同于以下代码:

```
async function upload() {
// 这里省略初始化过程和上传参数
var cos = new COS({ ... });
try {
var data = await cos.uploadFile({ ... });
return { err: null, data: data }
} catch (err) {
return { err: err, data: null };
}
}
// 可以同步拿到请求的返回值,这里举例说明,实际返回的数据格式可以自定义
var uploadResult = await upload();
if (uploadResult.err) {
console.log('上传出错', uploadResult.err);
} else {
console.log('上传成功', uploadResult.data);
}
```

版权所有：腾讯云计算（北京）有限责任公司

注意：
cos.getObjectUrl 目前只支持回调方式。

使用技巧

通常情况下我们只需要创建一个 COS SDK 实例，然后在需要调用SDK方法的地方直接使用这个实例即可，示例代码如下：

```
var cos = new COS({
....
});

/* 自己封装的上传方法 */
function myUpload() {
// 不需要在每个方法里创建一个 COS SDK 实例
// var cos = new COS({
// ...
// });
cos.putObject({
....
});
}

/* 自己封装的删除方法 */
function myDelete() {
// 不需要在每个方法里创建一个 COS SDK 实例
// var cos = new COS({
// ...
// });
cos.deleteObject({
....
});
}
```

以下是部分常用接口入口，更详细的初始化方法请参见 demo 示例。

创建存储桶

```
cos.putBucket({
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION'
}, function(err, data) {
console.log(err || data);
});
```

查询存储桶列表

```
cos.getService(function (err, data) {
console.log(data && data.Buckets);
});
```

上传对象

该接口适用于小文件上传，大文件请使用分块上传接口，详情请参见 对象操作 文档。

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: 'exampleobject', /* 必须 */
StorageClass: 'STANDARD',
Body: fs.createReadStream('./exampleobject'), // 上传文件对象
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

查询对象列表

```
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Prefix: 'a/', /* 非必须 */
}, function(err, data) {
console.log(err || data.Contents);
});
```

下载对象

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: 'exampleobject', /* 必须 */
Output: fs.createWriteStream('./exampleobject'),
}, function(err, data) {
console.log(err || data);
});
```

版权所有：腾讯云计算（北京）有限责任公司

删除对象

```
cos.deleteObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: 'exampleobject' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

**

存储桶操作

最近更新时间：2022-07-07 14:22:07

简介

本文档提供关于存储桶基本操作的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Service（List Buckets）	查询存储桶列表	查询指定账号下所有的存储桶列表
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

查询存储桶列表

功能说明

GET Service 接口是用来查询请求者名下的所有存储桶列表或特定地域下的存储桶列表（Bucket list），详情请参见 GET Service。

使用示例

示例一：列出存储桶列表。

```
cos.getService(function(err, data) {
console.log(err || data);
});
```

示例二：列出指定地域的存储桶列表。

```
cos.getService({
Region: 'COS_REGION',
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Owner	代表存储桶所有者的对象	Object
- - ID	存储桶持有者的完整 ID，格式为 `qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]`，例如 `qcs::cam::uin/100000000001:uin/100000000001`，其中100000000001为 uin	String
- - DisplayName	存储桶持有者的名称	String
- Buckets	存储桶信息列表	Object
- - Name	存储桶的名称，格式为，例如 examplebucket-1250000000	String
- - Location	存储桶所在地域。枚举值请参见 地域和访问域名 文档，例如 ap-guangzhou，ap-beijing，ap-hongkong 等	String
- - CreationDate	存储桶创建时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	String

创建存储桶

版权所有：腾讯云计算（北京）有限责任公司

功能说明

PUT Bucket 接口请求可以在指定账号下创建一个存储桶。

使用示例

```
cos.putBucket({
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION'
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
ACL	定义存储桶的访问控制列表（ACL）属性。枚举值请参见 ACL 概述 文档中存储桶的预设 ACL 部分，例如 private，public-read 等，默认为 private	String	否
GrantRead	赋予被授权者读取存储桶的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否
GrantWrite	赋予被授权者写入存储桶的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必
GrantReadAcp	赋予被授权者读取存储桶的访问控制列表（ACL）和存储桶策略（Policy）的权限。可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如`'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否
GrantWriteAcp	赋予被授权者写入存储桶的访问控制列表（ACL）和存储桶策略（Policy）的权限，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如`id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"`	String	否
GrantFullControl	赋予被授权者操作存储桶的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如`'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

检索存储桶及其权限

功能说明

HEAD Bucket 请求可以确认该存储桶是否存在，是否有权限访问。有以下几种情况：

- 存储桶存在且有读取权限，返回 HTTP 状态码200。
- 无存储桶读取权限，返回 HTTP 状态码403。
- 存储桶不存在，返回 HTTP 状态码404。

使用示例

检索存储桶信息：

```
cos.headBucket({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

判断存储桶是否存在：

```
function doesBucketExist() {
cos.headBucket({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
if (data) {
console.log('存储桶存在');
} else if (err.statusCode == 404) {
console.log('存储桶不存在');
} else if (err.statusCode == 403) {
console.log('没有该存储桶读权限');
}
```

```
});
}
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

删除存储桶

功能说明

删除指定账号下的空存储桶。注意，如果删除成功，则返回的 HTTP 状态码为200或204。

> 说明：
> 删除存储桶前，请确保存储桶内的数据和未完成上传的分块数据已全部清空，否则会无法删除存储桶。

使用示例

```
cos.deleteBucket({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

对象操作
上传对象

最近更新时间：2023-04-14 11:43:06

简介

本文档提供关于对象的高级接口，简单操作、分块操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object	简单上传对象	上传一个对象至存储桶
Append Object	追加上传对象	将对象以分块追加的方式上传至存储桶
POST Object	表单上传对象	使用表单请求上传对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传任务
Upload Part	上传分块	分块上传对象
List Parts	查询已上传块	查询特定分块上传操作中已上传的块
Complete Multipart Upload	完成分块上传	完成整个对象的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

简单操作

简单上传对象

功能说明

PUT Object 接口可以上传一个对象至指定存储桶中，该操作需要请求者对存储桶有 WRITE 权限。最大支持上传不超过5GB的对象，5GB以上对象请使用 分块上传 或 高级接口 上传。

注意：

- Key（文件名）不能以 `/` 结尾，否则会被识别为文件夹。
- Key（文件名）同名上传默认为覆盖操作。若您未开启版本控制且不想覆盖云上文件时，请确保上传时的Key不重复。
- 每个主账号（即同一个 APPID），存储桶的 ACL 规则数量最多为1000条，对象 ACL 规则数量不限制。如果您不需要进行对象 ACL 控制，请在上传时不要设置，默认继承存储桶权限。
- 上传之后，您可以用同样的 Key 生成预签名链接（下载请指定 method 为 GET，具体接口说明见下文），分享到其他端来进行下载。但注意如果您的文件是私有读权限，那么预签名链接只有一定的有效期。

使用示例

简单上传文件，适用于小文件上传。

```
const filePath = "temp-file-to-upload" // 本地文件路径
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
StorageClass: 'STANDARD',
/* 当Body为stream类型时，ContentLength必传，否则onProgress不能返回正确的进度信息 */
Body: fs.createReadStream(filePath), // 上传文件对象
ContentLength: fs.statSync(filePath).size,
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

上传 Buffer 作为文件内容：

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.txt', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: Buffer.from('hello!'), /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

上传字符串作为文件内容：

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.txt', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: 'hello!',
}, function(err, data) {
console.log(err || data);
});
```

上传base64作为文件内容：

```
var base64Url = '.....';
// 需要转为Buffer上传
var body = Buffer.from(base64Url.split(',')[1] , 'base64');
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: '1.png', /* 必须 */
Body: body,
}, function(err, data) {
console.log(err || data);
});
```

创建目录a：

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'a/', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: '',
}, function(err, data) {
console.log(err || data);
});
```

上传文件到指定目录a/b：

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'a/b/1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: fileObject, // 上传文件对象
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
```

版权所有：腾讯云计算（北京）有限责任公司

```
}
}, function(err, data) {
console.log(err || data);
});
```

上传对象（单链接限速）：

> 说明：
> 关于上传对象的限速说明，请参见 单链接限速。

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
StorageClass: 'STANDARD',
Body: fileObject, // 上传文件对象
Headers: {
'x-cos-traffic-limit': 819200, // 限速值设置范围为819200 - 838860800，即100KB/s - 10
0MB/s，如果超出该范围将返回400错误。
},
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
Body	上传文件的内容，可以为 FileStream、字符串、Buffer	Stream/E
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节），当 Body 为 Stream 类型时，ContentLength 必填	String
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存，过期后仅代表缓存失效，文件不会被删除	String
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，例如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String
GrantRead	赋予被授权者读取对象的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantReadAcp	赋予被授权者读取对象的访问控制列表（ACL）的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String

参数名	参数描述	类型
GrantWriteAcp	赋予被授权者写入对象的访问控制列表（ACL）的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如`'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如`'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE等，更多存储类型请参见 存储类型概述 文档。默认值：STANDARD	String
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据保存，大小限制2KB	String
onTaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function
- taskId	上传任务的编号	String
onProgress	进度的回调函数，进度回调响应对象（progressData）属性如下	Function
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number
- progressData.percent	文件上传的百分比，以小数形式呈现，例如：上传50%即为0.5	Number

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String
- Location	创建对象的外网访问域名	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

追加上传对象

功能说明

将对象以分块追加的方式上传至存储桶（APPEND Object）。

注意：

- COS Node.js SDK 版本需要大于等于 v2.11.2。
- 对象属性为 appendable 时才能使用本接口追加上传。
- 对象首次使用 APPEND Object 接口上传时，该对象的属性自动为 appendable。
- 可以使用 GET Object 或 HEAD Object 接口获取 x-cos-object-type 响应头来判断对象属性。

使用示例

初次追加上传对象:

```
cos.appendObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'test.txt', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: fileObject, // 上传文件对象
Position: 0, // 初次上传为0
}, function(err, data) {
console.log(err || data);
});
```

判断存储桶内的对象是否可追加对象:

```
cos.headObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'test.txt', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
if (err) return console.log(err);
// data.headers没有x-cos-object-type字段需要配置expose-headers，参考文档：https://clo
ud.tencent.com/document/product/436/13318
var objectType = data.headers['x-cos-object-type'];
console.log(objectType === 'appendable');
});
```

查询可追加对象的Position并追加上传:

```
cos.headObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'test.txt', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
if (err) return console.log(err);
// 首先取到要追加的文件当前长度，即需要上送的Position
var position = data.headers['content-length'];
cos.appendObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'test.txt', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: '66666',
Position: position,
},
function(err, data) {
if (err) return console.log(err);
// 也可以取到下一次上传的position继续追加上传
```

版权所有：腾讯云计算（北京）有限责任公司

```
// data.headers没有x-cos-next-append-position字段需要配置expose-headers，参考文档：https://cloud.tencent.com/document/product/436/13318
var nextPosition = data.headers['x-cos-next-append-position'];
console.log(nextPosition);
})
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
Body	上传文件的内容，可以为 FileStream、字符串、Buffer	Stream/Bufl
Position	追加操作的起始点，单位为字节。首次追加则设置 Position=0，后续追加则设置 Position 为当前 Object 的 content-length	Number
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，例如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
GrantRead	赋予被授权者读取对象的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantReadAcp	赋予被授权者读取对象的访问控制列表（ACL）的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantWriteAcp	赋予被授权者写入对象的访问控制列表（ACL）的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，默认值：STANDARD。更多存储类型请参见 存储类型概述	String

参数名	参数描述	类型
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据保存。大小限制2KB	String

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- RequestId	请求的唯一 ID	String

表单上传对象

Node.js SDK 未提供 POST Object 接口对应的方法，如果需要使用该接口，请参见 Web 端直传实践 里的“方案 B：使用 Form 表单上传”。

分块操作

查询分块上传

功能说明

List Multiparts Uploads 用来查询正在进行中的分块上传信息。单次最多列出1000个正在进行中的分块上传。

使用示例

获取前缀为 a 的未完成的 UploadId 列表，示例如下：

```
cos.multipartList({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a', /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Prefix	对象键前缀匹配，限定返回中只包含指定前缀的对象键。注意使用 prefix 查询时，返回的对象键 中仍会包含 Prefix	String	否
Delimiter	定界符为一个分隔符号，用于对对象键进行分组。一般是传 `/` 。所有对象键从 Prefix 或从头（如未指定 Prefix）到首个 delimiter 之间相同部分的路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix	String	否
EncodingType	规定返回值的编码格式，合法值：url	String	否
MaxUploads	设置最大返回的条目数量，合法取值为1 - 1000，默认1000	String	否
KeyMarker	与 upload-id-marker 一起使用 • 当 upload-id-marker 未被指定时： - ObjectName 字母顺序大于 key-marker 的条目将被列出 • 当 upload-id-marker 被指定时： - ObjectName 字母顺序大于 key-marker 的条目被列出 - ObjectName 字母顺序等于 key-marker 且 UploadID 大于 upload-id-marker 的条目将被列出	String	否
UploadIdMarker	与 key-marker 一起使用 • 当 key-marker 未被指定时： - upload-id-marker 将被忽略 • 当 key-marker 被指定时： - ObjectName 字母顺序大于 key-marker 的条目被列出 - ObjectName 字母顺序等于 key-marker 且 UploadID 大于 upload-id-marker 的条目将被列出	String	否

版权所有：腾讯云计算（北京）有限责任公司

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Bucket	分块上传的目标存储桶	String
- Encoding-Type	规定返回值的编码格式，合法值：url	String
- KeyMarker	列出条目从该 key 值开始	String
- UploadIdMarker	列出条目从该 UploadId 值开始	String
- NextKeyMarker	假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点	String
- NextUploadIdMarker	假如返回条目被截断，则返回 UploadId 就是下一个条目的起点	String
- MaxUploads	设置最大返回的条目数量，合法取值范围为1 - 1000	String
- IsTruncated	返回条目是否被截断，'true' 或者 'false'	String
- Prefix	对象键前缀匹配，限定返回中只包含指定前缀的对象键。	String
- Delimiter	定界符为一个分隔符号，用于对对象键进行分组。一般是传 `/` 。所有对象键从 Prefix 或从头（如未指定 Prefix）到首个 delimiter 之间相同部分的路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix	String
- CommonPrefixs	将 prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	ObjectArray
- - Prefix	显示具体的 Common Prefixs	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- Upload	分块上传的信息集合	ObjectArray
- - Key	对象的名称，即对象键	String
- - UploadId	表示本次分块上传的 ID	String
- - StorageClass	用于表示分块的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE 等，更多存储类型请参见 存储类型概述 文档	String
- - Initiator	用于表示本次上传发起者的信息	Object
- - - DisplayName	上传发起者的名称	String
- - - ID	上传发起者 ID，格式：`qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String
- - Owner	表示这些分块持有者的信息	Object
- - - DisplayName	分块持有者的名称	String
- - - ID	分块持有者 ID，格式：`qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String
- - Initiated	分块上传的起始时间	String

初始化分块上传

功能说明

Initiate Multipart Uploads 请求实现初始化分块上传，成功执行此请求后会返回 Upload ID ，用于后续的 Upload Part 请求。

使用示例

```
cos.multipartInit({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
if (data) {
uploadId = data.UploadId;
```

```
}
});
```

参数说明

参数名	参数描述	类型	必
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
CacheControl	RFC 2616 中定义的缓存策略，将作为对象的元数据保存	String	否
ContentDisposition	RFC 2616 中定义的文件名称，将作为对象的元数据保存	String	否
ContentEncoding	RFC 2616 中定义的编码格式，将作为对象的元数据保存	String	否
ContentType	RFC 2616 中定义的内容类型（MIME），将作为对象的元数据保存	String	否
Expires	RFC 2616 中定义的过期时间，将作为对象的元数据保存，过期后仅代表缓存失效，文件不会被删除	String	否
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String	否
GrantRead	赋予被授权者读取对象的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否

参数名	参数描述	类型	是
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE等，更多存储类型请参见 存储类型概述 文档。默认值：STANDARD	String	否
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据返回。大小限制2KB	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Bucket	分块上传的目标存储桶，格式为 BucketName-APPID，例如 examplebucket-1250000000	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
UploadId	在后续上传中使用的 ID	String

上传分块

功能说明

Upload Part 接口请求实现在初始化以后的分块上传，支持的块的数量为1 - 10000，块的大小为1MB - 5GB。

- 分块上传首先要进行初始化，用 Initiate Multipart Upload 接口初始化分块上传，得到一个 uploadId，该 ID 不但唯一标识这一分块数据，也标识了这分块数据在整个文件内的相对位置。
- 在每次请求 Upload Part 时候，需要携带 partNumber 和 uploadId，partNumber 为块的编号，支持乱序上传。

版权所有：腾讯云计算（北京）有限责任公司

- 当传入 uploadId 和 partNumber 都相同的时候，后传入的块将覆盖之前传入的块。当 uploadId 不存在时会返回404错误，错误码为 NoSuchUpload。

使用示例

```
const filePath = "temp-file-to-upload" // 本地文件路径
cos.multipartUpload({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
ContentLength: '1024',
UploadId: 'exampleUploadId',
PartNumber: 1,
Body: fs.createReadStream(filePath)
}, function(err, data) {
console.log(err || data);
if (data) {
eTag = data.ETag;
}
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
ContentLength	RFC 2616 中定义的 HTTP 请求内容长度（字节）	String	是
PartNumber	分块的编号	Number	是
UploadId	本次分块上传任务的编号	String	是
Body	上传文件分块的内容，可以为字符串，File 对象或者 Blob 对象	String\File\Blob	是
Expect	RFC 2616 中定义的 HTTP 请求内容长度（字节）。当使用 `Expect: 100-continue` 时，在收到服务端确认后，才会发送请求内容	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
ContentMD5	RFC 1864中定义的经过 Base64 编码的128-bit内容 MD5 校验值，此头部用来校验文件内容是否发生变化	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询已上传块

功能说明

List Parts 用来查询特定分块上传中的已上传的块，即列出指定 UploadId 所属的所有已上传成功的分块。

使用示例

```
cos.multipartListPart({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
UploadId: 'exampleUploadId', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
UploadId	标识本次分块上传的 ID，使用 Initiate Multipart Upload 接口初始化分块上传时得到的 UploadId	String	是
EncodingType	规定返回值的编码方式	String	否
MaxParts	单次返回最大的条目数量，默认为1000	String	否
PartNumberMarker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Bucket	分块上传的目标存储桶	String
- Encoding-type	规定返回值的编码方式	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- UploadId	标识本次分块上传的 ID，使用 Initiate Multipart Upload 接口初始化分块上传时得到的 UploadId	String
- Initiator	用来表示本次上传发起者的信息	Object
- - DisplayName	上传发起者的名称	String
- - ID	上传发起者 ID，格式：`qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String
- Owner	用来表示这些分块所有者的信息	Object
- - DisplayName	存储桶持有者的名称	String
- - ID	存储桶持有者 ID，一般为用户的 UIN	String
- StorageClass	用于表示这些分块的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE 等，更多存储类型请参见 存储类型概述 文档	String
- PartNumberMarker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
- NextPartNumberMarker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
- MaxParts	单次返回最大的条目数量	String
- IsTruncated	返回条目是否被截断，'true' 或者 'false'	String
- Part	分块信息列表	ObjectArray
- - PartNumber	块的编号	String
- - LastModified	块最后修改时间	String
- - ETag	块的 MD5 算法校验值	String
- - Size	块大小，单位为 Byte	String

完成分块上传

功能说明

Complete Multipart Upload 接口请求用来实现完成整个分块上传。当使用 Upload Parts 上传完所有块以后，必须调用该 API 来完成整个文件的分块上传。在使用该 API 时，您必须在请求 Body 中给出每一个块的 PartNumber 和

版权所有：腾讯云计算（北京）有限责任公司

ETag，用来校验块的准确性。
由于分块上传完后需要合并，而合并需要数分钟时间，因而当合并分块开始的时候，COS 就立即返回200的状态码，在合并的过程中，COS 会周期性的返回空格信息来保持连接活跃，直到合并完成，COS 会在 Body 中返回合并后块的内容。

- 当上传块小于1MB，在调用该 API 时，会返回400 EntityTooSmall。
- 当上传块编号不连续，在调用该 API 时，会返回400 InvalidPart。
- 当请求 Body 中的块信息未按序号从小到大排列，在调用该 API 时，会返回400 InvalidPartOrder。
- 当 UploadId 不存在，在调用该 API 时，会返回404 NoSuchUpload。

> 注意：
> 建议您及时完成分块上传或者舍弃分块上传，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

使用示例

```
cos.multipartComplete({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
UploadId: 'exampleUploadId', /* 必须 */
Parts: [
{PartNumber: 1, ETag: 'exampleETag'},
]
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
UploadId	上传任务编号	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Parts	用来说明本次分块上传中块的信息列表	ObjectArray	是
- PartNumber	分块的编号	Number	是
- ETag	每个块文件的 MD5 算法校验值 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Location	创建对象的外网访问域名	String
- Bucket	分块上传的目标存储桶	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- ETag	合并后文件的唯一 ID，格式："uuid-<分块数>" 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880-3"`，**注意前后携带双引号**	String

终止分块上传

功能说明

Abort Multipart Upload 用来实现终止一个分块上传操作并删除已上传的块。当您调用 Abort Multipart Upload 时，如果有正在使用这个 UploadId 上传块的请求，则 Upload Parts 会返回失败。当该 UploadId 不存在时，会返回404 NoSuchUpload。

> 注意：
> 建议您及时完成分块上传或者舍弃分块上传，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

使用示例

```
cos.multipartAbort({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
UploadId: 'exampleUploadId' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
UploadId	标识本次分块上传的 ID，使用 Initiate Multipart Upload 接口初始化分块上传时得到的 UploadId	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

高级接口（推荐）

该类方法是对上面原生方法的封装，实现了分块上传的全过程，支持并发分块上传，支持断点续传，支持上传任务的取消，暂停和重新开始等。

高级上传

功能说明

Upload File 实现高级上传，传入参数 SliceSize 可以控制文件大小超出一个数值（默认1MB）时自动使用分块上传，否则使用简单上传。

使用示例

```
const filePath = "temp-file-to-upload" // 本地文件路径
cos.uploadFile({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
FilePath: filePath, /* 必须 */
SliceSize: 1024 * 1024 * 5, /* 触发分块上传的阈值，超过5MB使用分块上传，非必须 */
onTaskReady: function(taskId) { /* 非必须 */
console.log(taskId);
},
onProgress: function (progressData) { /* 非必须 */
console.log(JSON.stringify(progressData));
},
onFileFinish: function (err, data, options) { /* 非必须 */
console.log(options.Key + '上传' + (err ? '失败' : '完成'));
},
}, function(err, data) {
```

```
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
FilePath	上传文件路径	String	是
SliceSize	表示文件大小超出一个数值时使用分块上传，单位 Byte，默认值1048576（1MB），小于等于该数值会使用 putObject 上传，大于该数值会使用 sliceUploadFile 上传	Number	否
AsyncLimit	分块的并发量，仅在触发分块上传时有效	Number	否
StorageClass	对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，更多存储类型请参见 存储类型概述	String	否
UploadAddMetaMd5	当上传时，给对象的元数据信息增加 x-cos-meta-md5 赋值为对象内容的 MD5 值，格式为 32 位小写字符串。例如：4d00d79b6733c9cc066584a02ed03410	String	否
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String	否
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String	否
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String	否
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	否
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String	否
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否
onTaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function	否
- taskId	上传任务的编号	String	否
onProgress	上传文件的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如，上传50%即为0.5	Number	否
onFileFinish	每个文件完成或错误回调	Function	否
- err	上传的错误信息	Object	否
- data	文件完成的信息	Object	否
- options	当前完成文件的参数信息	Object	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Location	上传完的文件访问地址	String
- Bucket	分块上传的目标存储桶,仅在触发分块上传时返回	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述，仅在触发分块上传时返回	String
- ETag	合并后文件的唯一 ID，格式："uuid-<分块数>" 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880-3"`，**注意前后携带双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

分块上传对象（断点续传）

功能说明

Slice Upload File 可用于实现文件的分块上传，适用于大文件上传。

使用示例

```
const filePath = "temp-file-to-upload" // 本地文件路径
cos.sliceUploadFile({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
FilePath: filePath, /* 必须 */
onTaskReady: function(taskId) { /* 非必须 */
console.log(taskId);
},
onHashProgress: function (progressData) { /* 非必须 */
console.log(JSON.stringify(progressData));
},
onProgress: function (progressData) { /* 非必须 */
console.log(JSON.stringify(progressData));
```

```
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
FilePath	上传文件路径	String	是
SliceSize	分块大小	Number	否
AsyncLimit	分块的并发量	Number	否
StorageClass	对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE 等，更多存储类型请参见 存储类型概述 文档	String	否
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String	否
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String	否
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String	否
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	否
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String	否
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String	否
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否
onTaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function	否
- taskId	上传任务的编号	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
onHashProgress	计算文件 MD5 值的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经校验的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的校验速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的校验百分比，以小数形式呈现，例如：校验50%即为0.5	Number	否
onProgress	上传文件的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如：上传50%即为0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- headers	请求返回的头部信息	Object
- Location	创建对象的外网访问域名	String
- Bucket	分块上传的目标存储桶	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- ETag	合并后文件的唯一 ID，格式："uuid-<分块数>" 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880-3"`，**注意前后携带双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

批量上传

功能说明

方法一：

批量上传可以直接多次调用 putObject 和 sliceUploadFile，达到批量上传效果。通过实例化参数 FileParallelLimit 控制文件并发数，默认3个并发。

方法二：

可以调用 cos.uploadFiles 实现批量上传，传入参数 SliceSize 可以控制文件大小超出一个数值时使用分块上传。以下是 uploadFiles 方法说明。

方法原型

调用 uploadFiles 操作：

```
const filePath1 = "temp-file-to-upload" // 本地文件路径
const filePath2 = "temp-file-to-upload" // 本地文件路径
cos.uploadFiles({
files: [{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
FilePath: filePath1,
onTaskReady: function(taskId) {
/* taskId可通过队列操作来取消上传cos.cancelTask(taskId)、停止上传cos.pauseTask(taskId)、重新开始上传cos.restartTask(taskId) */
console.log(taskId);
}
```

```
}, {
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '2.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
FilePath: filePath2,
onTaskReady: function(taskId) {
/* taskId可通过队列操作来取消上传cos.cancelTask(taskId)、停止上传cos.pauseTask(taskId)、重新开始上传cos.restartTask(taskId) */
console.log(taskId);
}
}],
SliceSize: 1024 * 1024 * 10, /* 设置大于10MB采用分块上传 */
onProgress: function (info) {
var percent = parseInt(info.percent * 10000) / 100;
var speed = parseInt(info.speed / 1024 / 1024 * 100) / 100;
console.log('进度：' + percent + '%; 速度：' + speed + 'Mb/s;');
},
onFileFinish: function (err, data, options) {
console.log(options.Key + '上传' + (err ? '失败' : '完成'));
},
}, function (err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
files	文件列表，每一项是传给 putObject 和 sliceUploadFile 的参数对象	Object	是
- Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
- Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
- FilePath	上传文件路径	String	是
- CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String	否
- ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String	否
- ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
- ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	否
- ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String	否
- Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String	否
- Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否
- onTaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function	否
-- taskId	上传任务的编号	String	否
SliceSize	表示文件大小超出一个数值时使用分块上传，单位 Byte，默认值1048576（1MB），小于等于该数值会使用 putObject 上传，大于该数值会使用 sliceUploadFile 上传	Number	是
onProgress	所有任务 进度汇总计算出来的上传进度	String	是
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如：上传50%即为0.5	Number	否
onFileFinish	每个文件完成或错误回调	Function	否
- err	上传的错误信息	Object	否
- data	文件完成的信息	Object	否
- options	当前完成文件的参数信息	Object	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- files	每个文件的 error 或 data	ObjectArray
- - error	上传的错误信息	Object
- - data	文件完成的信息	Object
- - options	当前完成文件的参数信息	Object

上传队列

Node.js SDK 针对 putObject 和 sliceUploadFile 发起的上传任务都有记录队列里，队列相关方法如下。

1. var taskList = cos.getTaskList() 可以获取任务列表。
2. cos.pauseTask()、cos.restartTask()、cos.cancelTask() 操作任务。
3. cos.on('list-update', callback); 可以监听列表和进度变化。

完整的队列使用示例请参见 demo-queue。

取消上传任务

根据 taskId 取消上传任务。

使用示例

```
var taskId = 'xxxxx'; /* 必须 */
cos.cancelTask(taskId);
```

参数说明

参数名	参数描述	类型	是否必填
taskId	文件上传任务的编号，在调用 sliceUploadFile 方法时，其 TaskReady 回调会返回该上传任务的 taskId	String	是

版权所有：腾讯云计算（北京）有限责任公司

暂停上传任务

根据 taskId 暂停上传任务。

使用示例

```
var taskId = 'xxxxx'; /* 必须 */
cos.pauseTask(taskId);
```

参数说明

参数名	参数描述	类型	是否必填
taskId	文件上传任务的编号，在调用 sliceUploadFile 方法时，其 TaskReady 回调会返回该上传任务的 taskId	String	是

重启上传任务

根据 taskId 重新开始上传任务，可以用于开启用户手动停止的（调用 pauseTask 停止）或者因为上传错误而停止的上传任务。

使用示例

```
var taskId = 'xxxxx'; /* 必须 */
cos.restartTask(taskId);
```

参数说明

参数名	参数描述	类型	是否必填
taskId	文件上传任务的编号，在调用 sliceUploadFile 方法时，其 TaskReady 回调会返回该上传任务的 taskId	String	是

版权所有：腾讯云计算（北京）有限责任公司

下载对象

最近更新时间：2023-03-08 17:11:19

简介

> 注意：
> 该接口用于读取对象内容，如果需要发起浏览器下载文件，可以通过 cos.getObjectUrl 获取 url 再触发浏览器下载，具体参见 预签名 URL 文档。

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地

简单操作

下载单个对象

功能说明

GET Object 接口请求可以获取存储桶里指定对象的内容并下载至本地。该 API 的请求者需要对目标对象有读取权限，或者目标对象向所有人开放了读取权限（公有读）。

使用示例

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data.Body);
});
```

指定 Range 获取文件内容：

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
```

版权所有：腾讯云计算（北京）有限责任公司

```
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Range: 'bytes=1-3', /* 非必须 */
}, function(err, data) {
console.log(err || data.Body);
});
```

下载文件到指定路径：

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Output: './exampleobject',
}, function(err, data) {
console.log(err || data);
});
```

下载文件到指定写文件流：

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Output: fs.createWriteStream('./exampleobject'),
}, function(err, data) {
console.log(err || data);
});
```

下载对象（单链接限速）：

> 说明：
> 关于下载对象的限速说明，请参见 单链接限速。

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Headers: {
'x-cos-traffic-limit': 819200, // 限速值设置范围为819200 - 838860800，即100KB/s - 10
0MB/s，如果超出该范围将返回400错误。
},
}, function(err, data) {
```

```
console.log(err || data.Body);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
Output	输出的文件路径或者一个写流，若不传入，则将完整内容写入回调函数data	String/WriteStream	否
ResponseContentType	设置响应头部中的 Content-Type 参数	String	否
ResponseContentLanguage	设置返回头部中的 Content-Language 参数	String	否
ResponseExpires	设置返回头部中的 Content-Expires 参数	String	否
ResponseCacheControl	设置返回头部中的 Cache-Control 参数	String	否
ResponseContentDisposition	设置返回头部中的 Content-Disposition 参数	String	否
ResponseContentEncoding	设置返回头部中的 Content-Encoding 参数	String	否
Range	RFC 2616 中定义的字节范围，范围值必须使用 bytes=first-last 格式，first 和 last 都是基于0开始的偏移量。例如 bytes=0-9 表示下载对象的开头10个字节的数据，如果不指定，则表示下载整个对象	String	否
IfModifiedSince	当对象在指定时间后被修改，则返回对应对象的元数据信息，否则返回304（not modified）	String	否
IfUnmodifiedSince	当对象在指定时间后未被修改，则返回对象，否则返回412（precondition failed）	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
IfMatch	当 ETag 与指定的内容一致，才返回对象，否则返回412（precondition failed）	String	否
IfNoneMatch	当 ETag 与指定的内容不一致，才返回对象，否则返回304（not modified）	String	否
VersionId	指定要下载的对象的版本 ID	String	否
onProgress	进度的回调函数，进度回调响应对象（progressData）属性如下	Function	否
- progressData.loaded	已经下载的对象部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个对象的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	对象的下载速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	对象下载的百分比，以小数形式呈现，例如：下载50%即为0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，304，403，404等	Number
- headers	请求返回的头部信息	Object
- CacheControl	RFC 2616 中定义的缓存指令，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- ContentDisposition	RFC 2616 中定义的文件名称，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- ContentEncoding	RFC 2616 中定义的编码格式，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- Expires	RFC 2616 中定义的缓存失效时间，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- x-cos-storage-class	对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE，**注意：如果没有返回该头部，则说明文件存储级别为 STANDARD （标准存储）**	String
- x-cos-meta-*	用户自定义的元数据	String
- NotModified	如果请求时带有 IfModifiedSince 则返回该属性，如果文件未被修改，则为 true，否则为 false	Boolean
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String
- Body	返回的文件内容，默认为 Buffer 形式	Buffer

批量下载对象

使用示例

按前缀下载多个对象（下载指定目录下的文件）：

```
var config = {
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
}

// 递归创建目录举例，可自行实现
function mkdirsSync(dirname) {
if(fs.existsSync(dirname)) {
return true;
}else{
if(mkdirsSync(path.dirname(dirname))) {
```

```
fs.mkdirSync(dirname);
return true;
}
}
}

// 下载单个文件
function downloadItem(Key, downloadPath) {
cos.getObject({
Bucket: config.Bucket,
Region: config.Region,
Key: Key,
Output: fs.createWriteStream(downloadPath),
},
function(err, data) {
console.log(err || data);
});
}

// 批量下载文件
function batchDownload(marker = undefined) {
cos.getBucket({
Bucket: config.Bucket,
Region: config.Region,
Prefix: '1', // 查询前缀是1的所有文件
Marker: marker,
MaxKeys: 1000,
},
function (listError, listResult) {
if (listError) return console.log(listError);
// 下载到本目录下的download目录下
var localPath = 'download';
listResult.Contents.forEach(function (item) {
var downloadPath = path.resolve(localPath, item.Key);
var pathParse = path.parse(item.Key);
if (pathParse.dir) {
// 如果被下载的对象在多层目录下，就在本地创建对应的目录
// 例如Key是a/b/1.png，就在本地创建a/b这样的目录结构
var mkdir = path.resolve(localPath, pathParse.dir);
mkdirsSync(mkdir);
downloadItem(item.Key, downloadPath);
} else {
downloadItem(item.Key, downloadPath);
}
});
})
```

```
}
batchDownload();
```

高级操作（推荐）

分块下载对象

功能说明

分块下载接口，支持分块并发下载。(要求sdk版本至少在v2.9.14)

方法原型

分块下载示例：

```
var Key = 'test.zip';
cos.downloadFile({
Bucket: 'examplebucket-1250000000',
Region: 'ap-beijing',
Key: Key,
FilePath: './' + Key, // 本地保存路径
ChunkSize: 1024 * 1024 * 8, // 分块大小
ParallelLimit: 5, // 分块并发数
RetryTimes: 3, // 分块失败重试次数
TaskId: '123', // 可以自己生成TaskId，用于取消下载

onProgress: function (progressData) {
console.log(JSON.stringify(progressData));
},
}, function (err, data) {
console.log(err || data);
});

// 取消下载任务
// cos.emit('inner-kill-task', {TaskId: '123'});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

参数名	参数描述	类型	是否必填
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
FilePath	下载文件是保存的路径	String	是
ChunkSize	下载时的分块大小	Number	否
ParallelLimit	分块并发数	Number	否
RetryTimes	分块下载失败重试次数	Number	否
onProgress	下载进度	String	否
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如：上传50%即为0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，304，403，404等	Number
- headers	请求返回的头部信息	Object
- CacheControl	RFC 2616 中定义的缓存指令，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- ContentDisposition	RFC 2616 中定义的文件名称，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- ContentEncoding	RFC 2616 中定义的编码格式，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- Expires	RFC 2616 中定义的缓存失效时间，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- x-cos-storage-class	对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE，**注意：如果没有返回该头部，则说明文件存储级别为 STANDARD （标准存储）**	String
- x-cos-meta-*	用户自定义的元数据	String
- NotModified	如果请求时带有 IfModifiedSince 则返回该属性，如果文件未被修改，则为 true，否则为 false	Boolean
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

列出对象

最近更新时间：2022-02-18 16:08:51

简介

本文档提供关于列出对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象

功能说明

查询存储桶下的部分或者全部对象。

> 注意：
> 在 COS 中，列出的对象键为 project/，表示是一个内容为空的对象，展示方式上模拟了传统文件夹。相关介绍可参见 文件夹和目录。

使用示例

示例一：列出目录 a 的所有文件。

```
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a/', /* Prefix表示列出的object的key以prefix开始，非必须 */
}, function(err, data) {
console.log(err || data.Contents);
});
```

返回值格式：

```
{
"Name": "examplebucket-1250000000",
"Prefix": "",
"Marker": "a/",
"MaxKeys": "1000",
"Delimiter": "",
"IsTruncated": "false",
```

```
"Contents": [{
"Key": "a/3mb.zip",
"LastModified": "2018-10-18T07:08:03.000Z",
"ETag": "\"05a9a30179f3db7b63136f30aa6aacae-3\"",
"Size": "3145728",
"Owner": {
"ID": "1250000000",
"DisplayName": "1250000000"
},
"StorageClass": "STANDARD"
}],
"statusCode": 200,
"headers": {}
}
```


示例二：列出目录 a 的文件，不深度遍历。

```
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a/', /* Prefix表示列出的object的key以prefix开始，非必须 */
Delimiter: '/', /* Deliter表示分隔符，设置为/表示列出当前目录下的object，设置为空表示列出所有的object，非必须 */
}, function(err, data) {
console.log(err || data.CommonPrefixes);
});
```

返回值格式：


```
{
"Name": "examplebucket-1250000000",
"Prefix": "a/",
"Marker": "",
"MaxKeys": "1000",
"Delimiter": "/",
"IsTruncated": "false",
"CommonPrefixes": [{
"Prefix": "a/1/"
}],
"Contents": [{
"Key": "a/3mb.zip",
"LastModified": "2018-10-18T07:08:03.000Z",
"ETag": "\"05a9a30179f3db7b63136f30aa6aacae-3\"",
"Size": "3145728",
"Owner": {
"ID": "1250000000",
```


版权所有：腾讯云计算（北京）有限责任公司

```
"DisplayName": "1250000000"
},
"StorageClass": "STANDARD"
}],
"statusCode": 200,
"headers": {}
}
```

示例三：列出目录a下所有文件。

```
var listFolder = function(marker) {
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a/', /* Prefix表示列出的object的key以prefix开始，非必须 */
Marker: marker,
MaxKeys: 1000,
}, function(err, data) {
if (err) {
return console.log('list error:', err);
} else {
console.log('list result:', data.Contents);
if (data.IsTruncated === 'true') listFolder(data.NextMarker);
else return console.log('list complete');
}
});
};
listFolder();
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Prefix	对象键前缀匹配，限定返回中只包含指定前缀的对象键	String	否
Delimiter	定界符。为一个分隔符号，用于对对象键进行分组。一般是传 `/` 。所有对象键从 Prefix 或从头（如未指定 Prefix）到首个 delimiter 之间相同部分的路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Marker	起始对象键标记。列出从 Marker 开始 MaxKeys 条目，默认顺序为 UTF-8 字典序	String	否
MaxKeys	单次返回最大的条目数量，默认1000，最大为1000	String	否
EncodingType	规定返回值的编码方式，可选值：url，代表返回的对象键为 URL 编码（百分号编码）后的值，例如“腾讯云”将被编码为 `%E8%85%BE%E8%AE%AF%E4%BA%91`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- headers	请求返回的头部信息	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- Name	存储桶的名称，格式为 <BucketName-APPID>，例如 examplebucket-1250000000	String
- Prefix	对象键前缀匹配，从该标记之后（不含）按照 UTF-8 字典序返回对象键条目	String
- Marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 Marker 开始	String
- MaxKeys	单次响应请求内返回结果的最大的条目数量	String
- Delimiter	定界符	String
- IsTruncated	响应请求条目是否被截断，值为 'true' 或者 'false'	String
- NextMarker	假如返回条目被截断，则返回 NextMarker 表示下一个条目的起点	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- CommonPrefixes	将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	ObjectArray
- - Prefix	单条 Common Prefix 的前缀	String
- EncodingType	返回值的编码方式，作用于 Delimiter，Marker，Prefix，NextMarker，Key	String
- Contents	对象元数据信息列表	ObjectArray
- - Key	对象键，即对象的名称	String
- - ETag	根据对象内容计算出的 MD5 算法校验值，例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- - Size	对象大小，单位 Byte	String
- - LastModified	对象最后修改时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	String
- - Owner	对象持有者信息	Object
- - - ID	对象持有者的完整 ID，格式为 `qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]`，例如 `qcs::cam::uin/100000000001:uin/100000000001`，其中100000000001为 uin	String
- - - DisplayName	对象持有者的名称	String
- - StorageClass	对象的存储类型，枚举值为：STANDARD、STANDARD_IA、ARCHIVE等，更多存储类型请参见 存储类型概述 文档	String

删除对象

最近更新时间：2022-01-10 14:27:52

简介

本文档提供关于对象的删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

删除单个对象

功能说明

DELETE Object 接口请求可以在 COS 的存储桶中将一个对象（Object）删除。该操作需要请求者对存储桶有 WRITE 权限。

使用示例

```
cos.deleteObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
VersionId	要删除的对象版本 ID 或 DeleteMarker 版本 ID	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，204，403，404等，**如果删除成功或者文件不存在则返回204或200，如果找不到指定的 Bucket，则返回404**	Number
- headers	请求返回的头部信息	Object

删除多个对象

功能说明

DELETE Multiple Objects 接口请求实现在指定存储桶中批量删除对象，单次请求最大支持批量删除1000个对象。对于响应结果，COS 提供 Verbose 和 Quiet 两种模式： Verbose 模式将返回每个对象的删除结果，Quiet 模式只返回报错的对象信息。

使用示例

删除多个文件：

```
cos.deleteMultipleObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Objects: [
{ Key: '1.jpg' }, /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
{ Key: '2.txt' }, /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
]
}, function(err, data) {
```

```
console.log(err || data);
});
```

按前缀删除多个对象（删除指定目录a下的文件）：

```
var deleteFiles = function (marker) {
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a/',
Marker: marker,
MaxKeys: 1000,
}, function (listError, listResult) {
if (listError) return console.log('list error:', listError);
var nextMarker = listResult.NextMarker;
var objects = listResult.Contents.map(function (item) {
return {Key: item.Key}
});
cos.deleteMultipleObject({
Bucket: bucket,
Region: region,
Objects: objects,
}, function (delError, deleteResult) {
if (delError) {
console.log('delete error', delError);
console.log('delete stop');
} else {
console.log('delete result', deleteResult);
if (listResult.IsTruncated === 'true') deleteFiles(nextMarker);
else console.log('delete complete');
}
});
});
}
deleteFiles();
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

参数名	参数描述	类型	是否必填
Quiet	布尔值，这个值决定了是否启动 Quiet 模式，值为 true 启动 Quiet 模式，值为 false 则启动 Verbose 模式，默认值为 false	Boolean	否
Objects	要删除的对象列表	ObjectArray	是
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
- VersionId	要删除的对象版本 ID 或 DeleteMarker 版本 ID	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200，204，403，404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，204，403，404等	Number
- headers	请求返回的头部信息	Object
- Deleted	说明本次删除成功的对象信息列表	ObjectArray
- - Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- - VersionId	如果参数传入了 VersionId，返回也会带上 VersionId，表示刚操作的对象版本或 DeleteMarker 版本	String
- - DeleteMarker	如果开启了版本控制，并且参数没有 VersionId，本次删除不会真正抹去文件内容，只新增一个 DeleteMarker 代表可见的文件已删除，枚举值：true、false	String
- - DeleteMarkerVersionId	当返回的 DeleteMarker 为 true 时，返回刚新增的 DeleteMarker 的 VersionId	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- Error	说明本次删除失败的对象信息列表	ObjectArray
- - Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- - Code	删除失败的错误码	String
- - Message	删除错误信息	String

复制与移动对象

最近更新时间：2023-01-30 11:50:32

简介

本文档提供关于对象的复制、移动操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object - Copy	设置对象复制（修改对象属性）	复制文件到目标路径

分块操作

API	操作名	操作描述
Initiate Multipart Upload	初始化分块复制	初始化分块复制任务
Upload Part - Copy	复制分块	将其他对象复制为一个分块
Complete Multipart Upload	完成分块复制	完成整个对象的分块复制

简单操作

复制对象

功能说明

PUT Object - Copy 请求创建一个已存在 COS 的对象的副本，即将一个对象从源路径（对象键）复制到目标路径（对象键）。在复制的过程中，对象元数据和访问控制列表（ACL）可以被修改。
用户可以通过该接口创建副本、修改对象元属性（源对象和目标文件的属性相同）、移动或重命名对象（先复制，再单独调用删除接口）。

> 注意：
> 建议对象大小1MB - 5GB，超过5GB的对象请使用高级接口复制对象 Slice Copy File。

使用示例

复制对象：

```
/* 把a/1.jpg复制一份到b/1.jpg */
cos.putObjectCopy({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'b/1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/a/1.jpg', /* 必须 */
/* CopySource中的Key含中文时，需要自行转义 */
// CopySource: `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/${encodeURIComponent('a/中文文件名.jpg')}`,
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
Region	存储桶所在地域，枚举值请参见 地域和访问域名
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述
CopySource	源对象 URL 路径，可以通过 URL 参数 ?versionId=<versionId> 参数指定指定历史版本
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，例如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**
GrantRead	赋予被授权者读取对象的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述
GrantWrite	赋予被授权者写入对象的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： ◦ 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` ◦ 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`
MetadataDirective	是否拷贝元数据，枚举值：Copy，Replaced，默认值 Copy。假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制；假如标记为 Replaced，按 Header 信息修改元数据。**当目标路径和原路径一致，即用户试图修改元数据时，必须为 Replaced**
CopySourceIfModifiedSince	当对象在指定时间后被修改，则执行操作，否则返回412。**可与 CopySourceIfNoneMatch 一起使用，与其他条件联合使用返回冲突**
CopySourceIfUnmodifiedSince	当对象在指定时间后未被修改，则执行操作，否则返回412。**可与 CopySourceIfMatch 一起使用，与其他条件联合使用返回冲突**
CopySourceIfMatch	当对象的 Etag 和给定一致时，则执行操作，否则返回412。**可与 CopySourceIfUnmodifiedSince 一起使用，与其他条件联合使用返回冲突**
CopySourceIfNoneMatch	当对象的 Etag 和给定不一致时，则执行操作，否则返回412。**可与 CopySourceIfModifiedSince 一起使用，与其他条件联合使用返回冲突**
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE 等，更多存储类型请参见 存储类型概述 文档。默认值：STANDARD
x-cos-meta-*	其他自定义的文件头部

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- ETag	文件的 MD5 算法校验值，例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- LastModified	返回对象最后被修改时间，例如2017-06-23T12:33:27.000Z	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

移动对象

功能说明

移动对象主要包括两个操作：复制源对象到目标位置，删除源对象。

由于 COS 通过存储桶名称（Bucket）和对象键（ObjectKey）来标识对象。移动对象也就意味着修改这个对象的标识，COS目前没有提供修改对象唯一标识名的单独接口，但是可以通过组合复制对象加上删除对象的基本操作，来达到修改对象标识的目的，从而实现移动对象。

使用示例

```
/* 把a/1.jpg移动到b/1.jpg */
/* 也可替换为高级复制cos.sliceCopyFile实现 */
cos.putObjectCopy({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'b/1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/a/1.jpg', /* 必须 */
}, function(err, data) {
```

```
if (err) return console.log(err);
/* 删除a/1.jpg */
cos.deleteObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'a/1.jpg' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
});
```

修改对象存储类型

功能说明

修改对象存储类型通过复制对象（源和目的是同一个对象）时设置StorageClass实现。

使用示例

```
/* 把根目录下的1.jpg设置为归档存储类型 */
/* 也可替换为高级复制cos.sliceCopyFile实现 */
cos.putObjectCopy({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/1.jpg', /* 必须 */
StorageClass: 'ARCHIVE', /* 设置为归档存储 */
}, function(err, data) {
console.log(err || data);
});
```

分块操作

初始化分块复制

功能说明

Initiate Multipart Uploads 请求实现初始化分块复制，成功执行此请求后会返回 Upload ID ，用于后续的 Upload Part - Copy 请求。

使用示例

```
cos.multipartInit({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
if (data) {
uploadId = data.UploadId;
}
});
```

参数说明

参数名	参数描述	类型	是
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
CacheControl	RFC 2616 中定义的缓存策略，将作为对象的元数据保存	String	否
ContentDisposition	RFC 2616 中定义的文件名称，将作为对象的元数据保存	String	否
ContentEncoding	RFC 2616 中定义的编码格式，将作为对象的元数据保存	String	否
ContentType	RFC 2616 中定义的内容类型（MIME），将作为对象的元数据保存	String	否
Expires	RFC 2616 中定义的过期时间，将作为对象的元数据保存，过期后仅代表缓存失效，文件不会被删除	String	否
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String	否

参数名	参数描述	类型	是
GrantRead	赋予被授权者读取对象的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE 等，更多存储类型请参见 存储类型概述 文档。默认值：STANDARD	String	否
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据返回。大小限制2KB	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Bucket	分块上传的目标存储桶，格式为 BucketName-APPID，例如 examplebucket-1250000000	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
UploadId	在后续上传中使用的 ID	String

版权所有：腾讯云计算（北京）有限责任公司

复制分块

功能说明

Upload Part - Copy 接口请求实现将一个对象的分块内容从源路径复制到目标路径。

> 注意：
> 使用上传分块对象，必须先初始化分块上传。在初始化分块上传的响应中，会返回一个唯一的描述符（upload ID），您需要在分块上传请求中携带此 ID。

使用示例

```
cos.uploadPartCopy({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject',
/* 必须 */
UploadId: 'exampleUploadId', /* 必须 */
PartNumber: '1', /* 必须 */
}, function(err, data) {
console.log(err || data);
if (data) {
eTag = data.ETag;
}
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
CopySource	源对象 URL 路径，可以通过 URL 参数 ?versionId=<versionId> 参数指定指定历史版本	String	是
PartNumber	分块拷贝的块号	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
UploadId	使用上传分块文件，必须先初始化分块上传。在初始化分块上传的响应中，会返回一个唯一的描述符（upload ID），您需要在分块上传请求中携带此 ID	String	是
CopySourceRange	源对象的字节范围，范围值必须使用 bytes=first-last 格式，first 和 last 都是基于0开始的偏移量。例如 bytes=0-9 表示您希望拷贝源对象的开头10个字节的数据，如果不指定，则表示拷贝整个对象	String	否
CopySourceIfMatch	当对象的 Etag 和给定一致时，则执行操作，否则返回412，可与 x-cos-copy-source-If-Unmodified-Since 一起使用，与其他条件联合使用返回冲突	String	否
CopySourceIfNoneMatch	当对象的 Etag 和给定不一致时，则执行操作，否则返回412，可与 x-cos-copy-source-If-Modified-Since 一起使用，与其他条件联合使用返回冲突	String	否
CopySourceIfUnmodifiedSince	当对象在指定时间后未被修改，则执行操作，否则返回412，可与 x-cos-copy-source-If-Match 一起使用，与其他条件联合使用返回冲突	String	否
CopySourceIfModifiedSince	当对象在指定时间后被修改，则执行操作，否则返回412，可与 x-cos-copy-source-If-None-Match 一起使用，与其他条件联合使用返回冲突	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- ETag	文件的 MD5 算法校验值，例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- LastModified	返回对象最后修改时间，GMT 格式	String

完成分块复制

功能说明

Complete Multipart Upload 接口请求用来实现完成整个分块复制。当使用 Copy Parts 复制完所有块以后，必须调用该 API 来完成整个文件的分块复制。在使用该 API 时，您必须在请求 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。
由于分块复制完后需要合并，而合并需要数分钟时间，因而当合并分块开始的时候，COS 就立即返回200的状态码，在合并的过程中，COS 会周期性的返回空格信息来保持连接活跃，直到合并完成，COS 会在 Body 中返回合并后块的内容。

- 当上传块小于1MB，在调用该 API 时，会返回400 EntityTooSmall。
- 当上传块编号不连续，在调用该 API 时，会返回400 InvalidPart。
- 当请求 Body 中的块信息未按序号从小到大排列，在调用该 API 时，会返回400 InvalidPartOrder。
- 当 UploadId 不存在，在调用该 API 时，会返回404 NoSuchUpload。

> 注意：
> 建议您及时完成分块上传或者舍弃分块上传，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

使用示例

```
cos.multipartComplete({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
UploadId: 'exampleUploadId', /* 初始化分块任务时拿到，必须 */
Parts: [
{PartNumber: '1', ETag: 'exampleETag'},
]
}, function(err, data) {
```

```
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
UploadId	上传任务编号	String	是
Parts	用来说明本次分块上传中块的信息列表	ObjectArray	是
- PartNumber	分块的编号	String	是
- ETag	每个块文件的 MD5 算法校验值 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- Location	创建对象的外网访问域名	String
- Bucket	分块上传的目标存储桶	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- ETag	合并后文件的唯一 ID，格式："uuid-<分块数>" 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880-3"`，**注意前后携带双引号**	String

高级接口（推荐）

强烈建议您使用高级接口，该类方法是对上面原生方法的封装，实现了分块复制的全过程。

分块复制对象

功能说明

Slice Copy File 可用于实现通过分块复制将一个文件从源路径复制到目标路径。在拷贝的过程中，对象元属性和 ACL 可以被修改。用户可以通过该接口实现文件移动，文件重命名，修改文件属性和创建副本。

方法原型

调用 Slice Copy File 操作：

```
/* 把a/1.jpg复制到b/1.jpg */
cos.sliceCopyFile({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'b/1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/a/1.jpg', /*
必须 */
onProgress:function (progressData) { /* 非必须 */
console.log(JSON.stringify(progressData));
}
},function (err,data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
CopySource	源对象 URL 路径，可以通过 URL 参数 ?versionId=<versionId> 参数指定指定历史版本	String	是
ChunkSize	分块复制时，每片的大小字节数，默认值 1048576（1MB）	Number	否
SliceSize	表示文件大小超出一个数值时使用分块复制，单位 Byte，默认值5G。小于等于该数值会使用 putObjectCopy 上传，大于该数值会使用 sliceCopyFile 上传	Number	否
onProgress	上传文件的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如：上传50%即为0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Location	创建对象的外网访问域名	String
- Bucket	分块上传的目标存储桶	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- ETag	合并后文件的 MD5 算法校验值 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

恢复归档对象

最近更新时间：2022-10-19 14:15:30

简介

本文档提供关于恢复归档对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
POST Object restore	恢复归档对象	将归档类型的对象取回访问

恢复归档对象

功能说明

POST Object restore 接口可以对一个通过 COS 归档为 archive 类型的对象进行恢复，恢复出的可读取对象是临时的，您可以设置需要保持可读，以及随后删除该临时副本的时间。您可以用 Days 参数来指定临时对象的过期时间，若超出该时间且期间您没有发起任何复制、延长等操作，该临时对象将被系统自动删除。临时对象仅为 archive 类型对象的副本，被归档的源对象在此期间将始终存在。

使用示例

```
cos.restoreObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
RestoreRequest: {
Days: 1,
CASJobParameters: {
Tier: 'Expedited'
}
},
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
RestoreRequest	用于恢复数据的容器	Object	是
- Days	设置临时副本的过期时间	Number	是
- CASJobParameters	归档存储工作参数的容器	Object	是
- - Tier	恢复归档存储类型的数据时，Tier 可以指定为 COS 支持的三种恢复模式，分别为： • Standard（标准取回模式，恢复任务在3 - 5小时内完成） • Expedited（快速取回模式，恢复任务在15分钟内可完成） • Bulk（批量取回模式，恢复任务在5 - 12小时内完成） 对于恢复深度归档存储类型的数据，有两种恢复模式，分别为： • Standard（标准取回模式，恢复时间为12 - 24小时） • Bulk（批量取回模式，恢复时间为24 - 48小时）	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询对象元数据

最近更新时间：2022-05-25 15:36:35

简介

本文档提供关于查询对象元数据操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

查询对象元数据

功能说明

查询对象的元数据信息。

使用示例

```
cos.headObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
IfModifiedSince	当在指定时间后被修改，则返回对应的元数据信息，否则返回304	String	否

版权所有：腾讯云计算（北京）有限责任公司

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，304等，如果在指定时间后未被修改，则返回304	Number
- headers	请求返回的头部信息	Object
- x-cos-object-type	用来表示对象是否可以被追加上传，枚举值：normal、appendable，默认 normal 不显示在返回中	String
- x-cos-storage-class	对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD 不显示在返回中	String
- x-cos-meta-*	用户自定义的 meta	String
- NotModified	对象是否在指定时间后未被修改	Boolean
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

检查对象是否存在

最近更新时间：2022-07-07 14:14:11

简介

本文档提供快捷查询存储桶中某个对象是否存在的示例代码。示例代码实际调用了 HEAD Object COS API，是该接口的简化版。

HEAD Object 除了检查对象是否存在，主要功能为返回对象元数据。如需查看包含了 HEAD Object 完整功能的 SDK 接口，请参见 查询对象元数据。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

功能说明

检查存储桶中是否存在某个对象,可通过封装cos.headObject方法实现。

使用示例

```
function doesObjectExist() {
cos.headObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
if (data) {
console.log('对象存在');
} else if (err.statusCode == 404) {
console.log('对象不存在');
} else if (err.statusCode == 403) {
console.log('没有该对象读权限');
}
});
}
```

参数说明

参数名	参数描述	类型	是否必填

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
IfModifiedSince	当在指定时间后被修改，则返回对应的元数据信息，否则返回304	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，304等，如果在指定时间后未被修改，则返回304	Number
- headers	请求返回的头部信息	Object
- x-cos-object-type	用来表示对象是否可以被追加上传，枚举值：normal、appendable，默认 normal 不显示在返回中	String
- x-cos-storage-class	对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD 不显示在返回中	String
- x-cos-meta-*	用户自定义的 meta	String
- NotModified	对象是否在指定时间后未被修改	Boolean
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String

参数名	参数描述	类型
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

对象访问 URL

最近更新时间：2023-05-24 15:27:03

简介

本文档提供获取已上传到存储的对象访问 URL 的代码示例。

获取对象访问 URL

功能说明

查询对象访问的 URL，该接口不会判断对象是否真实存在。

说明：

- 如何使生成的对象 URL 在浏览器中打开是预览，而不是下载：在获取的 url 后拼接参数 response-content-disposition=inline。
- 如何使生成的对象 URL 在浏览器中打开是下载，而不是预览：在获取的 url 后拼接参数 response-content-disposition=attachment。
- 下载时如何重命名文件：在获取的 URL 后拼接参数 filename，例如 response-content-disposition=attachment;filename=自定义文件名。
- 若使用临时密钥生成预签名 URL，请确保申请临时密钥的 action 添加了 `"name/cos:GetObject"` 权限。

使用示例

获取在浏览器默认下载的 url：

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', // 填入您自己的存储桶，必须字段
Region: 'COS_REGION', // 存储桶所在地域，例如 ap-beijing，必须字段
Key: '头像.jpg', // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段
Sign: true, // 获取带签名的对象 URL

},
function (err, data) {
if (err) return console.log(err);
```

```
/* url为对象访问 url */
var url = data.Url;
/* 复制 downloadUrl 的值到浏览器打开会自动触发下载 */
var downloadUrl =
data.Url +
(data.Url.indexOf('?') > -1 ? '&' : '?') +
'response-content-disposition=attachment;filename=图片.jpg'; // 补充强制下载的参数并重命名下载后的文件
}
);
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
Sign	是否返回带有签名的 Url，默认为 true，当对象为私有读时，获取到不带签名的 url 依然没有权限访问	Boolean	否
Protocol	可选填为 `http:` 或 `https:`，默认为 `http:`（带冒号）	String	否
Domain	存储桶访问域名，默认为 {BucketName-APPID}.cos.{Region}.myqcloud.com	String	否
Method	操作方法，例如 GET，POST，DELETE，HEAD 等 HTTP 方法，默认为 GET	String	否
Query	签名中要签入的请求参数，{key: 'val'} 的格式	Object	否
Headers	签名中要签入的请求头部，{key: 'val'} 的格式	Object	否
Expires	签名几秒后失效，默认为 900 秒	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- Url	计算得到的 Url	String

生成预签名链接

最近更新时间：2023-04-12 14:27:06

简介

Node.js SDK 提供获取对象 URL、获取请求预签名 URL 接口，详细操作请查看本文说明和示例。

说明：

- 建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。申请临时密钥的 action 需添加 `"name/cos:GetObject"` 权限。
- 如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。

计算签名

COS XML API 的请求里，私有资源操作都需要鉴权凭证 Authorization，用于判断当前请求是否合法。

鉴权凭证使用方式有两种：

1. 放在 header 参数里使用，字段名：authorization。
2. 放在 url 参数里使用，字段名：sign。

COS.getAuthorization 方法用于计算鉴权凭证（Authorization），用以验证请求合法性的签名信息。

注意：
该方法推荐只在前端调试时使用，项目上线不推荐使用前端计算签名的方法，有暴露密钥的风险。

使用示例

获取文件下载的鉴权凭证：

```
// SECRETID 和 SECRETKEY 请登录 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
var COS = require('cos-nodejs-sdk-v5');
```

```
var Authorization = COS.getAuthorization({
SecretId: process.env.SecretId, // 推荐使用环境变量获取；用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
SecretKey: process.env.SecretKey, // 推荐使用环境变量获取；用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
Method: 'get',
Key: 'a.jpg',
Expires: 60,
Query: {},
Headers: {},
});
```

参数说明

参数名	参数描述	类型	是否必填
SecretId	用户的 SecretId	String	是
SecretKey	用户的 SecretKey	String	是
Method	操作方法，如 get，post，delete， head 等 HTTP 方法	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，**如果请求操作是对文件的，则为文件名，且为必须参数**。如果操作是对于存储桶，则为空	String	否
Query	签名中要签入的请求参数，{key: 'val'} 的格式	Object	否
Headers	签名中要签入的请求头部，{key: 'val'} 的格式	Object	否
Expires	签名几秒后失效，默认为 900 秒	Number	否

返回值说明

返回值是计算得到的鉴权凭证字符串 authorization。

获取请求预签名 URL

下载请求示例

示例一：获取不带签名 Object Url

```
var url = cos.getObjectUrl({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: false,
});
```

示例二：获取带签名 Object Url

```
var url = cos.getObjectUrl({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
});
```

示例三：如果签名过程是异步获取，需要通过 callback 获取带签名 Url

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: false,
},
function (err, data) {
console.log(err || data.Url);
}
);
```

示例四：指定链接有效时间

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: true,
Expires: 3600, // 单位秒
},
function (err, data) {
console.log(err || data.Url);
}
);
```

示例五：获取文件 Url 并下载文件

版权所有：腾讯云计算（北京）有限责任公司

```
var request = require('request');
var fs = require('fs');
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: true,
},
function (err, data) {
if (err) return console.log(err);
console.log(data.Url);
var req = request(data.Url, function (err, response, body) {
console.log(err || body);
});
var writeStream = fs.createWriteStream(__dirname + '/1.jpg');
req.pipe(writeStream);
}
);
```

示例六：生成预签名 URL，并在签名中携带 Query 和 Header

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: true,
/* 传入的请求参数需与实际请求相同，能够防止用户篡改此 HTTP 请求的参数 */
Query: {
'imageMogr2/thumbnail/200x/': '',
},
/* 传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的 HTTP 请求头部 */
Headers: {
host: 'xxx', /* 指定 host 访问，非指定的 host 访问会报错403 */
},
},
function (err, data) {
console.log(err || data.Url);
}
);
```

上传请求示例

示例一：获取预签名 Put Object 上传 Url

```
var request = require('request');
var fs = require('fs');
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Method: 'PUT',
Sign: true,
},
function (err, data) {
if (err) return console.log(err);
console.log(data.Url);
var readStream = fs.createReadStream(__dirname + '/1.jpg');
var req = request(
{
method: 'PUT',
url: data.Url,
},
function (err, response, body) {
console.log(err || body);
}
);
readStream.pipe(req);
}
);
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，**如果请求操作是对文件的，则为文件名，且为必须参数**。如果操作是对于存储桶，则为空	String	是
Sign	是否返回带有签名的 Url，默认为 true	Boolean	否
Protocol	可选填为 `http:` 或 `https:`，默认为 `http:`（带冒号）	String	否
Domain	存储桶访问域名，默认为 {BucketName-APPID}.cos.{Region}.myqcloud.com	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Method	操作方法，如 get，post，delete， head 等 HTTP 方法，默认为 get	String	否
Query	签名中要签入的请求参数，{key: 'val'} 的格式	Object	否
Headers	签名中要签入的请求头部，{key: 'val'} 的格式	Object	否
Expires	签名几秒后失效，默认为 900 秒	Number	否

返回值说明

返回值是一个字符串，两种情况：

1. 如果签名计算可以同步计算（如：实例化传入了 SecretId 和 SecretKey），则默认返回带签名的 url。
2. 否则返回不带签名的 url。

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- Url	计算得到的 Url	String

版权所有：腾讯云计算（北京）有限责任公司

预请求跨域配置

最近更新时间：2022-01-10 14:27:53

简介

本文档提供关于预请求跨域配置操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
Options Object	预请求跨域配置	用预请求来确认是否可以发送真正的跨域请求

功能说明

OPTIONS Object 接口实现对对象进行跨域访问配置的预请求。即在发送跨域请求之前会发送一个 OPTIONS 请求并带上特定的来源域，HTTP 方法和 HEADER 信息等给 COS，以决定是否可以发送真正的跨域请求。当 CORS 配置不存在时，请求返回403 Forbidden。**用户可以通过 PUT Bucket cors 接口来开启存储桶的 CORS 支持。**

使用示例

```
cos.optionsObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Origin: 'https://www.qq.com', /* 必须 */
AccessControlRequestMethod: 'PUT', /* 必须 */
AccessControlRequestHeaders: 'origin,accept,content-type' /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Origin	模拟跨域访问的请求来源域名	String	是
AccessControlRequestMethod	模拟跨域访问的请求 HTTP 方法	String	是
AccessControlRequestHeaders	模拟跨域访问的请求头部	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- headers	请求返回的头部信息	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- AccessControlAllowOrigin	模拟跨域访问的请求来源域名，中间用逗号间隔，当来源不允许的时候，此 Header 不返回，例如 `*`	String
- AccessControlAllowMethods	模拟跨域访问的请求 HTTP 方法，中间用逗号间隔，当请求方法不允许的时候，此 Header 不返回，例如 PUT，GET，POST，DELETE，HEAD	String
- AccessControlAllowHeaders	模拟跨域访问的请求头部，中间用逗号间隔，当模拟任何请求头部不允许的时候，此 Header 不返回该请求头部，例如 accept，content-type，origin，authorization	String
- AccessControlExposeHeaders	跨域支持返回头部，中间用逗号间隔，例如 ETag	String
- AccessControlMaxAge	设置 OPTIONS 请求得到结果的有效期，例如3600	String
- OptionsForbidden	OPTIONS 请求是否被禁止，如果返回的 HTTP 状态码为403，则为 true	Boolean

版权所有：腾讯云计算（北京）有限责任公司

单链接限速

最近更新时间：2022-01-10 14:27:53

简介

本文档提供关于调用上传下载接口时对链接进行限速。

使用说明

限速值设置范围为**819200 - 838860800**，单位默认为 bit/s，即100KB/s - 100MB/s，如果超出该范围将返回400错误。

> 说明：
> 关于下载对象的限速说明，请参见 单链接限速。

示例一：上传时对单链接限速

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
StorageClass: 'STANDARD',
Body: fileObject, // 上传文件对象
Headers: {
'x-cos-traffic-limit': 819200, // 限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，如果超出该范围将返回400错误。
},
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

示例二：下载时对单链接限速

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
```

```
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Headers: {
'x-cos-traffic-limit': 819200, // 限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，如果超出该范围将返回400错误。
},
}, function(err, data) {
console.log(err || data.Body);
});
```

服务端加密

最近更新时间：2022-03-04 11:06:29

简介

本文档提供关于如何使用在上传对象时开启服务端加密。服务端加密的密钥分为三种：

- COS 托管加密密钥
- 客户提供的加密密钥
- KMS 托管加密密钥

使用 COS 托管加密密钥的服务端加密（SSE-COS）保护数据

由腾讯云 COS 托管主密钥和管理数据。COS 会帮助您在数据写入数据中心时自动加密，并在您取用该数据时自动解密。目前支持使用 COS 主密钥对数据进行 AES-256 加密。

Node.js 通过调用 API 时传入 `ServerSideEncryption` 参数完成。

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: 'hello!',
ServerSideEncryption: 'AES256',
}, function(err, data) {
console.log(err || data);
});
```

使用客户提供的加密密钥的服务端加密 （SSE-C）保护数据

加密密钥由用户自己提供，用户在上传对象时，COS 将使用用户提供的加密密钥对用户的数据进行 AES-256
Node.js SDK 通过调用 API 时传入 `SSECustomerKey` 参数完成。

注意：

- 该加密所运行的服务需要使用 HTTPS 请求。
- customerKey：用户提供的密钥，传入一个32字节的字符串的base64编码，字符串支持数字、字母、字符的组合，不支持中文。
- 如果上传源文件（putObject）使用了该参数，那么在使用 headObject（查询）、getObject（下载）、multipartInit（初始化分块上传）、multipartUpload（上传分块）、putObjectCopy（复制）时对源对象操作

版权所有：腾讯云计算（北京）有限责任公司

的时候也要调用该方法。

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: 'hello!',
SSECustomerAlgorithm: 'AES256',
SSECustomerKey: 'MDEyMzQ1Njc4OUFCQ0RFRjAxMjM0NTY3ODlBQkNERUY',
SSECustomerKeyMD5: 'U5L61r7jcwdNvT7frmUG8g==',
}, function(err, data) {
console.log(err || data);
});
```

使用 KMS 托管加密密钥的服务端加密（SSE-KMS）保护数据

SSE-KMS 加密即使用 KMS 托管密钥的服务端加密。KMS 是腾讯云推出的一款安全管理类服务，使用经过第三方认证的硬件安全模块 HSM（Hardware Security Module）来生成和保护密钥。它能够帮助用户轻松创建和管理密钥，满足用户多应用多业务的密钥管理需求以及满足监管和合规要求。关于如何开通 KMS 服务请参考：服务端加密概述。

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: 'hello!',
ServerSideEncryption: 'cos/kms', /* 上传或复制对象（包括简单上传/复制与分块上传/复制）时必选，下载对象时不能指定此头部 */
// SSEKMSKeyId: 用于指定 KMS 的用户主密钥 CMK，如不指定，则使用 COS 默认创建的 CMK，更多详细信息可参见 SSE-KMS 加密
SSEKMSKeyId: 'xxxx', /* 这里填入自己的key，非必须 */
// SSEContext: 用于指定加密上下文，值为 JSON 格式加密上下文键值对的 Base64 编码。
SSEContext: 'eyJhIjoiYXNkZmEiLCJiIjoiMTIzMzIxIn0=', /* 这里指定加密上下文，需要自行使用Base64编码，非必须 */
}, function(err, data) {
console.log(err || data);
});
```

异地容灾
版本控制

最近更新时间：2020-05-07 18:02:48

简介

本文档提供关于版本控制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

设置版本控制

功能说明

PUT Bucket versioning 接口实现启用或者暂停存储桶的版本控制功能。

1. 如果您从未在存储桶上启用过版本控制，则 GET Bucket versioning 请求不返回版本状态值。
2. 开启版本控制功能后，只能暂停，不能关闭。
3. 设置版本控制状态值为 Enabled 或者 Suspended，表示开启版本控制和暂停版本控制。
4. 设置存储桶的版本控制功能，您需要有存储桶的写权限。

请求示例

```
cos.putBucketVersioning({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
VersioningConfiguration: {
Status: "Enabled"
}
}, function (err, data) {
console.log(err || data);
});
```

版权所有：腾讯云计算（北京）有限责任公司

参数说明

参数名称	描述	类型	是否必填
Bucket	开启或暂停版本控制的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
VersioningConfiguration	定义存储桶的版本控制配置信息	Object	是
- Status	版本控制是否打开的状态，枚举值：Enabled、Suspended。 • Enabled 表示打开 • Suspended 表示暂停	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询版本控制

功能说明

查询存储桶的版本控制信息。

请求示例

```
cos.getBucketVersioning({
Bucket: 'examplebucket-1250000000', /* 必须 */
```

```
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function (err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询版本控制的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- VersioningConfiguration	存储桶的版本控制配置信息，若从未开启过，则为空对象 '{}'	Object
- - Status	版本控制是否打开的状态，枚举值：Enabled、Suspended	String

版权所有：腾讯云计算（北京）有限责任公司

跨地域复制

最近更新时间：2020-05-07 18:02:49

简介

本文档提供关于跨地域复制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

设置跨地域复制

功能说明

设置存储桶的跨地域复制规则。

> 存储桶必须开启版本控制才能使用跨地域复制功能。

请求示例

```
cos.putBucketReplication({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
ReplicationConfiguration: { /* 必须 */
Role: "qcs::cam::uin/100000000001:uin/100000000001",
Rules: [{
ID: "1",
Status: "Enabled",
Prefix: "sync/",
Destination: {
Bucket: "qcs::cos:ap-beijing::destinationbucket-1250000000",
StorageClass: "Standard",
}
}]
```

```
}
}, function (err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型
Bucket	源存储桶，格式：BucketName-APPID	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
ReplicationConfiguration	定义跨地域复制规则	Object
- Role	复制过程以什么角色的身份 格 式：`qcs::cam::uin/100000000001:uin/100000000011` 其中100000000001是主账号，100000000011是子账号	Object
- Rules	复制具体规则列表	ObjectArray
- - ID	标注具体 Rule 的 ID	String
- - Status	标识 Rule 是否生效，枚举值：Enabled、Disabled	String
- - Prefix	前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空	String
- - Destination	目标存储桶信息	Object
- - - Bucket	目标存储桶的名称 格式：`qcs::cos:<Region>::<BucketName-APPID>` 例如：`qcs::cos:ap-beijing::destinationbucket-1250000000`	Object
- - - StorageClass	复制后的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD	Object

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object

参数名称	描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询跨地域复制

功能说明

查询存储桶的跨地域复制规则。

请求示例

```
cos.getBucketReplication({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"ReplicationConfiguration": {
"Role": "qcs::cam::uin/100000000001:uin/100000000001",
"Rules": {
"ID": "1",
"Status": "Enabled",
"Prefix": "sync/",
"Destination": {
"Bucket": "qcs::cos:ap-beijing::destinationbucket-1250000000",
"StorageClass": "Standard"
}
}
},
"statusCode": 200,
```

版权所有：腾讯云计算（北京）有限责任公司

```
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询跨地域复制的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- ReplicationConfiguration	跨地域复制规则	Object
- - Role	复制过程以什么角色的身份 格 式：`qcs::cam::uin/100000000001:uin/100000000011` 其中100000000001是主账号，100000000011是子账号	Object
- - Rules	复制具体规则列表	ObjectArr
- - - ID	标注具体 Rule 的 ID	String
- - - Status	标识 Rule 是否生效，枚举值：Enabled、Disabled	String
- - - Prefix	前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空	String
- - - Destination	目标存储桶信息	Object
- - - - Bucket	目标存储桶的名称 格式：`qcs::cos:<Region>::<BucketName-APPID>` 例如：`qcs::cos:ap-beijing::destinationbucket-1250000000`	Object
- - - - StorageClass	复制后的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD	Object

版权所有：腾讯云计算（北京）有限责任公司

删除跨地域复制

功能说明

删除存储桶的跨地域复制规则。

请求示例

```
cos.deleteBucketReplication({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	删除跨地域复制的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

数据管理
生命周期

最近更新时间：2020-05-07 18:02:50

简介

本文档提供关于生命周期的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket lifecycle	设置生命周期	设置存储桶的生命周期管理的配置
GET Bucket lifecycle	查询生命周期	查询存储桶生命周期管理的配置
DELETE Bucket lifecycle	删除生命周期	删除存储桶生命周期管理的配置

设置生命周期

功能说明

设置指定存储桶的生命周期配置信息（PUT Bucket lifecycle）。

请求示例

示例一：上传30天后，当前版本沉降至低频存储。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "1",
"Status": "Enabled",
"Filter": {},
"Transition": {
"Days": "30",
"StorageClass": "STANDARD_IA"
}
}],
}, function(err, data) {
console.log(err || data);
});
```

示例二：指定目录前缀 `dir/`，上传90天后，当前版本沉降至归档存储。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "2",
"Filter": {
"Prefix": "dir/",
},
"Status": "Enabled",
"Transition": {
"Days": "90",
"StorageClass": "ARCHIVE"
}
}],
}, function(err, data) {
console.log(err || data);
});
```

示例三：上传180天后，清理过期文件删除标记。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "3",
"Status": "Enabled",
"Filter": {},
"Expiration": {
"Days": "180"
}
}],
}, function(err, data) {
console.log(err || data);
});
```

示例四：上传30天后，删除碎片。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "4",
"Status": "Enabled",
```

版权所有：腾讯云计算（北京）有限责任公司

```
"Filter": {},
"AbortIncompleteMultipartUpload": {
"DaysAfterInitiation": "30"
}
}],
}, function(err, data) {
console.log(err || data);
});
```

示例五：历史版本生成30天后沉降至归档存储。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "5",
"Status": "Enabled",
"Filter": {},
"NoncurrentVersionTransition": {
"NoncurrentDays": "30",
"StorageClass": 'ARCHIVE'
}
}],
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	设置生命周期的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Rules	指定生命周期规则列表	ObjectArray	是
- ID	规则的唯一标识 ID	String	是
- Status	规则的开启状态，枚举值：Enabled、Disabled	String	是
- Filter	指定过滤条件	Object	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
- - Prefix	规则要匹配上的对象键前缀	String	否
- Transition	规则转换属性，表示对象存储级别何时转换为 Standard_IA 或 Archive	Object	否
- - Days	指明规则对应的动作在对象最后的修改日期过后多少天执行，该字段有效值是非负整数，最大支持3650天	Number	是
- - StorageClass	表示规则对应的动作执行后变更对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD	String	否
- NoncurrentVersionTransition	指明历史版本对象转换属性	ObjectArray	否
- - NoncurrentDays	指明历史版本对象在该值确定的生效天数后进行转换	Number	是
- - StorageClass	表示规则对应的动作执行后变更对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE	String	是
- Expiration	规则过期属性	Object	否
- - ExpiredObjectDeleteMarker	删除过期对象删除标记，枚举值 true，false，与 Days 不可并存	Boolean	是
- - Days	指明规则对应的动作在对象最后的修改日期过后多少天进行删除操作，与 ExpiredObjectDeleteMarker 不可并存	Number	是
- AbortIncompleteMultipartUpload	表示删除碎片	Object	否
- - DaysAfterInitiation	碎片在该值所确定的生效天数后被删除，按文件上传时间开始算，必须为正整数	Number	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询生命周期

功能说明

GET Bucket lifecycle 接口可以查询存储桶生命周期管理的配置。

请求示例

```
cos.getBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"Rules": [{
"ID": "1",
"Filter": "",
"Status": "Enabled",
"Transition": {
"Days": "30",
"StorageClass": "STANDARD_IA"
}
}, {
"ID": "2",
"Filter": {
```

```
"Prefix": "dir/"
},
"Status": "Enabled",
"Transition": {
"Days": "90",
"StorageClass": "ARCHIVE"
}
}, {
"ID": "3",
"Filter": "",
"Status": "Enabled",
"Expiration": {
"Days": "180"
}
}, {
"ID": "4",
"Filter": "",
"Status": "Enabled",
"AbortIncompleteMultipartUpload": {
"DaysAfterInitiation": "30"
}
}, {
"ID": "5",
"Filter": "",
"Status": "Enabled",
"NoncurrentVersionTransition:": {
"NoncurrentDays": "30",
"StorageClass": "ARCHIVE"
}
}],
"statusCode": 200,
"headers": {}
}
```


参数说明

参数名称	描述	类型	是否必填
Bucket	查询生命周期的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Rules	指定生命周期规则列表	ObjectArray
- - ID	规则的唯一标识 ID	String
- - Status	规则的开启状态，枚举值：Enabled、Disabled	String
- - Filter	指定过滤条件	Object
- - - Prefix	规则要匹配上的对象键前缀	String
- - Transition	规则转换属性，表示对象存储级别何时转换为 Standard_IA 或 Archive	ObjectArray
- - - Days	指明规则对应的动作在对象最后的修改日期过后多少天执行，该字段有效值是非负整数，最大支持3650天	Number
- - - StorageClass	表示规则对应的动作执行后变更对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE	String
- - NoncurrentVersionTransition	指明历史版本对象转换属性	ObjectArray
- - - NoncurrentDays	指明历史版本对象在该值确定的生效天数后进行转换	Number
- - - StorageClass	表示规则对应的动作执行后变更对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE	String
- - Expiration	规则过期属性	Object
- - - ExpiredObjectDeleteMarker	删除过期对象删除标记，枚举值 true，false，与 Days 不可并存	Boolean

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- - - Days	指明规则对应的动作在对象最后的修改日期过后多少天进行删除操作，与 ExpiredObjectDeleteMarker 不可并存	Number
- - AbortIncompleteMultipartUpload	表示删除碎片	Object
- - - DaysAfterInitiation	碎片在该值所确定的生效天数后被删除，按文件上传时间开始算，必须为正整数	Number

删除生命周期

功能说明

DELETE Bucket lifecycle 接口可以删除存储桶生命周期管理的配置。

请求示例

```
cos.deleteBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	被删除生命周期配置的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object

参数名称	描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

日志管理

最近更新时间：2020-05-07 18:02:51

简介

本文档提供关于日志管理的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket logging	设置日志管理	为源存储桶开启日志记录
GET Bucket logging	查询日志管理	查询源存储桶的日志配置信息

设置日志管理

功能说明

PUT Bucket Logging 用于为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中。

> 只有源存储桶拥有者才可进行该请求操作。

请求示例

示例1：设置将源存储桶 `sourcebucket-1250000000` 的日志信息投递到目标存储桶 `targetbucket-1250000000` 的 `bucket-logging-prefix/` 路径下。

```
cos.putBucketLogging({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
BucketLoggingStatus: { /* 必须 */
LoggingEnabled: {
TargetBucket: 'targetbucket-1250000000',
TargetPrefix: 'bucket-logging-prefix/'
}
}
}, function(err, data) {
console.log(err || data);
});
```

示例2：关闭目标存储桶 `sourcebucket-1250000000` 的日志投递。

```
cos.putBucketLogging({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
BucketLoggingStatus: {} /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置日志管理的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
BucketLoggingStatus	说明日志记录配置的状态，如果为空字符串则意为关闭日志记录	Object	是
- LoggingEnabled	存储桶 logging 设置的具体信息，主要是目标存储桶	Object	否
- - TargetBucket	存放日志的目标存储桶，可以是同一个存储桶（但不推荐），或同一账户下、同一地域的存储桶	String	否
- - TargetPrefix	日志存放在目标存储桶的指定路径	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
- headers	请求返回的头部信息	Object

查询日志管理

功能说明

GET Bucket logging 用于查询源存储桶的日志配置信息。

> 只有源存储桶拥有者才可进行该请求操作。

请求示例

```
cos.getBucketLogging({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"BucketLoggingStatus": {
"LoggingEnabled": {
"TargetBucket": "targetbucket-1250000000",
"TargetPrefix": "bucket-logging-prefix/"
}
},
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询日志管理的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是

参数名称	描述	类型	是否必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- BucketLoggingStatus	说明日志记录配置的状态，如果为空字符串则意为关闭日志记录	Object/String
- - LoggingEnabled	存储桶 logging 设置的具体信息，主要是目标存储桶	Object
- - - TargetBucket	存放日志的目标存储桶，可以是同一个存储桶（但不推荐），或同一账户下、同一地域的存储桶	String
- - - TargetPrefix	日志存放在目标存储桶的指定路径	String

版权所有：腾讯云计算（北京）有限责任公司

存储桶标签

最近更新时间：2020-05-07 18:02:52

简介

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket tagging	设置存储桶标签	为已存在的存储桶设置标签
GET Bucket tagging	查询存储桶标签	查询指定存储桶下已有的存储桶标签
DELETE Bucket tagging	删除存储桶标签	删除指定的存储桶标签

设置存储桶标签

功能说明

为已存在的存储桶设置标签。

请求示例

```
cos.putBucketTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Tagging: {
Tags: [
{"Key": "k1", "Value": "v1"},
{"Key": "k2", "Value": "v2"}
]
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	设置存储桶标签的存储桶，格式：BucketName-APPID	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Tagging	标签信息	Object	是
- Tags	标签信息	ObjectArray	是
- - Key	标签名	String	是
- - Value	标签值	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询存储桶标签

功能说明

查询指定存储桶下已有的存储桶标签。

请求示例

```
cos.getBucketTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"Tags": [
{"Key": "k1", "Value": "v1"},
{"Key": "k2", "Value": "v2"}
],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	查询存储桶标签的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Tags	标签信息	ObjectArray
- - Key	标签名	String
- - Value	标签值	String

版权所有：腾讯云计算（北京）有限责任公司

删除存储桶标签

功能说明

删除指定的存储桶标签。

请求示例

```
cos.deleteBucketTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	描述	类型	是否必填
Bucket	删除存储桶标签的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

对象标签

最近更新时间：2022-10-25 14:26:56

简介

本文档提供关于对象标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Object tagging	设置对象标签	为已存在的对象设置标签
GET Object tagging	查询对象标签	查询指定对象下已有的对象标签
DELETE Object tagging	删除对象标签	删除指定的对象标签

设置对象标签

功能说明

为已存在的对象设置标签。

请求示例

```
cos.putObjectTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.png', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Tags: [
{"Key": "k1", "Value": "v1"},
{"Key": "k2", "Value": "v2"}
],
// VersionId: 'MTg0NDUwOTMyODM1MDg2MTA1xxx' /* 对象版本 ID，非必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	设置对象标签的存储桶，格式：BucketName-APPID	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
Tags	标签信息	ObjectArray	是
- Key	标签名	String	是
- Value	标签值	String	是
VersionId	当启用版本控制时，指定要操作的对象版本 ID，如不指定则将添加标签到最新版本的对象	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询对象标签

功能说明

查询指定对象下已有的对象标签。

请求示例

```
cos.getObjectTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
```

```
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.png', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"Tags": [
{"Key": "k1", "Value": "v1"},
{"Key": "k2", "Value": "v2"}
],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	查询对象标签的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- headers	请求返回的头部信息	Object
- Tags	标签信息	ObjectArray
- - Key	标签名	String
- - Value	标签值	String

删除对象标签

功能说明

删除指定的对象标签。

请求示例

```
cos.deleteObjectTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.png', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	描述	类型	是否必填
Bucket	删除对象标签的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

静态网站

最近更新时间：2020-05-07 18:02:53

简介

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket website	设置静态网站	为已存在的存储桶设置静态网站配置信息
GET Bucket website	查询静态网站	查询指定存储桶的静态网站配置信息
DELETE Bucket website	删除静态网站	删除指定存储桶的静态网站配置信息

设置静态网站

功能说明

为已存在的存储桶配置静态网站。

请求示例

```
cos.putBucketWebsite({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
WebsiteConfiguration: {
IndexDocument: {
Suffix: "index.html"
},
ErrorDocument: {
Key: "error.html"
},
RedirectAllRequestsTo: {
Protocol: "https"
},
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置静态网站的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
WebsiteConfiguration	静态网站配置，包括索引文档、错误文档、协议转换和重定向规则	Object	是
- IndexDocument	索引文档	Object	是
- - Suffix	指定索引文档	String	是
- ErrorDocument	错误文档	Object	否
- - Key	指定通用错误返回	String	否
- RedirectAllRequestsTo	重定向所有请求	Object	否
- - Protocol	指定全站重定向的协议，只能设置为 https	String	否
- RoutingRules	设置重定向规则，最多设置100条	ObjectArray	否
- - Condition	指定重定向发生的条件，前缀匹配重定向和错误码重定向只能指定一个	Object	否
- - - HttpErrorCodeReturnedEquals	指定重定向错误码，只支持配置4XX返回码，优先级高于ErrorDocument	String	否
- - - KeyPrefixEquals	指定前缀重定向的路径，替换指定的 folder/	String	否
- - Redirect	指定满足重定向 conditon 时重定向的具体替换规则	Object	否
- - - ReplaceKeyWith	替换整个 Key 为指定的内容	String	否
- - - ReplaceKeyPrefixWith	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String	否

回调函数说明

```
function(err, data) { ... }
```

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询静态网站配置

功能说明

查询与指定存储桶关联的静态网站配置信息。

请求示例

```
cos.getBucketWebsite({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"WebsiteConfiguration": {
"IndexDocument": {
"Suffix": "index.html"
},
"ErrorDocument": {
"Key": "error.html"
},
"RedirectAllRequestsTo": {
"Protocol": "https"
},
},
```

```
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询静态网站配置的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
- WebsiteConfiguration	静态网站配置，包括索引文档、错误文档、协议转换和重定向规则	Object
- - IndexDocument	索引文档	Object
- - - Suffix	指定索引文档	String
- - ErrorDocument	错误文档	Object
- - - Key	指定通用错误返回	String
- - RedirectAllRequestsTo	重定向所有请求	Object
- - - Protocol	指定全站重定向的协议，只能设置为 https	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
- - RoutingRules	设置重定向规则，最多设置100条	ObjectArray
- - - Condition	指定重定向发生的条件，前缀匹配重定向和错误码重定向只能指定一个	Object
- - - - HttpErrorCodeReturnedEquals	指定重定向错误码，只支持配置4XX 返回码，优先级高于 ErrorDocument	String
- - - - KeyPrefixEquals	指定前缀重定向的路径，替换指定的 folder/	String
- - - Redirect	指定满足重定向 conditon 时重定向的具体替换规则	Object
- - - - ReplaceKeyWith	替换整个 Key 为指定的内容	String
- - - - ReplaceKeyPrefixWith	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String

删除静态网站配置

功能说明

删除指定存储桶中的静态网站配置。

请求示例

```
cos.deleteBucketWebsite({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	被删除静态网站配置的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

版权所有：腾讯云计算（北京）有限责任公司

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

清单

最近更新时间：2020-06-10 16:46:13

简介

本文档提供关于清单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket inventory	设置清单任务	设置存储桶的清单任务
GET Bucket inventory	查询清单任务	查询存储桶的清单任务
DELETE Bucket inventory	删除清单任务	删除存储桶的清单任务

设置清单任务

功能说明

PUT Bucket inventory 用于在存储桶中创建清单任务，您可以对清单任务命名后，使用该请求创建清单任务，详细信息请参见 清单功能概述。

- COS 支持在每个存储桶中创建最多1000条清单任务。
- 您必须在目标存储桶中写入存储桶策略，以供 COS 将清单任务的结果文件写入该存储桶中。
- 调用该请求时，请确保您有足够的权限对存储桶的清单任务进行操作。存储桶所有者默认拥有该权限，若您无该项权限，请先向存储桶所有者申请该项操作的权限。

请求示例

示例1：创建清单任务，并设置服务端加密。

```
cos.putBucketInventory({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
Id: 'inventory_test', /* 必须 */
InventoryConfiguration: {
Id: 'inventory_test',
IsEnabled: 'true',
Destination: {
```

```
COSBucketDestination: {
Format: 'CSV',
AccountId: '100000000001',
Bucket: 'qcs::cos:ap-beijing::targetbucket-1250000000',
Prefix: 'inventory_test_prefix',
Encryption: {
SSECOS: ''
}
}
},
Schedule: {
Frequency: 'Daily'
},
Filter: {
Prefix: 'filter_prefix'
},
IncludedObjectVersions: 'All',
OptionalFields: [
'Size',
'LastModifiedDate',
'StorageClass',
'ETag'
]
}
}, function(err, data) {
console.log(err || data);
});
```

示例2：创建清单任务，不设置服务端加密。

```
cos.putBucketInventory({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
Id: 'inventory_test', /* 必须 */
InventoryConfiguration: {
Id: 'inventory_test',
IsEnabled: 'true',
Destination: {
COSBucketDestination: {
Format: 'CSV',
AccountId: '100000000001',
Bucket: 'qcs::cos:ap-beijing::targetbucket-1250000000',
Prefix: 'inventory_test_prefix'
}
},
Schedule: {
Frequency: 'Daily'
```

版权所有：腾讯云计算（北京）有限责任公司

```
},
Filter: {
Prefix: 'filter_prefix'
},
IncludedObjectVersions: 'All',
OptionalFields: [
'Size',
'LastModifiedDate',
'StorageClass',
'ETag'
]
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置清单任务的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Id	清单任务的名称。 缺省值：None 合法字符：`a-z，A-Z，0-9，-，_，.`	String	是
InventoryConfiguration	包含清单的配置参数	Object	是
- Id	清单任务的名称。 缺省值：None 合法字符：`a-z，A-Z，0-9，-，_，.`	String	是
- IsEnabled	清单是否启用的标识。如果设置为 true，清单功能将生效；如果设置为 false，将不生成任何清单	String	是
- IncludedObjectVersions	是否在清单中包含对象版本 如果设置为 All，清单中将会包含所有对象版本，并在清单中增加 VersionId，IsLatest，DeleteMarker 这几个字段 如果设置为 Current，则清单中不包含对象版本信息	String	是
- Filter	筛选待分析对象。清单功能将分析符合 Filter 中设置的前缀的对象	Object	否
- - Prefix	需要分析的对象的前缀	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型	是否必填
- OptionalFields	清单结果中可选包含的分析项目名称，可选字段包括：Size，LastModifiedDate，StorageClass，ETag，IsMultipartUploaded，ReplicationStatus	Array	否
- Schedule	配置清单任务周期	Object	是
- - Frequency	清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly	String	是
- Destination	描述存放清单结果的信息	Object	是
- - COSBucketDestination	清单结果导出后存放的存储桶信息	Object	是
- - - Bucket	清单分析结果的存储桶名	String	是
- - - AccountId	存储桶的所有者 ID，例如100000000001	String	否
- - - Prefix	清单分析结果的前缀。如果您指定了清单投递的前缀，COS 后端会自动在您指定的前缀后边加上'/'。如您指定了'Prefix'作为前缀，则 COS 后端投递的清单报告路径为'Prefix/inventory_report'	String	否
- - - Format	清单分析结果的文件形式，可选项为 CSV 格式	String	是
- - - Encryption	为清单结果提供服务端加密的选项	Object	否
- - - - SSECOS	COS 托管密钥的加密方式，无需填充，设置为空字符串	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
- headers	请求返回的头部信息	Object

查询清单任务

功能说明

GET Bucket inventory 用于查询存储桶中用户的清单任务信息。用户在发起该请求时，需要用户提供清单任务的名称，发起该请求时需获得请求签名，表明该请求已获得许可。有关清单的详细介绍，请参见 清单功能概述。

- 调用该请求时，请确保您有足够的权限对存储桶的清单任务进行操作。
- 存储桶所有者默认拥有该权限，若您无该项权限，请先向存储桶所有者申请该项操作的权限。

请求示例

```
cos.getBucketInventory({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
Id: 'inventory_test' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"InventoryConfiguration": {
"Id":"inventory_test",
"IsEnabled":"true",
"Destination":{
"COSBucketDestination": {
"Format":"CSV",
"AccountId":"100000000001",
"Bucket":"qcs::cos:ap-beijing::targetbucket-1250000000",
"Prefix":"inventory_test_prefix",
"Encryption": {
"SSECOS":""
}
}
},
```

```
"Schedule": {
"Frequency":"Daily"
},
"Filter": {
"Prefix":"filter_prefix"
},
"IncludedObjectVersions":"All",
"OptionalFields": [
"Size",
"LastModifiedDate",
"StorageClass",
"ETag"
]
}
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询清单任务的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Id	清单任务的名称。 缺省值：None 合法字符：`a-z，A-Z，0-9，-，_，.`	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误则为空	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- InventoryConfiguration	包含清单的配置参数	Object
- - Id	清单任务的名称。 缺省值：None 合法字符：`a-z，A-Z，0-9，-，_，.`	String
- - IsEnabled	清单是否启用的标识。如果设置为 true，清单功能将生效；如果设置为 false，将不生成任何清单	String
- - IncludedObjectVersions	是否在清单中包含对象版本 如果设置为 All，清单中将会包含所有对象版本，并在清单中增加 VersionId，IsLatest，DeleteMarker 这几个字段 如果设置为 Current，则清单中不包含对象版本信息	String
- - Filter	筛选待分析对象。清单功能将分析符合 Filter 中设置的前缀的对象	Object
- - - Prefix	需要分析的对象的前缀	String
- - OptionalFields	清单结果中可选包含的分析项目名称，可选字段包括：Size，LastModifiedDate，StorageClass，ETag，IsMultipartUploaded，ReplicationStatus	Array
- - Schedule	配置清单任务周期	Object
- - - Frequency	清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly	String
- - Destination	描述存放清单结果的信息	Object
- - - COSBucketDestination	清单结果导出后存放的存储桶信息	Object
- - - - Bucket	清单分析结果的存储桶名	String
- - - - AccountId	存储桶的所有者 ID，例如100000000001	String
- - - - Prefix	清单分析结果的前缀。如果您指定了清单投递的前缀，COS后端会自动在您指定的前缀后边加上'/'。如您指定了'Prefix'作为前缀，则COS后端投递的清单报告路径为'Prefix/inventory_report'	String
- - - - Format	清单分析结果的文件形式，可选项为 CSV 格式	String
- - - - Encryption	为清单结果提供服务端加密的选项	Object

参数名称	描述	类型
- - - - - SSECOS	COS 托管密钥的加密方式，无需填充，设置为空字符串	String

删除清单任务

功能说明

DELETE Bucket inventory 用于删除存储桶中指定的清单任务，用户需提供待删除的清单任务的名称。
有关清单的详细介绍，请参见 清单功能概述。

- 调用该请求时，请确保您有足够的权限对存储桶的清单任务进行操作。
- 存储桶所有者默认拥有该权限，如您无该项权限，请先向存储桶所有者申请该项操作的权限。

请求示例

```
cos.deleteBucketInventory({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
Id: 'inventory_test' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	被删除清单任务的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域。枚举值请参见 地域和访问域名	String	是
Id	清单任务的名称。缺省值：None 合法字符：`a-z, A-Z, 0-9, -, _, .`	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

访问管理
跨域访问

最近更新时间：2020-05-07 18:02:54

简介

本文档提供关于跨域访问的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域名访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域名访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域名访问配置信息

设置跨域配置

1. 如果要在前端修改跨域访问配置，需要该存储桶本身支持跨域，您可以在控制台进行跨域访问配置，详情请参见 快速入门 文档。
2. 在修改跨域访问配置时，请注意不要影响到当前的 Origin 下的跨域请求。

功能说明

设置存储桶的跨域名访问权限，您可以通过传入 XML 格式的配置文件来实现配置，文件大小限制为64KB。默认情况下，存储桶的持有者直接有权限使用该 API 接口，存储桶持有者也可以将权限授予其他用户。

请求示例

```
cos.putBucketCors({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
CORSRules: [{
"AllowedOrigin": ["*"],
"AllowedMethod": ["GET", "POST", "PUT", "DELETE", "HEAD"],
"AllowedHeader": ["*"],
"ExposeHeader": ["ETag", "x-cos-acl", "x-cos-version-id", "x-cos-delete-marker",
"x-cos-server-side-encryption"],
```

```
"MaxAgeSeconds": "5"
}]
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置跨域配置的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
CORSRules	说明跨域资源共享配置的所有信息列表	ObjectArray	否
- ID	配置规则的 ID	String	否
- AllowedMethods	允许的 HTTP 操作，枚举值：GET、PUT、HEAD、POST、DELETE 等	StringArray	是
- AllowedOrigins	允许的访问来源，支持通配符 `*`，格式为：`协议://域名[:端口]`，例如：`http://www.qq.com`	StringArray	是
- AllowedHeaders	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 `*`	StringArray	否
- ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	StringArray	否
- MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询跨域配置

功能说明

GET Bucket cors 接口实现查询存储桶的跨域名访问配置信息。（CORS 是一个 W3C 标准，全称是"跨域资源共享"（Cross-origin Resource Sharing））。默认情况下，存储桶的持有者直接有权限使用该 API 接口，存储桶持有者也可以将权限授予其他用户。

请求示例

```
cos.getBucketCors({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"CORSRules": [{
"MaxAgeSeconds": "5",
"AllowedOrigins": ["*"],
"AllowedHeaders": ["*"],
"AllowedMethods": ["GET", "POST", "PUT", "DELETE", "HEAD"],
"ExposeHeaders": ["ETag", "Content-Length", "x-cos-acl", "x-cos-version-id", "x-cos-request-id", "x-cos-delete-marker", "x-cos-server-side-encryption"]
}],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	查询跨域配置的存储桶，格式：BucketName-APPID	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- CORSRules	说明跨域资源共享配置的所有信息列表	ObjectArray
- - AllowedMethods	允许的 HTTP 操作，枚举值：GET、PUT、HEAD、POST、DELETE	StringArray
- - AllowedOrigins	允许的访问来源，支持通配符 `*` ，格式为： `协议://域名[:端口]` 例如： `http://www.qq.com`	StringArray
- - AllowedHeaders	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 `*`	StringArray
- - ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	StringArray
- - MaxAgeSeconds	设置 OPTIONS 跨域信息缓存秒数	String
- - ID	配置规则的 ID	String

删除跨域配置

功能说明

删除存储桶的跨域名访问配置信息。

1. 删除当前存储桶的跨域访问配置信息，会导致所有请求跨域失败，请谨慎操作。
2. 不推荐在浏览器端使用该方法。

请求示例

```
cos.deleteBucketCors({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	被删除跨域配置的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

存储桶策略

最近更新时间：2020-10-15 14:25:54

简介

本文档提供关于存储桶策略的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket policy	设置存储桶策略	设置指定存储桶的权限策略
GET Bucket policy	查询存储桶策略	查询指定存储桶的权限策略
DELETE Bucket policy	删除存储桶策略	删除指定存储桶的权限策略

设置存储桶策略

功能说明

设置指定存储桶的权限策略。

请求示例

```
cos.putBucketPolicy({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Policy: {
"version": "2.0",
"Statement": [{
"Effect": "allow",
"Principal": {
"qcs": ["qcs::cam::uin/100000000001:uin/100000000001"]
},
"Action": [
"name/cos:PutObject",
"name/cos:InitiateMultipartUpload",
"name/cos:ListMultipartUploads",
"name/cos:ListParts",
"name/cos:UploadPart",
"name/cos:CompleteMultipartUpload"
],
"Resource": ["qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"],
```

版权所有：腾讯云计算（北京）有限责任公司

```
}]
},
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是
Bucket	设置存储桶策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Policy	权限策略，详情请参见 访问管理策略语法	Object	是
- version	版本号，固定2.0	String	是
- statement	权限策略生命列表	ObjectArray	是
- - effect	效力，枚举值：allow、deny	String	是
- - principal	身份信息	ObjectArray	是
- - - qcs	身份信息标识字符串 格 式：`qcs::cam::uin/100000000001:uin/100000000011` 其中100000000001 是主账号，100000000011是子账号	String	是
- - action	策略生效的相关 Action 列表，支持通配符 `*`	StringArray	是
- - resource	相关的资源标识字符串列表 格式：`qcs::cos:<Region>:uid/<AppId>:<ShortBucketName>/*` 例如：`qcs::cos:ap-beijing:uid/1250000000:examplebucket/*`	StringArray	是
- - condition	约束条件，可以不填，具体说明请参见 condition 说明	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，	Object

	更多详情请参见 错误码 文档	
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询存储桶策略

功能说明

查询指定存储桶的权限策略。

请求示例

```
cos.getBucketPolicy({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"Policy": {
"version": "2.0",
"Statement": [{
"Action": [
"name/cos:PutObject",
"name/cos:InitiateMultipartUpload",
"name/cos:ListMultipartUploads",
"name/cos:ListParts",
"name/cos:UploadPart",
"name/cos:CompleteMultipartUpload"
],
"Effect": "allow",
"Principal": {
"qcs": ["qcs::cam::uin/100000000001:uin/100000000001"]
},
```

```
"Resource": ["qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*"],
"Sid": "costs-1539833197000000307620-46518-39"
}]
},
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询存储桶权限策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- Policy	权限策略，详情请参见 访问管理策略语法	Object
- - version	版本号，固定2.0	String
- - Statement	权限策略生命列表	ObjectArray
- - - effect	效力，枚举值：allow、deny	String
- - - principal	身份信息	ObjectArray
- - - - qcs	身份信息标识字符串 格式：`qcs::cam::uin/100000000001:uin/100000000011` 其中100000000001是主账号，100000000011是子账号	String
- - - action	策略生效的相关 Action 列表，支持通配符 `*`	StringArray
- - - resource	相关的资源标识字符串列表 格式：`qcs::cos:<Region>:uid/&ltAppId>:<ShortBucketName>/*`	StringArray

	例如：`qcs::cos:ap-beijing:uid/1250000000:examplebucket/*`	
- - - condition	约束条件，可以不填，具体说明请参见 condition 说明	ObjectArray

删除存储桶策略

功能说明

DELETE Bucket policy 请求删除指定存储桶的权限策略。

只有 Bucket 所有者有权限发起该请求。如果权限策略不存在，将返回204 No Content。

请求示例

```
cos.deleteBucketPolicy({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	被删除存储桶权限策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

自定义域名

最近更新时间：2020-05-07 18:02:56

简介

本文档提供关于自定义域名的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket domain	设置自定义域名	设置存储桶的自定义域名信息
GET Bucket domain	查询自定义域名	查询存储桶的自定义域名信息
DELETE Bucket domain	删除自定义域名	删除存储桶的自定义域名信息

设置自定义域名

功能说明

为已存在的存储桶绑定自定义域名。

请求示例

```
cos.putBucketDomain({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
DomainRule: [{
Status: "DISABLED",
Name: "www.example.com",
Type: "REST"
},
{
Status: "DISABLED",
Name: "www.example.net",
Type: "WEBSITE",
}]
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置自定义域名的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
DomainRule	自定义域名配置	Object	是
- Status	域名上线/下线状态，枚举值： ENABLED、DISABLED	String	是
- Name	用户的自定义域名	String	是
- Type	绑定的源站类型，枚举值：REST、WEBSITE	String	是
- ForcedReplacement	替换已存在的配置，枚举值：CNAME、TXT。填写则强制校验域名所有权后，再下发配置。	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询自定义域名

功能说明

查询与指定存储桶关联的自定义域名信息。

请求示例

```
cos.getBucketDomain({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"DomainRule": [{
"Status": "DISABLED",
"Name": "www.example.com",
"Type": "REST"
}, {
"Status": "DISABLED",
"Name": "www.example.net",
"Type": "WEBSITE"
}],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询自定义域名的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
- DomainRule	自定义域名配置	Object
- - Status	域名上线/下线状态，枚举值： ENABLED、DISABLED	String
- - Name	用户的自定义域名	String
- - Type	绑定的源站类型，枚举值：REST、WEBSITE	String
- - ForcedReplacement	替换已存在的配置，枚举值：CNAME、TXT。填写则强制校验域名所有权后，再下发配置。	String

删除自定义域名

功能说明

删除指定存储桶中的自定义域名配置。

请求示例

```
cos.deleteBucketDomain({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	被删除自定义域名的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

访问控制

最近更新时间：2021-01-13 19:37:14

简介

本文档提供关于存储桶、对象的访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

存储桶 ACL

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶的访问权限控制列表（ACL）
GET Bucket acl	查询存储桶 ACL	查询指定存储桶的访问权限控制列表（ACL）

对象 ACL

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表
GET Object acl	查询对象 ACL	查询对象的访问控制列表

存储桶 ACL

设置存储桶 ACL

功能说明

PUT Bucket acl 接口用来设置指定存储桶访问权限控制列表（ACL）。

使用示例

设置存储桶公有读：

```
cos.putBucketAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
ACL: 'public-read'
}, function(err, data) {
console.log(err || data);
});
```

为某个用户赋予存储桶所有权限：

```
cos.putBucketAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
GrantFullControl: 'id="qcs::cam::uin/100000000001:uin/100000000001",id="qcs::ca
m::uin/100000000011:uin/100000000011"' // 100000000001是 uin
}, function(err, data) {
console.log(err || data);
});
```

通过 AccessControlPolicy 修改存储桶权限：

```
cos.putBucketAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
AccessControlPolicy: {
"Owner": { // AccessControlPolicy 里必须有 owner
"ID": 'qcs::cam::uin/100000000001:uin/100000000001' // 100000000001 是主账号 uin
},
"Grants": [{
"Grantee": {
"ID": "qcs::cam::uin/100000000011:uin/100000000011", // 100000000011 是主账号下的子
账号 uin
},
"Permission": "WRITE"
}]
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
ACL	定义存储桶的访问控制列表（ACL）属性。枚举值请参见 ACL 概述 文档中存储桶的预设 ACL 部分，例如 private， public-read 等，默认为 private	String
GrantRead	赋予被授权者读取存储桶的权限，格式：id="[OwnerUin]"，可使用半角逗号	String

版权所有：腾讯云计算（北京）有限责任公司

	（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	
GrantWrite	赋予被授权者写入存储桶的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantReadAcp	赋予被授权者读取存储桶的访问控制列表（ACL）和存储桶策略（Policy）的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantWriteAcp	赋予被授权者写入存储桶的访问控制列表（ACL）和存储桶策略（Policy）的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantFullControl	赋予被授权者操作存储桶的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
AccessControlPolicy	跨域资源共享配置的所有信息列表	Object

版权所有：腾讯云计算（北京）有限责任公司

- Owner	存储桶持有者的信息	Object
- - ID	存储桶持有者的完整 ID，格式为 `qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]`，例如 `qcs::cam::uin/100000000001:uin/100000000001`，其中100000000001为 uin	String
- Grants	被授权者信息与权限信息列表	ObjectA
- - Permission	授予的权限信息，可选项 READ、WRITE、READ_ACP、WRITE_ACP、FULL_CONTROL。枚举值详情参见 ACL 概述 文档中存储桶的操作部分	String
- - Grantee	被授权者的信息	Object
- - - ID	被授权者的完整 ID，格式为 `qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]`，例如 `qcs::cam::uin/100000000001:uin/100000000001`，其中100000000001为 uin	String
- - - DisplayName	被授权者的名称，一般填写成和 ID 一致的字符串	String
- - - URI	预设用户组，请参见 ACL 概述 文档中预设用户组部分，例如 `http://cam.qcloud.com/groups/global/AllUsers` 或 `http://cam.qcloud.com/groups/global/AuthenticatedUsers`	String

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询存储桶 ACL

功能说明

GET Bucket acl 接口用来查询存储桶的访问控制列表（ACL）。该 API 的请求者需要对存储桶有写入 ACL 权限。

使用示例

```
cos.getBucketAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"GrantFullControl": "",
"GrantWrite": "",
"GrantRead": "",
"GrantReadAcp": "id=\"qcs::cam::uin/100000000011:uin/100000000011\"",
"GrantWriteAcp": "id=\"qcs::cam::uin/100000000011:uin/100000000011\"",
"ACL": "private",
"Owner": {
"ID": "qcs::cam::uin/100000000001:uin/100000000001",
"DisplayName": "qcs::cam::uin/100000000001:uin/100000000001"
},
"Grants": [{
"Grantee": {
"ID": "qcs::cam::uin/100000000011:uin/100000000011",
"DisplayName": "qcs::cam::uin/100000000011:uin/100000000011"
},
"Permission": "READ"
}],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是

Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- ACL	定义存储桶的访问控制列表（ACL）属性。枚举值请参见 ACL 概述 文档中存储桶的预设 ACL 部分，例如 private，public-read 等，默认为 private	String
- GrantRead	具有读取存储桶权限的被授权者 ID 信息	String
- GrantWrite	具有写入存储桶权限的被授权者 ID 信息	String
- GrantReadAcp	具有读取存储桶的访问控制列表（ACL）和存储桶策略（Policy）权限的被授权者 ID 信息	String
- GrantWriteAcp	具有写入存储桶的访问控制列表（ACL）和存储桶策略（Policy）权限的被授权者 ID 信息	String
- GrantFullControl	具有存储桶所有权限的被授权者 ID 信息	String
- Owner	存储桶持有者的信息	Object
- - DisplayName	存储桶持有者的名称	String
- - ID	存储桶持有者的完整 ID，格式为 `qcs::cam::uin/<OwnerUin>:uin/<SubUin>` 如果是主帐号，<OwnerUin> 和 <SubUin> 是同一个值	String
- Grants	被授权者信息与权限信息列表	ObjectArray
- - Permission	指明授予被授权者的权限信息，枚举值：READ、WRITE、READ_ACP、	String

	WRITE_ACP、FULL_CONTROL	
- - Grantee	被授权者的信息	Object
- - - DisplayName	被授权者的名称	String
- - - ID	被授权者的完整 ID： • 如果是主账号，格式为 `qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>` 或 `qcs::cam::anyone:anyone` （指代所有用户） • 如果是子帐号，格式为 `qcs::cam::uin/<OwnerUin>:uin/<SubUin>`	String
- - - URI	预设用户组，请参见 ACL 概述 文档中预设用户组部分，例如 `http://cam.qcloud.com/groups/global/AllUsers` 或 `http://cam.qcloud.com/groups/global/AuthenticatedUsers`	String

对象 ACL

设置对象 ACL

功能说明

PUT Object acl 接口用来设置特定存储桶中某个对象的访问控制列表（ACL）。

> ⚠ **注意：**
>
> 每个主账号（即同一个 APPID），存储桶的 ACL、Policy 和 CAM 关联的策略数量总和最多为1000条，对象访问控制列表（ACL）规则数量不限制。如果您不需要进行对象访问控制列表（ACL）控制，请不要设置，默认继承存储桶权限。

使用示例

```
cos.putObjectAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: 'exampleobject', /* 必须 */
ACL: 'public-read', /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

为某个用户赋予对象的所有权限：

```
cos.putObjectAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: 'exampleobject', /* 必须 */
GrantFullControl: 'id="100000000001"' // 100000000001是主账号 uin
}, function(err, data) {
console.log(err || data);
});
```

通过 AccessControlPolicy 赋予对象写权限：

```
cos.putObjectAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: 'exampleobject', /* 必须 */
AccessControlPolicy: {
"Owner": { // AccessControlPolicy 里必须有 owner
"ID": 'qcs::cam::uin/100000000001:uin/100000000001' // 100000000001 是主账号 uin
},
"Grants": [{
"Grantee": {
"ID": "qcs::cam::uin/100000000011:uin/100000000011", // 100000000011 是主账号下的子账号 uin
},
"Permission": "WRITE"
}]
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，例如 default，private，public-read 等	String

	注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限	
GrantRead	赋予被授权者读取对象的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
AccessControlPolicy	设置对象的访问控制列表（ACL）属性信息	Object
- Owner	对象持有者的信息	Object
- - ID	对象持有者 ID，格式：`qcs::cam::uin/<OwnerUin>:uin/<SubUin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String
- - DisplayName	对象持有者的名称	String
- Grants	被授权者信息与权限信息列表	ObjectArray
- - Permission	指明授予被授权者的权限信息，枚举值：READ、WRITE、READ_ACP、WRITE_ACP、FULL_CONTROL	String
- - Grantee	被授权者的信息	Object
- - - DisplayName	被授权者的名称	String
- - - ID	被授权者的 ID，格式：`qcs::cam::uin/<OwnerUin>:uin/<SubUin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，204，403，404等	Number
- headers	请求返回的头部信息	Object

查询对象 ACL

功能说明

GET Object acl 接口用来查询某个存储桶下的某个对象的访问权限。只有存储桶的持有者才有权限操作。

使用示例

```
cos.getObjectAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: 'exampleobject', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象	String	是

	概述		

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- ACL	对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，例如 default，private，public-read 等	String
- Owner	标识资源的所有者	Object
- - ID	对象持有者 ID，格式为：`qcs::cam::uin/<OwnerUin>:uin/<SubUin>` 如果是主账号 ID，<OwnerUin> 和 <SubUin> 是同一个值	String
- - DisplayName	对象持有者的名称	String
- Grants	被授权者信息与权限信息列表	ObjectArray
- - Permission	指明授予被授权者的权限信息，枚举值：READ、READ_ACP、WRITE_ACP、FULL_CONTROL	String
- - Grantee	被授权者的信息	Object
- - - DisplayName	用户的名称	String
- - - ID	用户的 ID，格式为：`qcs::cam::uin/<OwnerUin>:uin/<SubUin>` 如果是主账号 ID，<OwnerUin> 和 <SubUin> 是同一个值	String

版权所有：腾讯云计算（北京）有限责任公司

防盗链

最近更新时间：2022-01-10 12:34:44

简介

本文档提供关于存储桶 Referer 白名单或者黑名单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket referer	设置存储桶 Referer	设置存储桶 Referer 白名单或者黑名单
GET Bucket referer	查询存储桶 Referer	查询存储桶 Referer 白名单或者黑名单

设置存储桶 Referer

功能说明

设置指定存储桶的 Referer 白名单或者黑名单（PUT Bucket referer）。

请求示例

```
cos.putBucketReferer({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
RefererConfiguration: {
Status: 'Enabled',
RefererType: 'White-List',
DomainList: {
Domains: [
'*.qq.com',
'*.qcloud.com',
]
},
EmptyReferConfiguration: 'Allow',
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置存储桶策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
RefererConfiguration	防盗链配置信息，详情请参见 PUT Bucket referer	Object	是
- Status	是否开启防盗链，枚举值：Enabled、Disabled	String	是
- RefererType	防盗链类型，枚举值：Black-List、White-List	String	是
- DomainList	生效域名列表， 支持多个域名且为前缀匹配， 支持带端口的域名和 IP， 支持通配符*，做二级域名或多级域名的通配	Object	是
- - Domains	生效域名,支持两种写法：单条'*.qq.com'或多条['.qq.com', '*.qcloud.com']	String\Array	是
- EmptyReferConfiguration	是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询存储桶 Referer

功能说明

查询指定存储桶 Referer 白名单或者黑名单（GET Bucket referer）。

请求示例

```
cos.getBucketReferer({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"RefererConfiguration": {
"Status": "Enabled",
"RefererType": "White-List",
"DomainList": {
"Domains": [
"*.qq.com",
"*.qcloud.com"
]
},
"EmptyReferConfiguration": "Allow"
},
"statusCode": 200,
"headers": {},
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询存储桶权限策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- RefererConfiguration	防盗链配置信息，详情请参见 GET Bucket referer	Object
- - Status	是否开启防盗链，枚举值：Enabled、Disabled	String
- - RefererType	防盗链类型，枚举值：Black-List、White-List	String
- - DomainList	生效域名列表， 支持多个域名且为前缀匹配， 支持带端口的域名和IP， 支持通配符*，做二级域名或多级域名的通配	Object
- - - Domains	生效域名	Array
- - EmptyReferConfiguration	是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny	String

版权所有：腾讯云计算（北京）有限责任公司

数据校验
CRC64 校验

最近更新时间：2023-11-27 11:02:01

简介

数据在客户端和服务器间传输时可能会出现错误，COS 除了可以通过 MD5 和自定义属性 验证数据完整性外，还可以通过 CRC64 检验码来进行数据校验。

COS 会对新上传的对象进行 CRC64 计算，并将结果作为对象的属性进行存储，随后在返回的响应头部中携带 x-cos-hash-crc64ecma，该头部表示上传对象的 CRC64 值，根据 ECMA-182标准 计算得到。对于 CRC64 特性上线前就已经存在于 COS 的对象，COS 不会对其计算 CRC64 值，所以获取此类对象时不会返回其 CRC64 值。

操作说明

目前支持 CRC64 的 API 如下：

- 简单上传接口
  - PUT Object 和 POST Object：用户可在返回的响应头中获得文件 CRC64 校验值。
- 分块上传接口
  - Upload Part：用户可以根据 COS 返回的 CRC64 值与本地计算的数值进行比较验证。
  - Complete Multipart Upload：如果每个分块都有 CRC64 属性，则会返回整个对象的 CRC64 值，如果某些分块不具备 CRC64 值，则不返回。
- 执行 Upload Part - Copy 时，会返回对应的 CRC64 值。
- 执行 PUT Object - Copy 时，如果源对象存在 CRC64 值，则返回 CRC64，否则不返回。
- 执行 HEAD Object 和 GET Object 时，如果对象存在 CRC64，则返回。用户可以根据 COS 返回的 CRC64 值和本地计算的 CRC64 进行比较验证。

SDK 示例

功能说明

用于在对象上传和下载的时候对对象数据做 CRC64 一致性校验。

请求示例

这里只用简单上传举例，其它接口也是同样使用方式。

版权所有：腾讯云计算（北京）有限责任公司

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'exampleobject', /* 必须 */
StorageClass: 'STANDARD',
Body: fileObject, // 上传文件对象
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
if (err) {
console.log(err);
} else {
// 需要给Expose-Headers添加字段x-cos-hash-crc64ecma字段才可正确返回
// 参考文档：https://cloud.tencent.com/document/product/436/13318
var crc64 = data.headers['x-cos-hash-crc64ecma'];
console.log(crc64);
}
});
```

版权所有：腾讯云计算（北京）有限责任公司

设置访问域名（CDN/全球加速）

最近更新时间：2022-01-06 16:16:28

简介

本文档提供关于如何使用非默认域名请求 COS 服务。

相关参数说明

通过初始化参数，来控制请求域名，相关的参数说明如下，更多初始化参数说明请查看 配置项。

参数名	参数描述	类型	是否必填
Domain	调用操作存储桶和对象的 API 时自定义请求域名。可以使用模版，如 `"{Bucket}.cos.{Region}.myqcloud.com"`，即在调用 API 时会使用参数中传入的 Bucket 和 Region 进行替换。	String	否
Protocol	发请求时用的协议，可选项 `https:`、`http:`，默认判断当前页面是 `http:` 时使用 `http:`，否则使用 `https:`	String	否

默认 CDN 加速域名

关于如何开启默认加速域名请参考 开启默认 CDN 加速域名。

以下代码展示了如何使用默认加速域名访问 COS 服务。

```
var cos = new COS({
Domain: '{Bucket}.file.myqcloud.com', // 自定义加速域名，Domain 参数支持模板，这个例子里 {Bucket} 会在请求时按传入的 Bucket 自动替换
Protocol: 'https:', // 请求协议： 'https:' 或 'http:'
});
```

> 说明：
> 更多完整示例，请前往 GitHub 查看。

自定义 CDN 加速域名

关于如何开启 CDN 自定义加速域名请参考 开启自定义 CDN 加速域名。

以下代码展示了如何使用自定义加速域名访问 COS 服务。

版权所有：腾讯云计算（北京）有限责任公司

```
var cos = new COS({
Domain: 'example-cdn-domain.com', // 自定义加速域名
Protocol: 'https:', // 请求协议： 'https:' 或 'http:'
});
```

说明：
更多完整示例，请前往 GitHub 查看。

自定义源站域名

关于如何设置自定义源站域名请参考 自定义源站域名。

以下代码展示了如何使用自定义源站域名访问 COS 服务。

```
var cos = new COS({
Domain: 'example-cos-domain.com', // 自定义源站域名
Protocol: 'https:', // 请求协议： 'https:' 或 'http:'
});
```

说明：
更多完整示例，请前往 GitHub 查看。

全球加速域名

关于全球加速功能请参考 全球加速功能概述。

以下代码展示了如何使用全球加速域名访问 COS 服务。

```
var cos = new COS({
UseAccelerate: true, // 指定 true，使用全球加速域名请求
Protocol: 'https:', // 请求协议： 'https:' 或 'http:'
});
```

说明：
更多完整示例，请前往 GitHub 查看。

版权所有：腾讯云计算（北京）有限责任公司

图片处理
基础图片处理

最近更新时间：2022-10-09 16:15:31

简介

本文档提供关于基础图片处理的相关的 API 概览以及 SDK 示例代码。

服务	功能	说明
基础图片处理服务	缩放	等比缩放、设定目标宽高缩放等多种方式
	裁剪	普通裁剪、缩放裁剪、内切圆、人脸智能裁剪
	旋转	自适应旋转、普通旋转
	格式转换	格式转换、GIF 格式优化、渐进显示
	质量变换	针对 JPG 和 WEBP 图片进行质量变换
	高斯模糊	对图片进行模糊处理
	锐化	对图片进行锐化处理
	添加水印	图片水印、文字水印
	获取图片信息	基本信息、EXIF 信息、主色调
	去除元信息	包括 EXIF 信息
	快速缩略模板	快速实现图片格式转换、缩略、剪裁等功能，生成缩略图

上传时使用图片处理

下面示例展示了如何在上传图片时自动实现图片处理。

图片上传完成后，COS 会存储原始图片和已处理过的图片。后续用户可以通过普通的下载请求获取处理结果。

示例代码

版权所有：腾讯云计算（北京）有限责任公司

```
const filePath = "temp-file-to-upload" // 本地文件路径
cos.putObject({
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleobject',
Body: fs.createReadStream(filePath), // 上传文件对象
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
},
Headers: {
// 通过 imageMogr2 接口使用图片缩放功能：指定图片宽度为 200，宽度等比压缩
'Pic-Operations': '{"is_pic_info": 1, "rules": [{"fileid": "desample_photo.jpg",
"rule": "imageMogr2/thumbnail/200x/"}]}'
},
}, function(err, data) {
console.log(err || data);
});
```

对云上数据进行图片处理

下面示例展示了如何在对已存储在 COS 的图片进行相应处理操作，并将结果存入到 COS。

示例代码

```
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Key: 'exampleobject',
Method: 'POST',
Action: 'image_process',
Headers: {
// 通过 imageMogr2 接口使用图片缩放功能：指定图片宽度为 200，宽度等比压缩
'Pic-Operations': '{"is_pic_info": 1, "rules": [{"fileid": "desample_photo.jpg",
"rule": "imageMogr2/thumbnail/200x/"}]}'
},
}, function (err, data) {
console.log(err || data);
});
```

下载时使用图片处理

下面示例展示了如何在下载图片时实现图片处理。

示例代码

```
cos.getObject({
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleobject',
QueryString: `imageMogr2/thumbnail/200x/`,
}, function (err, data) {
console.log(err || data);
fs.writeFileSync('filepath', data.Body); // 将图片内容保存本地
});
```

生成带图片处理参数的签名 URL

示例代码

```
// 生成带图片处理参数的文件签名 URL，过期时间设置为 30 分钟。
var url1 = cos.getObjectUrl({
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleobject',
Query: {
`imageMogr2/thumbnail/200x/`: ''
},
Expires: 1800,
Sign: true,
}, function (err, data) {
if(data){
console.log(data.Url);
}
});
// 生成带图片处理参数的文件 URL，不带签名。
var url2 = cos.getObjectUrl({
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleobject',
Query: {
`imageMogr2/thumbnail/200x/`: ''
},
Sign: false,
}, function (err, data) {
if(data){
```

```
console.log(data.Url);
}
});
```

版权所有：腾讯云计算（北京）有限责任公司

盲水印

最近更新时间：2022-10-09 16:15:31

简介

本文档提供关于盲水印的相关的 API 概览以及 SDK 示例代码。

关于盲水印 API 文档请参见 盲水印。

添加盲水印

功能说明

盲水印支持在上传时添加和下载时添加。

示例代码一：上传时添加盲水印

```
const filePath = "temp-file-to-upload" // 本地文件路径
cos.putObject({
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleobject',
Body: fs.createReadStream(filePath), // 上传文件对象
Headers: {
// 万象持久化接口，上传时持久化
'Pic-Operations': '{"is_pic_info": 1, "rules": [{"fileid": "desample_photo.jpg",
"rule": "watermark/3/type/1/image/aHR0cDovL2V4YW1wbGVzLTEyNTEwMDAwMDQucGljc2gubXl
xY2xvdWQuY29tL3NodWl5aW4uanBn"}]}'
},
}, function(err, data) {
console.log(err || data);
});
```

示例代码二：下载时添加盲水印

```
cos.getObject({
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleobject',
QueryString: 'watermark/3/type/1/image/aHR0cDovL2V4YW1wbGVzLTEyNTEwMDAwMDQucGljc2
gubXlxY2xvdWQuY29tL3NodWl5aW4uanBn',
```

```
}, function (err, data) {
console.log(err || data);
fs.writeFileSync('filepath', data.Body); // 将图片内容保存本地
});
```

版权所有：腾讯云计算（北京）有限责任公司

异常处理

最近更新时间：2022-01-06 15:49:12

简介

调用 SDK 接口请求 COS 服务失败时，将在回调函数里返回错误信息。

错误处理示例

```
cos.headBucket({
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
}, function(err, data) {
if (err) {
console.log(err.error);
}
});
```

客户端异常

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- error	请求报错信息	Object/String

服务端异常

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

- headers	请求返回的头部信息	Object
- error	请求报错信息	Object/String
- - Code	请求失败时 body 返回的 Error Code，更多详情请参见 错误码 文档	String
- - Message	请求失败时 body 返回的 Error Message，更多详情请参见 错误码 文档	String
- - RequestId	服务器请求日志里的唯一 ID，可用于遇到问题 联系我们 排查错误原因	String

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。
   COS 自助诊断工具
   点击自助诊断
   输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**。
4. 稍侯片刻，便能看到相应的智能诊断结果。

版权所有：腾讯云计算（北京）有限责任公司

内容审核
视频审核

最近更新时间：2022-12-20 14:25:36

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于视频审核的 API 概览和 SDK 示例代码。

> 注意：
> COS Node.js SDK 版本需要大于等于 v2.11.2。

API	操作描述
提交视频审核任务	用于提交一个视频审核任务
查询视频审核任务结果	用于查询指定的视频审核任务

提交视频审核任务

功能说明

用于提交一个视频审核任务。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function postVideoAuditing() {
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/video/auditing';
var body = COS.util.json2xml({
Request: {
Input: {
Object: '1.mp4', /* 需要审核的视频文件，存储桶里的路径 */
},
Conf: {
BizType: '',
DetectType: 'Porn',
Snapshot: {
Count: 1000, // 视频截帧数量
},
DetectContent: 1, // 是否审核视频声音，0-只审核视频不审核声音；1-审核视频+声音
}
}
});
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'POST',
Url: url,
Key: '/video/auditing', /** 固定值，必须 */
ContentType: 'application/xml', /** 固定值，必须 */
Body: body
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的视频。	Container	是
Conf	Request	审核规则配置。	Container	是

版权所有：腾讯云计算（北京）有限责任公司

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	当前 COS 存储桶中的视频文件名称，例如在目录 test 中的文件 video.mp4，则文件名称为 test/video.mp4。	String	是
Url	Request.Input	视频文件的链接地址，例如 http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/test.mp4。Object 和 Url 只能选择其中一种。	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。	String	否
Snapshot	Request.Conf	视频画面的审核通过视频截帧能力截取出一定量的截图，通过对截图逐一审核而实现的，该参数用于指定视频截帧的配置。	Container	是
Callback	Request.Conf	回调地址，以 `http://` 或者 `https://` 开头的地址。	String	否
CallbackVersion	Request.Conf	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。	String	否
BizType	Request.Conf	审核策略，不带审核策略时使用默认策略。	String	否
DetectContent	Request.Conf	用于指定是否审核视频声音，当值为0时：表示只审核视频画面截图；值为1时：表示同时审核视频画面截图和视频声音。默认值为0。	Integer	否

Container 类型 Snapshot 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Mode	Request.Conf.Snapshot	截帧模式。Interval 表示间隔模式；Average 表示平均模式；Fps 表示固定帧率模式。 • Interval 模式：TimeInterval，Count 参数生效。当设置 Count，未设置 TimeInterval 时，表示截取所有帧，共 Count 张图片。 • Average 模式：Count 参数生效。表示整个视频，按平均间隔截取共 Count 张图片。 • Fps 模式：TimeInterval 表示每秒截取多少帧，Count 表示共截取多少帧。	String	否
Count	Request.Conf.Snapshot	视频截帧数量，范围为(0, 10000]。	String	否
TimeInterval	Request.Conf.Snapshot	视频截帧频率，范围为(0, 60]，单位为秒，支持 float 格式，执行精度精确到毫秒。	Float	否

返回结果说明

详情请参见 提交视频审核任务。

查询视频审核任务结果

功能说明

用于查询视频审核任务执行情况和结果。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function getVideoAuditingResult() {
var jobId = 'av14d9ca15af3a11eca0d6525400d88xxx'; // jobId可以通过提交视频审核任务返回
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/video/auditing/' + jobId;
cos.request({
```

```
Bucket: config.Bucket,
Region: config.Region,
Method: 'GET',
Key: '/video/auditing/' + jobId,
Url: url,
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID。	String	是

返回结果说明

详情请参见 查询视频审核任务结果。

PHP SDK
快速入门

最近更新时间：2023-02-23 16:39:52

下载与安装

相关资源

- 对象存储 COS 的 XML PHP SDK 源码下载地址：XML PHP SDK。
- SDK 快速下载地址：XML PHP SDK。
- 示例 Demo 程序地址：PHP sample。
- SDK 文档中的所有示例代码请参见 SDK 代码示例。
- SDK 更新日志请参见 ChangeLog。
- SDK 常见问题请参见：PHP SDK 常见问题。

> 说明：
> 如果您在使用 XML 版本 SDK 时遇到函数或方法不存在等错误，请先将 XML 版本 SDK 升级到最新版再重试。

环境依赖

- PHP 5.6+
  您可以通过 `php -v` 命令查看当前的 PHP 版本。

  > 注意：
  > 如果您的 PHP 版本 `>=5.3` 且 `<5.6`，请使用 v1.3 版本。

- cURL 扩展
- xml 扩展
- dom 扩展

- mbstring 扩展
- json 扩展
  您可以通过 `php -m` 命令查看以上扩展是否已经安装好。
- Ubuntu 系统中，您可以使用 apt-get 包管理器安装 PHP 的 相关 扩展，安装命令如下：

```
sudo apt-get install php-curl php-xml php-dom php-mbstring php-json
```

- CentOS 系统中，您可以使用 yum 包管理器安装 PHP 的 cURL 扩展。

```
sudo yum install php-curl php-xml php-dom php-mbstring php-json
```

安装 SDK

SDK 安装有三种方式：Composer 方式、Phar 方式 和 源码方式。

Composer 方式

推荐使用 Composer 安装 cos-php-sdk-v5，Composer 是 PHP 的依赖管理工具，允许您声明项目所需的依赖，然后自动将它们安装到您的项目中。

> 说明：
> 您可以在 Composer 官网 上找到更多关于如何安装 Composer，配置自动加载以及用于定义依赖项的其他最佳实践等相关信息。

安装步骤

1. 打开终端。
2. 执行以下命令，下载 Composer。

```
curl -sS https://getcomposer.org/installer | php
```

3. 创建一个名为 `composer.json` 的文件，内容如下：

```
{
"require": {
```

```
"qcloud/cos-sdk-v5": ">=2.0"
}
}
```

4. 执行以下命令，使用 Composer 安装。

```
php composer.phar install
```

使用该命令后会在当前目录中创建一个 vendor 文件夹，里面包含 SDK 的依赖库和一个 autoload.php 脚本，方便在项目中调用。

> 注意：
> 目前已支持 Composer 根据当前 PHP 版本下载 guzzle6 或 guzzle7。guzzle7 版本支持 laravel8 框架。当 PHP 版本 `>= 7.2.5` 时自动下载 guzzle7 版本，反之下载 guzzle6 版本。

5. 通过 autoloader 脚本调用 cos-php-sdk-v5，在代码中引入 autoload.php 文件:

```
require '/path/to/sdk/vendor/autoload.php';
```

> 注意：
> 执行 composer 安装后，这里的路径需更改为 autoload.php 文件所对应的路径，否则会调用不到相关方法。
> 如您安装的路径为/Users/username/project，则项目中引用的路径应填写为/Users/username/project/vendor/autoload.php。

至此，您的项目已经可以使用 COS XML PHP SDK 了。

Phar 方式

Phar 方式安装 SDK 的步骤如下：

1. 在 GitHub 发布页面 下载相应的 phar 文件。

> 说明：
> - 对于 PHP 版本 `>= 5.6` 且 `<7.2.5` ，请下载 `cos-sdk-v5-6.phar` ，以使用 Guzzle6 版本。

- 对于 PHP 版本 `>=7.2.5` 的请下载 `cos-sdk-v5-7.phar`，以使用 Guzzle7 版本。

2. 在代码中引入 phar 文件：

```
require '/path/to/cos-sdk-v5-x.phar';
```

源码方式

源码方式安装 SDK 的步骤如下：

1. 在 SDK 下载页面 下载 `cos-sdk-v5.tar.gz` 压缩文件。

说明：

- 对于 PHP 版本 `>= 5.6` 且 `<7.2.5`，请下载 `cos-sdk-v5-6.tar.gz`，以使用 Guzzle6 版本。
- 对于 PHP 版本 `>=7.2.5` 的请下载 `cos-sdk-v5-7.tar.gz` ,以使用 Guzzle7 版本。

2. 解压后通过 `autoload.php` 脚本加载 SDK，在代码中引入 autoload.php 文件:

```
require '/path/to/sdk/vendor/autoload.php';
```

注意：
`Source code` 压缩包为 Github 默认打包的代码包，里面不包含 `vendor` 目录。请注意下载release包（cos-sdk-v5-x.tar.gz包）而不是Source包，也不要直接 clone 整个仓库。否则会缺失index.php以及vendor包。

开始使用

下面为您介绍如何使用 COS PHP SDK 完成一个基础操作，如初始化客户端、创建存储桶、查询存储桶列表、上传对象、查询对象列表、下载对象和删除对象。关于示例中的参数说明，请参见 存储桶操作 和 对象操作 文档。

初始化

注意：

- 建议用户使用子账号密钥 + 环境变量的方式调用 SDK，提高 SDK 使用的安全性。为子账号授权时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

若您使用 临时密钥 初始化，请用下面方式创建实例：

```
$tmpSecretId = "TmpSecretId"; //临时密钥的 SecretId，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
$tmpSecretKey = "TmpSecretKey"; //临时密钥的 SecretKey，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
$tmpToken = "TmpToken"; //临时密钥的 Token，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $tmpSecretId,
'secretKey' => $tmpSecretKey,
'token' => $tmpToken)));
```

若您使用永久密钥初始化 COSClient，可以先在访问管理控制台中的 API 密钥管理页面 获取 SecretId、SecretKey，使用永久密钥适用于大部分的应用场景。

```
// SECRETID 和 SECRETKEY 请登录访问管理控制台进行查看和管理
$secretId = getenv('COS_SECRET_ID'); //用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考https://intl.cloud.tencent.com/document/product/598/32675
$secretKey = getenv('COS_SECRET_KEY'); //用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参考https://intl.cloud.tencent.com/document/product/598/32675
$region = "ap-beijing"; //用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

版权所有：腾讯云计算（北京）有限责任公司

注意：
若未配置 HTTPS 证书，则需要删除 schema 参数或填入 `'schema' => 'http'` ；若填入 https 会出现 certificate problem。若您需要配置证书，可参考 PHP SDK 常见问题。

创建存储桶

```
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$result = $cosClient->createBucket(array('Bucket' => $bucket));
//请求成功
print_r($result);
} catch (\Exception $e) {
//请求失败
echo($e);
}
```

查询存储桶列表

```
try {
//请求成功
$result = $cosClient->listBuckets();
print_r($result);
} catch (\Exception $e) {
//请求失败
echo($e);
}
```

上传对象

注意：

- 使用 putObject 接口上传文件（最大5G）。
- 使用 Upload 接口分块上传文件，Upload 接口为复合上传接口，对小文件进行简单上传，对大文件进行分块上传。
- 参数说明可参见 对象操作 文档。

```
# 上传文件
## putObject(上传接口，最大支持上传5G文件)
### 上传内存中的字符串
```

```
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$key = "exampleobject"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
$result = $cosClient->putObject(array(
'Bucket' => $bucket,
'Key' => $key,
'Body' => 'Hello World!'));
print_r($result);
} catch (\Exception $e) {
echo "$e\n";
}

### 上传文件流
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$key = "exampleobject"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
$srcPath = "path/to/localFile";//本地文件绝对路径
$file = fopen($srcPath, "rb");
if ($file) {
$result = $cosClient->putObject(array(
'Bucket' => $bucket,
'Key' => $key,
'Body' => $file));
print_r($result);
}
} catch (\Exception $e) {
echo "$e\n";
}

## Upload(高级上传接口，默认使用分块上传最大支持50T)
### 上传内存中的字符串
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$key = "exampleobject"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
$result = $cosClient->Upload(
$bucket = $bucket,
$key = $key,
$body = 'Hello World!');
print_r($result);
} catch (\Exception $e) {
echo "$e\n";
}

### 上传文件流
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$key = "exampleobject"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
```

```
$srcPath = "path/to/localFile";//本地文件绝对路径
$file = fopen($srcPath, 'rb');
if ($file) {
$result = $cosClient->Upload(
$bucket = $bucket,
$key = $key,
$body = $file);
}
print_r($result);
} catch (\Exception $e) {
echo "$e\n";
}
```

查询对象列表

```
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$result = $cosClient->listObjects(array(
'Bucket' => $bucket
));
// 请求成功
if (isset($result['Contents'])) {
foreach ($result['Contents'] as $rt) {
print_r($rt);
}
}
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

单次调用 `listObjects` 接口一次只能查询1000个对象，如需要查询所有的对象，则需要循环调用。

```
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$prefix = ''; //列出对象的前缀
$marker = ''; //上次列出对象的断点
while (true) {
$result = $cosClient->listObjects(array(
'Bucket' => $bucket,
'Marker' => $marker,
'MaxKeys' => 1000 //设置单次查询打印的最大数量，最大为1000
));
if (isset($result['Contents'])) {
foreach ($result['Contents'] as $rt) {
// 打印key
```

版权所有：腾讯云计算（北京）有限责任公司

```
echo($rt['Key'] . "\n");
}
}
$marker = $result['NextMarker']; //设置新的断点
if (!$result['IsTruncated']) {
break; //判断是否已经查询完
}
}
} catch (\Exception $e) {
echo($e);
}
```

下载对象

- 使用 getObject 接口下载文件。
- 使用 getObjectUrl 接口获取文件下载 URL。

```
# 下载文件
## getObject(下载文件)
### 下载到内存
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
$result = $cosClient->getObject(array(
'Bucket' => $bucket,
'Key' => $key));
// 请求成功
echo($result['Body']);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}

### 下载到本地
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
$localPath = @"path/to/localFile";//下载到本地指定路径
$result = $cosClient->getObject(array(
'Bucket' => $bucket,
'Key' => $key,
'SaveAs' => $localPath));
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
```

版权所有：腾讯云计算（北京）有限责任公司

```
}

### 指定下载范围
/*
* Range 字段格式为 'bytes=a-b'
*/
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
$localPath = @"path/to/localFile";//下载到本地指定路径
$result = $cosClient->getObject(array(
'Bucket' => $bucket,
'Key' => $key,
'Range' => 'bytes=0-10',
'SaveAs' => $localPath));
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}

## getObjectUrl(获取文件 UrL)
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
$signedUrl = $cosClient->getObjectUrl($bucket, $key, '+10 minutes');
// 请求成功
echo $signedUrl;
} catch (\Exception $e) {
// 请求失败
print_r($e);
}
```

删除对象

```
# 删除 object
## deleteObject
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
$result = $cosClient->deleteObject(array(
'Bucket' => $bucket,
'Key' => $key,
'VersionId' => 'string'
));
// 请求成功
print_r($result);
```

版权所有：腾讯云计算（北京）有限责任公司

```
} catch (\Exception $e) {
// 请求失败
echo($e);
}
# 删除多个 object
## deleteObjects
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key1 = "exampleobject1"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
$key2 = "exampleobject2"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
$result = $cosClient->deleteObjects(array(
'Bucket' => $bucket,
'Objects' => array(
array(
'Key' => $key1,
),
array(
'Key' => $key2,
),
//...
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

PHP SDK 常见问题

最近更新时间：2023-01-30 16:15:54

PHP SDK 运行时抛出 `Call to undefined method GuzzleHttp\Utils::chooseHandler()` 异常，该如何处理？

PHP SDK 依赖 Guzzle，推荐用户使用 Composer 方式安装 SDK。

- 在使用 Composer 方式安装时，需要执行 `php composer.phar install` 命令安装 SDK 和依赖。
- 在使用源码方式安装时，需要执行 `composer install` 命令安装 SDK 和依赖，详情请参阅 PHP SDK - 下载与安装。

COS 接入 PHP SDK 上传文件报错：`cURL error 60: See http://curl.haxx.se/libcurl/c/libcurl-errors.html`，该如何处理？

如果您的 PHP 环境证书有问题，可能会遇到报错，类似于 `cURL error 60: See http://curl.haxx.se/libcurl/c/libcurl-errors.html`，请尝试按以下步骤解决：

1. 访问地址 `https://curl.haxx.se/ca/cacert.pem` 下载证书文件 cacert.pem，将其保存到 PHP 安装路径下。
2. 编辑 php.ini 文件，删除 curl.cainfo 配置项前的分号注释符（;），值设置为保存的证书文件 cacert.pem 的绝对路径。
3. 重启依赖 PHP 的服务。

版权所有：腾讯云计算（北京）有限责任公司

对象操作
上传对象

最近更新时间：2022-05-19 14:24:27

简介

本文档提供关于对象的上传相关的 API 概览以及 SDK 示例代码。

简单上传

API	操作名	操作描述
PUT Object	简单上传对象	上传一个对象至存储桶
PUT Object - Copy	设置对象复制（修改对象属性）	复制文件到目标路径
APPEND Object	追加上传对象	将对象以分块追加的方式上传至存储桶

分块上传

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传任务
Upload Part	上传分块	分块上传对象
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个对象的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

高级接口（推荐）

该小节主要讲述由 COS 提供的封装了上传和复制操作的高级接口，用户只需要设置相应的参数，该接口内部会根据文件大小决定是进行简单上传（复制）还是分块上传（复制），使用接口前请确认已完成了 快速入门 中指引的初始

版权所有：腾讯云计算（北京）有限责任公司

化步骤。

上传对象（断点续传）

方法原型

```
public Qcloud\Cos\Client upload(string $bucket, string $key, $body, array $options = array());
```

功能说明

该接口内部会根据文件大小，对小文件调用简单上传接口，对大文件调用分块上传接口。接口参数可参照 `PUT Object` 和 `Upload Part` 接口。

参数说明

参数名称	类型	描述	是否必填
bucket	String	存储桶名称，格式：BucketName-APPID	是
key	String	对象键	是
body	Stream/String	上传的内容	是
options	Array	附加的配置项	否

options 参数	类型	描述	是否必填
Progress	Function	进度条回调，参数为总大小($totalSize), 已上传大小($uploadedSize)	否
PartSize	Int	最小分块文件大小，默认为5M	否
Concurrency	Int	并发度，默认为10	否
ACL	String	设置对象的 ACL，例如 private、public-read	否
CacheControl	String	缓存策略，设置 Cache-Control	否
ContentDisposition	String	文件名称，设置 Content-Disposition	否
ContentEncoding	String	编码格式，设置 Content-Encoding	否
ContentLanguage	String	语言类型，设置 Content-Language	否
ContentLength	Int	设置传输长度	否

版权所有：腾讯云计算（北京）有限责任公司

options 参数	类型	描述	是否必填
ContentType	String	内容类型，设置 Content-Type	否
Expires	String	设置 Content-Expires	否
Metadata	Array	用户自定义的文件元信息	否
StorageClass	String	文件的存储类型，例如 STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD。更多存储类型，请参见 存储类型概述	否
ContentMD5	Boolean	是否自动生成上传文件的 MD5 值用于校验	否
ServerSideEncryption	String	服务端加密方法	否

请求示例

示例一：上传本地对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
try {
$result = $cosClient->upload(
$bucket = 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
$key = 'exampleobject', //此处的 key 为对象键
$body = fopen($local_path, 'rb')
);
// 请求成功
print_r($result);
} catch (\Exception $e) {
```

```
// 请求失败
echo($e);
}
```

示例二：上传归档对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
try {
$result = $cosClient->upload(
$bucket = 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
$key = 'exampleobject', //此处的 key 为对象键
$body = fopen($local_path, 'rb'),
$options = array(
'StorageClass' => 'Archive'
)
);
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例三：指定单个分块大小上传带有 meta 的对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
```

```
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
try {
$result = $cosClient->upload(
$bucket = 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
$key = 'exampleobject', //此处的 key 为对象键
$body = fopen($local_path, 'rb'),
$options = array(
'Metadata' => array(
'string' => 'string',
),
'PartSize' => 10 * 1024 * 1024
)
);
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例四：断点续传对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
```

```
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
try {
//生成一个随机字符串作为上传的body
$characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
$randomString = '';
for ($i = 0; $i < 10; $i++) {
$randomString .= $characters[rand(0, strlen($characters) - 1)];
}
//块大小
$partSize = 1024 * 1024 + 1;
//初始化分块上传
$rt = $this->cosClient->CreateMultipartUpload(array(
'Bucket' => $this->bucket,
'Key' => $this->key
));
//获取上传ID
$uploadId = $rt['UploadId'];
//上传分块
$this->cosClient->uploadPart(array(
'Bucket' => $this->bucket,
'Key' => $this->key,
'Body' => substr($body, 0, $partSize),
'UploadId' => $uploadId,
'PartNumber' => 1
));
//断点续传
$this->cosClient->resumeUpload(
$bucket=$this->bucket,
$this->key,
$body,
$uploadId,
array('PartSize'=>$partSize)
);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

批量上传（本地文件夹上传）

功能说明

将本地一个文件夹里所有的文件上传至 COS。

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

function uploadfiles( $path, $cosClient ) {
foreach ( scandir( $path ) as $afile ) {
if ( $afile == '.' || $afile == '..' ) continue;
if ( is_dir( $path.'/'.$afile ) ) {
uploadfiles( $path.'/'.$afile, $cosClient );
} else {
$local_file_path = $path.'/'.$afile;
$cos_file_path = $local_file_path;
// 按照需求自定义拼接上传路径
try {
$cosClient->upload(
$bucket = 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
$key = $cos_file_path,
$body = fopen( $cos_file_path, 'rb' )
);
} catch ( \Exception $e ) {
echo( $e );
}
}
}
}
$local_path = '/data/home/folder';
uploadfiles( $local_path, $cosClient );
```

简单上传

简单上传对象

功能说明

上传对象到指定的存储桶中（PUT Object），最大支持上传不超过5GB的对象，5GB以上对象请使用 分块上传 或 高级接口 上传。

方法原型

```
public Guzzle\Service\Resource\Model putObject(array $args = array())
```

请求示例

示例一：上传本地文件

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
try {
$result = $cosClient->putObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Body' => fopen($local_path, 'rb'),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二：上传归档文件

```php
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
try {
$result = $cosClient->putObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Body' => fopen($local_path, 'rb'),
'StorageClass' => 'Archive'
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例三：上传指定 Content-type 的文件

```php
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
```

```
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
try {
$result = $cosClient->putObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Body' => fopen($local_path, 'rb'),
'ContentType' => 'text/xml'
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例四：创建一个空的虚拟目录

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->putObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'folder/',
'Body' => "",
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
```

```
// 请求失败
echo($e);
}
```

示例五：上传一个生成 ContentMD5 校验的本地文件

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
try {
$result = $cosClient->putObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Body' => fopen($local_path, 'rb'),
'ContentMD5' => true,
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例六：上传对象（单链接限速）

说明：
关于上传对象的限速说明，请参见 单链接限速。

```php
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
try {
$result = $cosClient->putObject(array(
'Bucket' => 'examplebucket-125000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Body' => fopen($local_path, 'rb'),
'TrafficLimit' => 8 * 1024 * 1024 // 限制为1MB/s
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	否
ACL	String	设置对象的 ACL，例如 private、public-read	否

参数名称	类型	描述	是否必填
Body	Stream/String	上传的内容	是
CacheControl	String	缓存策略，设置 Cache-Control	否
ContentDisposition	String	文件名称，设置 Content-Disposition	否
ContentEncoding	String	编码格式，设置 Content-Encoding	否
ContentLanguage	String	语言类型，设置 Content-Language	否
ContentLength	Int	设置传输长度	否
ContentType	String	内容类型，设置 Content-Type	否
Expires	String	设置 Content-Expires	否
Metadata	Array	用户自定义的文件元信息	否
StorageClass	String	文件的存储类型，例如 STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD。更多存储类型，请参见 存储类型概述	否
ContentMD5	Boolean	是否自动生成上传文件的 MD5 值用于校验	否
ServerSideEncryption	String	服务端加密方法	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[ETag] => "698d51a19d8a121ce581499d7b701668"
[VersionId] => MTg0NDUxODMyMTE2ODY0OTExOTk
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[Location] => http://examplebucket-1250000000.cos.ap-chengdu.myqcloud.com/123
[CRC] => 16749565679157681890
)
)
```

返回结果说明

参数名称	类型	描述	父节点

参数名称	类型	描述	父节点
ETag	String	上传文件的 MD5 值	无
VersionId	String	开启版本控制后，文件的版本号	无
CRC	String	CRC64 检验码来进行 数据校验	无

追加上传对象

功能说明

将对象以分块追加的方式上传至存储桶（APPEND Object）。

方法原型

```
public Guzzle\Service\Resource\Model appendObject(array $args = array());
```

请求示例

示例一：追加上传输入流

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->appendObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Position' => 0,
'Body' => "hello,world",
));
// 请求成功
```

```
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二：追加上传本地文件流

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
try {
$result = $cosClient->appendObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Position' => 0,
'Body' => fopen($local_path, 'rb'),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例三：多数据追加上传

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
```

版权所有：腾讯云计算（北京）有限责任公司

```
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
try {
$result = $cosClient->appendObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Position' => 0, //追加对象位置
'Body' => fopen($local_path, 'rb'),//读取文件内容
));

$result = $cosClient->appendObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Position' => (integer)$result['Position'], //取出上一个追加文件的对象位置进行追加
'Body' => "hello, world",
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是

参数名称	类型	描述	是否必填
Position	Integer	追加操作的起始点。首次追加则设置 Position=0，后续追加则设置 Position 为当前 Object 的 content-length	是
Body	File/String	上传分块的内容，可以为本地文件流或输入流	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[ETag] => "9a74ded332531da4c295934ba5a9cf8b"
[Position] => 4
[RequestId] => NjExNWU4NTlfZDIyZjJjMGJfM2Q2ZV8xMzJjZThhZg==
[Key] => exampleobject
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobjec
t
)
```

返回结果说明

参数名称	类型	描述	父节点
ETag	String	文件的 MD5 值	无
Position	Integer	追加操作的起始点	无

查询对象元数据

功能说明

查询 Object 的 Meta 信息（HEAD Object）。

方法原型

```
public Guzzle\Service\Resource\Model headObject(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
```

```
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->headObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
VersionId	String	开启版本控制后，指定文件的具体版本	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[DeleteMarker] =>
```

```
[AcceptRanges] =>
[Expiration] =>
[Restore] =>
[LastModified] => Tue, 02 Apr 2019 12:38:09 GMT
[ContentLength] => 238186
[ETag] => "af9f3b8eaf64473278909183abba1e31"
[MissingMeta] =>
[VersionId] =>
[CacheControl] =>
[ContentDisposition] =>
[ContentEncoding] =>
[ContentLanguage] =>
[ContentType] => text/plain; charset=utf-8
[Expires] =>
[ServerSideEncryption] =>
[Metadata] => Array
(
[md5] => af9f3b8eaf64473278909183abba1e31
)
[SSECustomerAlgorithm] =>
[SSECustomerKeyMD5] =>
[SSEKMSKeyId] =>
[StorageClass] =>
[RequestCharged] =>
[ReplicationStatus] =>
[RequestId] => NWNhMzU3Y2ZfMzFhYzM1MGFfODdhMF8xOTExM2U=
[CRC] => 16749565679157681890
)
)
```

返回结果说明

参数名称	类型	描述	父节点
CacheControl	String	缓存策略，设置 Cache-Control	无
ContentDisposition	String	文件名称，设置 Content-Disposition	无
ContentEncoding	String	编码格式，设置 Content-Encoding	无
ContentLanguage	String	语言类型，设置 Content-Language	无
ContentLength	Int	设置传输长度	无
ContentType	String	内容类型，设置 Content-Type	无
Metadata	Array	用户自定义的文件元信息	无

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	父节点
StorageClass	String	文件的存储类型，例如 STANDARD、STANDARD_IA、ARCHIVE。更多存储类型请参见 存储类型概述	无
ServerSideEncryption	String	服务端加密方法	无
ETag	String	文件的 MD5 值	无
Restore	String	归档文件的回热信息	无
CRC	String	CRC64 检验码来进行 数据校验	无

设置对象复制

将一个对象复制到目标路径（PUT Object - Copy）。

方法原型

```
public Guzzle\Service\Resource\Model copyObject(array $args = array());
```

请求示例

示例一：复制对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->copyObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'CopySource' => 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObje
```

```
ct',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二：复制指定版本的对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->copyObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'CopySource' => 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject?versionId=MTg0NDUxNjI3NTM0ODE2Njc0MzU',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例三：修改存储类型为归档

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
```

```
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->copyObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'CopySource' => 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject',
'StorageClass' => 'Archive'
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例四：修改 meta 属性

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

```
try {
$result = $cosClient->copyObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'CopySource' => 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject',
'MetadataDirective' => 'Replaced',
'Metadata' => array(
'key1' => 'value1',
'key2' => 'value2',
)
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
CopySource	String	描述拷贝源文件的路径，包含 Appid、Bucket、Key、Region，例如 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`	是
MetadataDirective	String	可选值为 Copy、Replaced。设置为 Copy 时，忽略设置的用户元数据信息直接复制，设置为 Replaced 时，按设置的元信息修改元数据，当目标路径和源路径一样时，必须设置为 Replaced	否

分块上传

分块上传对象可包括的操作：

- 分块上传对象： 初始化分块上传， 上传分块， 完成分块上传。
- 分块续传：查询已上传块， 上传分块，完成分块上传。
- 删除已上传分块。

查询分块上传

功能说明

查询指定存储桶中正在进行的分块上传（List Multipart Uploads）。

方法原型

```
public Guzzle\Service\Resource\Model listMultipartUploads(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->listMultipartUploads(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Delimiter' => '/',
'EncodingType' => 'url',
'KeyMarker' => 'prfixKeyMarker',
'UploadIdMarker' => 'string',
'Prefix' => 'prfix',
'MaxUploads' => 1000,
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
```

```
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Delimiter	String	默认为空，设置分隔符，例如设置 `/` 来模拟文件夹	否
EncodingType	String	默认不编码，规定返回值的编码方式，可选值：url	否
KeyMarker	String	标记返回 parts 的 list 的起点位置	否
UploadIdMarker	String	标记返回 parts 的 list 的起点位置	否
Prefix	String	默认为空，对 parts 的 key 进行筛选，匹配指定前缀（prefix）的 objects	否
MaxUploads	Int	最多返回的 parts 数量，默认为最大的1000	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Bucket] => examplebucket-1250000000
[EncodingType] =>
[KeyMarker] =>
[UploadIdMarker] =>
[MaxUploads] => 1000
[Prefix] =>
[IsTruncated] =>
[Uploads] => Array
(
[0] => Array
(
[Key] => exampleobject
[UploadId] => 1551693693b1e6d0e00eec30c534059865ec89c9393028b60bfaf167e9420524b25
eeb2940
[Initiator] => Array
(
```

```
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => 100000000001
)
[Owner] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => 100000000001
)
[StorageClass] => STANDARD
[Initiated] => 2019-03-04T10:01:33.000Z
)
[1] => Array
(
[Key] => exampleobject
[UploadId] => 155374001100563fe0e9d37964d53077e54e9d392bce78f630359cd3288e62acee2
b719534
[Initiator] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => 100000000001
)
[Owner] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => 100000000001
)
[StorageClass] => STANDARD
[Initiated] => 2019-03-28T02:26:51.000Z
)
)
[RequestId] => NWNhNDJmNzBfZWFhZDM1MGFfMjYyM2FfMWIyNzhh
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无
IsTruncated	Int	表示返回的 objects 否被截断	无
Uploads	Array	返回的分块列表	无
Upload	Array	返回的分块属性	Uploads
Key	String	对象键	Upload

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	父节点
UploadId	String	对象的分块上传 ID	Upload
Initiator	String	初始化该分块的操作者	Upload
Owner	String	分块拥有者	Upload
StorageClass	String	分块的存储类型	Upload
Initiated	String	分块初始化时间	Upload

初始化分块上传

功能说明

初始化 Multipart Upload 上传操作（Initiate Multipart Upload）。

方法原型

```
public Guzzle\Service\Resource\Model createMultipartUpload(array $args = array
());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，htt
ps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->createMultipartUpload(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在
COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
));
```

```
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
CacheControl	String	缓存策略，设置 Cache-Control	否
ContentDisposition	String	文件名称，设置 Content-Disposition	否
ContentEncoding	String	编码格式，设置 Content-Encoding	否
ContentLanguage	String	语言类型，设置 Content-Language	否
ContentLength	Int	设置传输长度	否
ContentType	String	内容类型，设置 Content-Type	否
Expires	String	设置 Content-Expires	否
Metadata	Array	用户自定义的文件元信息	否
StorageClass	String	文件的存储类型，例如 STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD。更多存储类型请参见 存储类型概述	否
ContentMD5	Boolean	是否自动生成上传文件的 MD5 值用于校验	否
ServerSideEncryption	String	服务端加密方法	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
```

版权所有：腾讯云计算（北京）有限责任公司

```
[structure:protected] =>
[data:protected] => Array
(
[Bucket] => examplebucket-1250000000
[Key] => exampleobject
[UploadId] => 1554277569b3e83df05c730104c325eb7b56000449fb7d51300b0728aacde02a6ea
7f6c033
[RequestId] => NWNhNDY0YzFfMmZiNTM1MGFfNTM2YV8xYjliMTg=
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Key	String	对象键	无
UploadId	String	对象分块上传的 ID	无

上传分块

分块上传文件（Upload Part）。

方法原型

```
public Guzzle\Service\Resource\Model uploadPart(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，htt
ps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
```

```
'secretKey' => $secretKey)));
try {
$result = $cosClient->uploadPart(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Body' => 'string',
'UploadId' => 'exampleUploadId', //UploadId 为对象分块上传的 ID，在分块上传初始化的返回参数里获得
'PartNumber' => 1, //PartNumber 为分块的序列号，COS 会根据携带序列号合并分块
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
UploadId	String	UploadId 为对象分块上传的 ID，在分块上传初始化的返回参数里获得	是
Body	File/String	上传的内容	是
PartNumber	Int	PartNumber 为分块的序列号，COS 会根据携带序列号合并分块	是
ContentLength	Int	设置传输长度	否
ContentMD5	Boolean	是否自动生成上传文件的 MD5 值用于校验	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[ETag] => "96e79218965eb72c92a549dd5a330112"
[RequestId] => NWNhNDdjYWFfNjNhYjM1MGFfMjk2NF8xY2ViMWM=
```

版权所有：腾讯云计算（北京）有限责任公司

```
[CRC] => 16749565679157681890
)
)
```

返回结果说明

参数名称	类型	描述	父节点
ETag	String	分块的 MD5 值	无
CRC	String	CRC64 检验码来进行 数据校验	无

查询已上传的分块

功能说明

查询特定分块上传操作中的已上传的块（List Parts）。

方法原型

```
public Guzzle\Service\Resource\Model listParts(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->listParts(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'UploadId' => 'exampleUploadId',
```

```
'PartNumberMarker' => 1,
'MaxParts' => 1000,
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
UploadId	String	对象分块上传的 ID	是
PartNumberMarker	Int	标记返回 parts 的 list 的起点位置	否
MaxParts	Int	最多返回的 parts 数量，默认最大值为1000	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Bucket] => examplebucket-1250000000
[Key] => exampleobject
[UploadId] => 1554279643cf19d71bb5fb0d29613e5541131f3a96387d9e168cd939c23a3d608c9
eb94707
[Owner] => Array
(
[ID] => 1250000000
[DisplayName] => 1250000000
)
[PartNumberMarker] => 1
[Initiator] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => 100000000001
)
```

```
[StorageClass] => Standard
[MaxParts] => 1000
[IsTruncated] =>
[Parts] => Array
(
[0] => Array
(
[PartNumber] => 2
[LastModified] => 2019-04-03T08:21:28.000Z
[ETag] => "b948e77469189ac94b98e09755a6dba9"
[Size] => 1048576
)
[1] => Array
(
[PartNumber] => 3
[LastModified] => 2019-04-03T08:21:22.000Z
[ETag] => "9e5060e2994ec8463bfbebd442fdff16"
[Size] => 1048576
)
)
[RequestId] => NWNhNDZkNTJfOGNiMjM1MGFfMTRlYl8xYmJiOTU=
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Key	String	对象键	无
UploadId	String	对象分块上传的 ID	无
IsTruncated	Int	表示返回的 objects 否被截断	无
PartNumberMarker	Int	标记返回 parts 的 list 的起点位置	无
MaxParts	Int	最多返回的 parts 数量，默认最大值为1000	无
Initiator	String	初始化该分块的操作者	无
Parts	Array	返回的分块列表	无
Part	Array	返回的分块属性	Parts
PartNumber	Int	分块标号	Part

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	父节点
LastModified	String	分块的最后上传时间	Part
ETag	String	分块的 MD5 值	Part
Size	String	分块的大小	Part

完成分块上传

功能说明

完成整个文件的分块上传（Complete Multipart Upload）。

方法原型

```
public Guzzle\Service\Resource\Model completeMultipartUpload(array $args = array
());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，htt
ps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->completeMultipartUpload(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在
COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'UploadId' => 'exampleUploadId',
'Parts' => array(
array(
'ETag' => 'exampleETag',
'PartNumber' => 1,
```

```
)),
// ... repeated
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
UploadId	String	对象分块上传的 ID	是
Parts	Array	分块信息列表	是
Part	Array	上传分块的内容信息	是
ETag	String	分块内容的 MD5	是
PartNumber	Int	分块编号	是

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块（Abort Multipart Upload）。

方法原型

```
public Guzzle\Service\Resource\Model abortMultipartUpload(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
```

版权所有：腾讯云计算（北京）有限责任公司

```php
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->abortMultipartUpload(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'UploadId' => 'exampleUploadId',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
UploadId	String	对象分块上传的 ID	是

复制与移动对象

最近更新时间：2022-05-19 14:24:27

简介

本文档提供关于复制与移动对象的相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Object - Copy	设置对象复制（修改对象属性）	复制文件到目标路径
Upload Part - Copy	复制分块	将其他对象复制为一个分块
DELETE Object	删除单个对象	在存储桶中删除指定对象

高级接口（推荐）

复制对象

功能说明

该接口内部会根据文件大小，对小文件调用设置对象复制接口，对大文件调用分块复制接口。该功能常用于修改对象属性。接口参数可参照 `PUT Object - Copy` 和 `Upload Part - Copy` 接口。

请求示例

示例一：复制对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
```

```
'secretKey' => $secretKey)));
try {
$result = $cosClient->Copy(
$bucket = 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
$key = 'exampleobject', //此处的 key 为对象键
$copySorce = array(
'Region' => 'COS_REGION',
'Bucket' => 'sourcebucket-1250000000',
'Key' => 'sourceobject',
)
);
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二：转换对象存储类型

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->Copy(
$bucket = 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
$key = 'exampleobject', //此处的 key 为对象键
$copySorce = array(
'Region' => 'COS_REGION',
'Bucket' => 'examplebucket-1250000000',
'Key' => 'exampleobject',
),
```

```
$options = array(
'StorageClass' => 'Archive'
)
);
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例三：修改对象存储属性

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->Copy(
$bucket = 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
$key = 'exampleobject', //此处的 key 为对象键
$copySorce = array(
'Region' => 'COS_REGION',
'Bucket' => 'sourcebucket-1250000000',
'Key' => 'sourceObject',
),
$options = array(
'MetadataDirective' => 'Replaced',
'Metadata' => array(
'string' => 'string',
),
)
);
// 请求成功
```

```
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

简单复制

设置对象复制

将一个对象复制到目标路径（PUT Object - Copy）。

方法原型

```
public Guzzle\Service\Resource\Model copyObject(array $args = array());
```

请求示例

示例一：复制对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->copyObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'CopySource' => 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject',
));
// 请求成功
```

```
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二：复制指定版本的对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->copyObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'CopySource' => 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject?versionId=MTg0NDUxNjI3NTM0ODE2Njc0MzU',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例三：修改存储类型为归档

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
```

```
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，htt
ps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->copyObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在
COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'CopySource' => 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObje
ct',
'StorageClass' => 'Archive'
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例四：修改 meta 属性

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，htt
ps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->copyObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在
```

```
COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'CopySource' => 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObje
ct',
'MetadataDirective' => 'Replaced',
'Metadata' => array(
'key1' => 'value1',
'key2' => 'value2',
)
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
CopySource	String	描述拷贝源文件的路径，包含 Appid、Bucket、Key、Region，例如 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`	是
MetadataDirective	String	可选值为 Copy、Replaced。设置为 Copy 时，忽略设置的用户元数据信息直接复制，设置为 Replaced 时，按设置的元信息修改元数据，当目标路径和源路径一样时，必须设置为 Replaced	否

分块复制

> 说明：
> 如果为了方便查看理解分块复制接口，这里会罗列其相关所有接口，与上传对象文档中一致。

查询分块

功能说明

查询指定存储桶中正在进行的分块上传（List Multipart Uploads）。

方法原型

```
public Guzzle\Service\Resource\Model listMultipartUploads(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->listMultipartUploads(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Delimiter' => '/',
'EncodingType' => 'url',
'KeyMarker' => 'prfixKeyMarker',
'UploadIdMarker' => 'string',
'Prefix' => 'prfix',
'MaxUploads' => 1000,
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Delimiter	String	默认为空，设置分隔符，例如设置 `/` 来模拟文件夹	否
EncodingType	String	默认不编码，规定返回值的编码方式，可选值：url	否
KeyMarker	String	标记返回 parts 的 list 的起点位置	否
UploadIdMarker	String	标记返回 parts 的 list 的起点位置	否
Prefix	String	默认为空，对 parts 的 key 进行筛选，匹配指定前缀（prefix）的 objects	否
MaxUploads	Int	最多返回的 parts 数量，默认为最大的1000	否

返回结果说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无
IsTruncated	Int	表示返回的 objects 否被截断	无
Uploads	Array	返回的分块列表	无
Upload	Array	返回的分块属性	Uploads
Key	String	对象键	Upload
UploadId	String	对象的分块上传 ID	Upload
Initiator	String	初始化该分块的操作者	Upload
Owner	String	分块拥有者	Upload
StorageClass	String	分块的存储类型	Upload
Initiated	String	分块初始化时间	Upload

初始化分块

功能说明

初始化 Multipart Upload 上传操作（Initiate Multipart Upload）。

方法原型

版权所有：腾讯云计算（北京）有限责任公司

```
public Guzzle\Service\Resource\Model createMultipartUpload(array $args = array
());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->createMultipartUpload(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	是否必填
CacheControl	String	缓存策略，设置 Cache-Control	否
ContentDisposition	String	文件名称，设置 Content-Disposition	否
ContentEncoding	String	编码格式，设置 Content-Encoding	否
ContentLanguage	String	语言类型，设置 Content-Language	否
ContentLength	Int	设置传输长度	否
ContentType	String	内容类型，设置 Content-Type	否
Expires	String	设置 Content-Expires	否
Metadata	Array	用户自定义的文件元信息	否
StorageClass	String	文件的存储类型，例如 STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD。更多存储类型请参见 存储类型概述	否
ContentMD5	Boolean	是否自动生成上传文件的 MD5 值用于校验	否
ServerSideEncryption	String	服务端加密方法	否

返回结果说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Key	String	对象键	无
UploadId	String	对象分块上传的 ID	无

复制分块

功能说明

将其他对象复制为一个分块（Upload Part - Copy）。

方法原型

```
public Guzzle\Service\Resource\Model uploadPartCopy(array $args = array());
```

请求示例

```php
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->uploadPartCopy(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'CopySource' => 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject',
'CopySourceRange' => 'bytes=0-1', //这里举例只拷贝前两个字节
'UploadId' => 'exampleUploadId', //UploadId 为对象分块上传的 ID，在分块上传初始化的返回参数里获得
'PartNumber' => 1, //PartNumber 为分块的序列号，COS 会根据携带序列号合并分块
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
UploadId	String	UploadId 为对象分块上传的 ID，在分块上传初始化的返回参数里获得	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	是否必填
CopySource	String	描述拷贝源文件的路径，包含 APPID、Bucket、Key、Region，例如 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`	是
CopySourceRange	String	描述拷贝源对象的范围，格式为 bytes=first-last。不指定时，默认拷贝整个源对象	否
PartNumber	Int	PartNumber 为分块的序列号，COS 会根据携带序列号合并分块	是
ContentLength	Int	设置传输长度	否
ContentMD5	Boolean	是否自动生成上传文件的 MD5 值用于校验	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[ETag] => "96e79218965eb72c92a549dd5a330112"
[LastModified] => "2017-09-04T04:45:45"
[RequestId] => NWNhNDdjYWFfNjNhYjM1MGFfMjk2NF8xY2ViMWM=
[CRC] => 16749565679157681890
)
)
```

返回结果说明

参数名称	类型	描述	父节点
ETag	String	分块的 MD5 值	无
LastModified	String	返回对象最后修改时间，GMT 格式	无
CRC	String	CRC64 检验码来进行 数据校验	无

完成分块

功能说明

完成整个文件的分块上传（Complete Multipart Upload）。

方法原型

```
public Guzzle\Service\Resource\Model completeMultipartUpload(array $args = array
());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，htt
ps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->completeMultipartUpload(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在
COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'UploadId' => 'exampleUploadId',
'Parts' => array(
array(
'ETag' => 'exampleETag',
'PartNumber' => 1,
)),
// ... repeated
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
UploadId	String	对象分块上传的 ID	是
Parts	Array	分块信息列表	是
Part	Array	上传分块的内容信息	是
ETag	String	分块内容的 MD5	是
PartNumber	Int	分块编号	是

移动对象

移动对象主要包括两个操作：复制源对象到目标位置，删除源对象。

由于 COS 通过存储桶名称（Bucket）和对象键（ObjectKey）来标识对象。移动对象也就意味着修改这个对象的标识，COS PHP SDK 目前没有提供修改对象唯一标识名的单独接口，但是可以通过组合**复制对象**加上**删除对象**的基本操作，来达到修改对象标识的目的，从而实现移动对象。

例如将 mybucket-1250000000 这个存储桶中的 picture.jpg 这个对象移动到同个存储桶的 doc 路径下。首先可以复制 picture.jpg 对象到存储桶的 doc 路径下，即对象键设定为 doc/picture.jpg，复制完成后删除 picture.jpg 这个对象，来实现“移动”的效果。

同样的，如果想将 mybucket-1250000000 这个存储桶里的 picture.jpg 这个对象移动到 myanothorbucket-1250000000 这个存储桶里，可以先将对象复制到 myanothorbucket-1250000000 存储桶，然后删除原来的对象。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
```

```
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$cosClient->copy(
$bucket = 'examplebucket-125000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
$key = 'exampleobject',
$copySource = array(
'Region' => '<sourceRegion>',
'Bucket' => '<sourceBucket>',
'Key' => '<sourceKey>',
)
);
$result = $cosClient->deleteObject(array(
'Bucket' => 'examplebucket-125000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

下载对象

最近更新时间：2022-06-09 15:57:35

简介

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地

高级接口（推荐）

下载对象（断点续传）

功能说明

该接口内部会根据文件大小，对小文件调用设置对象下载接口，对大文件采取并发 range 下载。接口参数可参照 `GET Object` 接口。

方法原型

```
public Qcloud\Cos\Client download(string $bucket, string $key, string $saveAs, array $options = array());
```

参数名称	类型	描述	是否必填
bucket	String	存储桶名称，格式：BucketName-APPID	是
key	String	对象键	是
saveAs	String	保存的本地路径	是
options	Array	附加的配置项	否

options 参数	类型	描述	是否必填
Progress	Function	进度条回调，参数为总大小($totalSize)，已上传大小($downloadedSize)	否

options 参数	类型	描述	是否必填
PartSize	Int	最小分块文件大小，默认为5M	否
Concurrency	Int	并发度，默认为10	否
ResumableDownload	Bool	是否开启断点续传，默认为 False	否
ResumableTaskFile	String	断点文件路径。默认为<saveAs.cosresumabletask>	否

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
$printbar = function($totalSize, $downloadedSize) {
printf("downloaded [%d/%d]\n", $downloadedSize, $totalSize);
};
try {
$result = $cosClient->download(
$bucket = 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
$key = 'exampleobject',
$saveAs = $local_path,
$options= array(
'Progress' => $printbar, //指定进度条
'PartSize' => 10 * 1024 * 1024, //分块大小
'Concurrency' => 5, //并发数
'ResumableDownload' => true, //是否开启断点续传，默认为false
'ResumableTaskFile' => 'tmp.cosresumabletask' //断点文件信息路径，默认为<localpath>.cosresumabletask
)
);
```

```
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

批量下载（从 COS 下载目录）

功能说明

将 COS 目录及其文件下载到本地磁盘。

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$cos_path = 'cos/folder';
$nextMarker = '';
$isTruncated = true;
while ($isTruncated) {
try {
$result = $cosClient->listObjects(array(
'Bucket' => 'examplebucket-125000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Delimiter' => '/', //Delimiter表示分隔符，设置为/表示列出当前目录下的object，设置为空表示列出所有的object
'EncodingType' => 'url',//编码格式，对应请求中的 encoding-type 参数
'Marker' => 'prefix/picture.jpg',//起始对象键标记
'Prefix' => 'prfix/', //Prefix表示列出的object的key以prefix开始
'MaxKeys' => 1000, // 设置最大遍历出多少个对象，一次listObjects最大支持1000
));
} catch (\Exception $e) {
echo($e);
```

```
}
$isTruncated = $result['IsTruncated'];
$nextMarker = $result['NextMarker'];
foreach ( $result['Contents'] as $content ) {
$cos_file_path = $content['Key'];
$local_file_path = $content['Key'];
// 按照需求自定义拼接下载路径
try {
$result = $cosClient->download(
$bucket = 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
$key = $cos_file_path,
$saveAs = $local_file_path
);
echo ($cos_file_path . "\n");
} catch ( \Exception $e ) {
echo($e);
}
}
}
```

简单下载

下载对象

功能说明

下载对象到本地（GET Object）。

方法原型

```
public Guzzle\Service\Resource\Model getObject(array $args = array());
```

请求示例

示例一：下载文件到本地

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，htt
```

```
ps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$local_path = "/Users/xxx/Desktop/exampleobject.txt"; //保存到用户本地路径
try {
$result = $cosClient->getObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'SaveAs' => $local_path,
));
// 请求成功
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二：按照 range 获取文件内容

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->getObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Range' => 'bytes=0-10'
));
```

```
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例三：下载指定版本的文件

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->getObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'VersionId' => 'exampleVersionId'
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例四：下载文件（单链接限速）

> 说明：
> 关于下载对象的限速说明，请参见 单链接限速。

```php
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->getObject(array(
'Bucket' => 'examplebucket-125000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'SaveAs' => '/data/exampleobject',
'TrafficLimit' => 8 * 1024 * 1024 // 限制为1MB/s
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

列出对象

最近更新时间：2022-05-19 14:24:28

简介

本文档提供关于列出对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象
GET Bucket Object Versions	查询对象及其历史版本列表	查询存储桶下的部分或者全部对象及其历史版本信息

查询对象列表

功能说明

查询指定存储桶中所有的对象（List Objects）。

方法原型

```
public Guzzle\Service\Resource\Model listObjects(array $args = array());
```

请求示例

示例一：查询指定前缀、指定开始对象的对象列表

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
```

```
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->listObjects(array(
'Bucket' => 'examplebucket-125000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Delimiter' => '/', //Delimiter表示分隔符，设置为/表示列出当前目录下的object，设置为空表示列出所有的object
'EncodingType' => 'url',//编码格式，对应请求中的 encoding-type 参数
'Marker' => 'prefix/picture.jpg',//起始对象键标记
'Prefix' => 'prfix/', //Prefix表示列出的object的key以prefix开始
'MaxKeys' => 1000, // 设置最大遍历出多少个对象，一次listObjects最大支持1000
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Delimiter	String	默认为空，设置分隔符，例如设置 `/` 来模拟文件夹	否
EncodingType	String	默认不编码，规定返回值的编码方式，可选值：url	否
Marker	String	默认以 UTF-8 二进制顺序列出条目，标记返回 objects 的 list 的起点位置	否
Prefix	String	默认为空，对 object 的 key 进行筛选，匹配指定前缀（prefix）的 objects	否
MaxKeys	Int	最多返回的 objects 数量，默认为最大的1000	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
```

```
[Name] => examplebucket-1250000000
[Prefix] => doc
[Marker] => doc/picture.jpg
[MaxKeys] => 10
[IsTruncated] => 1
[NextMarker] => doc/exampleobject
[Contents] => Array
(
[0] => Array
(
[Key] => doc/exampleobject
[LastModified] => 2019-02-14T12:20:40.000Z
[ETag] => "e37b429559d82e852af0b2f5b4d078ab72b90208"
[Size] => 6532594
[Owner] => Array
(
[ID] => 100000000001
[DisplayName] => 100000000001
)
[StorageClass] => STANDARD
)
[1] => Array
(
[Key] => doc/exampleobject2
[LastModified] => 2019-03-04T06:34:43.000Z
[ETag] => "988f9f28e68eba9b8c1f5f98ccce0a3c"
[Size] => 28
[Owner] => Array
(
[ID] => 100000000001
[DisplayName] => 100000000001
)
[StorageClass] => STANDARD
)
)
[RequestId] => NWNhMzM0MmZfOWUxYzBiMDlfOTk2YV83ZWE3ODE=
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Name	String	存储桶名称，格式：BucketName-APPID	无
Delimiter	String	设置分隔符，例如设置 `/` 来模拟文件夹	无

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	父节点
EncodingType	String	规定返回值的编码方式	无
Marker	String	默认以 UTF-8 二进制顺序列出条目，标记返回 objects 的 list 的起点位置，通过返回的Key值设为Marker实现翻页	无
Prefix	String	对 object 的 key 进行筛选，匹配指定前缀（prefix）的 objects	无
MaxKeys	Int	最多返回的 objects 数量，默认为最大的1000	无
IsTruncated	Int	表示返回的 objects 否被截断	无
Contents	Array	返回的对象列表	无
Content	Array	返回的对象属性，包含所有 objects 元信息的 list，包括 'ETag'，'StorageClass'，'Key'，'Owner'，'LastModified'，'Size' 等信息	Contents

查询对象及其历史版本列表

功能说明

查询存储桶下的部分或者全部对象及其历史版本信息。

方法原型

```
public Guzzle\Service\Resource\Model listObjectVersions(array $args = array());
```

请求示例

示例一：查询历史对象列表

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
```

```
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->listObjectVersions(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Delimiter' => '',
'EncodingType' => 'url',
'KeyMarker' => 'doc/picture.jpg',
'VersionIdMarker' => 'MTg0NDUxODMyMTE2ODY0OTExOTk3W',
'Prefix' => 'doc',
'MaxKeys' => 1000,
));
print_r($result);
} catch (\Exception $e) {
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，由 BucketName-APPID 构成	是
Prefix	String	默认为空，对对象的对象键进行筛选，匹配 prefix 为前缀的对象	否
Delimiter	String	默认为空，设置分隔符，例如设置 `/` 来模拟文件夹	否
KeyMarker	String	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 Key 的起点位置	否
VersionIdMarker	String	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 VersionId 的起点位置	否
MaxKeys	Int	最多返回的对象数量，默认为最大的1000	否
EncodingType	String	默认不编码，规定返回值的编码方式，可选值：url	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
```

版权所有：腾讯云计算（北京）有限责任公司

```
[Name] => examplebucket-1250000000
[Prefix] => doc
[KeyMarker] => string
[VersionIdMarker] => string
[MaxKeys] => 10
[IsTruncated] => 1
[NextKeyMarker] => string
[NextVersionIdMarker] => string
[Versions] => Array
(
[0] => Array
(
[Key] => doc/exampleobject1
[VersionId] => null
[IsLatest] => 1
[LastModified] => 2019-06-13T09:24:52.000Z
[ETag] => "96e79218965eb72c92a549dd5a330112"
[Size] => 6
[StorageClass] => STANDARD
[Owner] => Array
(
[UID] => 1250000000
)
)
[1] => Array
(
[Key] => doc/exampleobject2
[VersionId] => MTg0NDUxODMyMTE2ODY0OTExOTk
[IsLatest] => 1
[LastModified] => 2019-06-18T12:47:03.000Z
[ETag] => "698d51a19d8a121ce581499d7b701668"
[Size] => 3
[StorageClass] => STANDARD
[Owner] => Array
(
[UID] => 1250000000
)
)
)
[RequestId] => NWQwOGVkZGRfMjViMjU4NjRfODNjN18xMTE5YWI4
)
)
```

返回结果说明

参数名称	类型	描述

参数名称	类型	描述
Name	String	存储桶名称，格式：BucketName-APPID
Delimiter	String	设置分隔符，例如设置 `/` 来模拟文件夹
EncodingType	String	规定返回值的编码方式
KeyMarker	String	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 Key 的起点位置
VersionIdMarker	String	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 VersionId 的起点位
NextKeyMarker	String	当 IsTruncated 为 true 时，标记下一次返回对象的 list 的 Key 的起点位置
NextVersionIdMarker	String	当 IsTruncated 为 true 时，标记下一次返回对象的 list 的 VersionId 的起点位置
Prefix	String	对 object 的 key 进行筛选，匹配指定前缀（prefix）的 objects
MaxKeys	Int	最多返回的 objects 数量，默认为最大的1000
IsTruncated	Int	表示返回的 objects 否被截断
Versions	Array	包含所有多个版本对象元数据的 list
Version	Array	包含所有多个版本对象元数据的 list，包括 'ETag'，'StorageClass'，'Key'，'VersionId'，'IsLatest'，'Owner'，'LastModifie 等信息
CommonPrefixes	Array	所有以 Prefix 开头，以 Delimiter 结尾的对象被归到同一类

版权所有：腾讯云计算（北京）有限责任公司

删除对象

最近更新时间：2022-06-23 14:10:20

简介

本文档提供关于对象的删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

删除单个对象

功能说明

在存储桶中删除指定 Object （文件/对象）。

方法原型

```
public Guzzle\Service\Resource\Model deleteObject(array $args = array());
```

请求示例

示例一：删除一个简单对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
```

```
'secretKey' => $secretKey)));
try {
$result = $cosClient->deleteObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject' //若多路径则写为folder/exampleobject，不要在第一层带/，否则删除会失败
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二：删除一个带版本号的对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->deleteObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject', //若多路径则写为folder/exampleobject，不要在第一层带/，否则删除会失败
'VersionId' => 'exampleVersionId' //存储桶未开启版本控制时请勿携带此参数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
VersionId	String	删除文件的版本号	否

删除多个对象

功能说明

在存储桶中批量删除 Object （文件/对象）。

方法原型

```
public Guzzle\Service\Resource\Model deleteObjects(array $args = array());
```

请求示例

示例一：删除多个简单对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
```

```
$result = $cosClient->deleteObjects(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Objects' => array(
array(
'Key' => 'exampleobject', //若多路径则写为folder/exampleobject，不要在第一层带/，否则删除会失败
),
// ... repeated
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二：删除多个带版本号的对象

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->deleteObjects(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Objects' => array(
array(
'Key' => 'exampleobject', //若多路径则写为folder/exampleobject，不要在第一层带/，否则删除会失败
'VersionId' => 'string' //存储桶未开启版本控制时请勿携带此参数
),
// ... repeated
```

```
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

指定前缀删除（删除文件夹）

功能说明

在存储桶中删除文件夹。

方法原型

```
public Guzzle\Service\Resource\Model deleteObjects(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
$cos_prefix = "cos/folder";
$nextMarker = '';
$isTruncated = true;
while ( $isTruncated ) {
try {
$result = $cosClient->listObjects(
['Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
```

```
'Delimiter' => '',
'EncodingType' => 'url',
'Marker' => $nextMarker,
'Prefix' => $cos_prefix,
'MaxKeys' => 1000]
);
$isTruncated = $result['IsTruncated'];
$nextMarker = $result['NextMarker'];
foreach ( $result['Contents'] as $content ) {
$cos_file_path = $content['Key'];
$local_file_path = $content['Key'];
// 按照需求自定义拼接下载路径
try {
$cosClient->deleteObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => $cos_file_path,
));
echo ( $cos_file_path . "\n" );
} catch ( \Exception $e ) {
echo( $e );
}
}
} catch ( \Exception $e ) {
echo( $e );
}
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Objects	Array	删除对象列表	是
Object	Array	删除的对象	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
VersionId	String	删除文件的版本号	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Deleted] => Array
(
[0] => Array
(
[Key] => exampleobject1
)
)
[Errors] => Array
(
[0] => Array
(
[Key] => exampleobject2
[Code] =>
[Message] =>
)
)
[RequestId] => NWNhZWYzYWNfMTlhYTk0MGFfNGRjX2MzZTVhOQ==
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Deleted	Array	成功删除的对象的列表	无
Errors	Array	失败删除的对象的列表	无
Key	String	对象键	Deleted/Errors
Code	String	失败错误码	Errors
Message	String	失败错误信息	Errors

版权所有：腾讯云计算（北京）有限责任公司

恢复归档对象

最近更新时间：2022-01-05 14:51:45

简介

本文档提供关于恢复归档对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
POST Object restore	恢复归档对象	将归档类型的对象取回访问

恢复归档对象

功能说明

将归档类型的对象取回访问（POST Object restore）。

方法原型

```
public Guzzle\Service\Resource\Model restoreObject(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->restoreObject(array(
```

```
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Days' => 10,
'CASJobParameters' => array(
'Tier' =>'Expedited'
)
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
Days	String	设置临时副本的过期时间，单位(天)	是
CASJobParameters	Array	恢复信息	是
Tier	String	恢复归档存储类型的数据时，Tier 可以指定为 COS 支持的三种恢复模式，分别为 Expedited、Standard、Bulk。恢复深度归档存储类型的数据时，Tier 可以指定为 COS 支持的两种恢复模式，分别为 Standard、Bulk。	是

版权所有：腾讯云计算（北京）有限责任公司

生成对象访问 URL

最近更新时间：2022-03-21 14:28:01

简介

本文档提供获取对象访问 URL 的代码示例。

获取对象访问 URL

功能说明

查询对象访问 URL。

说明：

- 如何使生成的对象 URL 在浏览器中打开是预览，而不是下载：在获取的 URL 后拼接参数 `response-content-disposition=inline` 。
- 如何使生成的对象 URL 在浏览器中打开是下载，而不是预览：在获取的 URL 后拼接参数 `response-content-disposition=attachment` 。

使用示例

示例一

```
//获取带有签名的对象下载链接
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
```

```
'secretKey' => $secretKey)));

try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
$signedUrl = $cosClient->getObjectUrl($bucket, $key, '+10 minutes');
// 请求成功
echo $signedUrl;
} catch (\Exception $e) {
// 请求失败
print_r($e);
}
```

示例二

```
//获取不带有签名的对象下载链接，用于匿名下载或分发
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$bucket = 'examplebucket-125000000'; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //此处的 key 为对象键，对象键是对象在存储桶中的唯一标识
$Url = $cosClient -> getObjectUrlWithoutSign($bucket, $key);
// 请求成功
echo $Url;
} catch (\Exception $e) {
// 请求失败
print_r($e);
}
```

说明：

更多示例请查看 预签名 URL。

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
Expires	签名几秒后失效，默认为1800秒	String	是

返回结果说明

方法返回值为对象访问的 URL。

生成预签名 URL

最近更新时间：2022-06-09 15:50:44

简介

PHP SDK 提供获取请求预签名 URL 接口，请求示例如下。

说明：

- 建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。
- 获取对象的 URL 并下载对象参数，可在获取的 URL 后拼接参数 `response-content-disposition=attachment` 。

永久密钥预签名请求示例

上传请求示例

示例一：简单上传生成永久密钥预签名

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'signHost' => true, //默认签入Header Host；您也可以选择不签入Header Host，但可能导致请求失败或安全漏洞，若不签入host则填false
'credentials'=> array(
'secretId' => $secretId ,
```

```
'secretKey' => $secretKey)));
### 简单上传预签名
try {
$signedUrl = $cosClient->getPreSignedUrl('putObject', array(
'Bucket' => "examplebucket-1250000000", //存储桶，格式：BucketName-APPID
'Key' => "exampleobject", //对象在存储桶中的位置，即对象键
'Body' => 'string', //可为空或任意字符串
'Params'=> array(), //http 请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此HTTP请求的参数,默认为空
'Headers'=> array(), //http 请求头部，传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的HTTP请求头部,默认签入host
), '+10 minutes'); //签名的有效时间
// 请求成功
echo ($signedUrl);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二：分块上传生成永久密钥预签名

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'signHost' => true, //默认签入Header Host；您也可以选择不签入Header Host，但可能导致请求失败或安全漏洞,若不签入host则填false
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
### 分块上传预签名
try {
$signedUrl = $cosClient->getPreSignedUrl('uploadPart', array(
'Bucket' => "examplebucket-1250000000", //存储桶，格式：BucketName-APPID
'Key' => "exampleobject", //对象在存储桶中的位置，即对象键
'UploadId' => 'string', //上传ID
'PartNumber' => '1', //块号
```

```
'Body' => 'string',
'Params'=> array(), //http 请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此HTTP请求的参数,默认为空
'Headers'=> array(), //http 请求头部，传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的HTTP请求头部,默认已签入host
), '+10 minutes'); //签名的有效时间
// 请求成功
echo ($signedUrl);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

下载请求示例

示例一：简单下载生成永久密钥预签名

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'signHost' => true, //默认签入Header Host；您也可以选择不签入Header Host，但可能导致请求失败或安全漏洞,若不签入host则填false
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
### 简单下载预签名
try {
$signedUrl = $cosClient->getPreSignedUrl('getObject', array(
'Bucket' => "examplebucket-1250000000", //存储桶，格式：BucketName-APPID
'Key' => "exampleobject", //对象在存储桶中的位置，即对象键
'Params'=> array(), //http 请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此HTTP请求的参数,默认为空
'Headers'=> array(), //http 请求头部，传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的HTTP请求头部,默认已签入host
), '+10 minutes'); //签名的有效时间
// 请求成功
echo ($signedUrl);
```

```
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二：使用封装的 getObjectUrl 获取下载签名生成永久密钥预签名

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'signHost' => true, //默认签入Header Host；您也可以选择不签入Header Host，但可能导致请求失败或安全漏洞，若不签入host则填false
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
### 使用封装的 getObjectUrl 获取下载签名
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //对象在存储桶中的位置，即对象键
$signedUrl = $cosClient->getObjectUrl($bucket, $key, '+10 minutes'); //签名的有效时间
// 请求成功
echo $signedUrl;
} catch (\Exception $e) {
// 请求失败
print_r($e);
}
```

临时密钥预签名请求示例

上传请求示例

示例一：简单上传生成临时密钥预签名

```php
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$tmpSecretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$tmpSecretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$tmpToken = "COS_TOKEN"; //使用临时密钥需要填入，临时密钥生成和使用指引参见https://cloud.tencent.com/document/product/436/14048
$region = "COS_REGION"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'signHost' => true, //默认签入Header Host；您也可以选择不签入Header Host，但可能导致请求失败或安全漏洞,若不签入host则填false
'credentials'=> array(
'secretId' => $tmpSecretId,
'secretKey' => $tmpSecretKey,
'token' => $tmpToken)));
### 简单上传预签名
try {
$signedUrl = $cosClient->getPreSignedUrl('putObject', array(
'Bucket' => "examplebucket-1250000000", //存储桶，格式：BucketName-APPID
'Key' => "exampleobject", //对象在存储桶中的位置，即对象键
'Body' => 'string',
'Params'=> array(), //http 请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此HTTP请求的参数,默认为空
'Headers'=> array(), //http 请求头部，传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的HTTP请求头部,默认已签入host
), '+10 minutes'); //签名的有效时间
// 请求成功
echo ($signedUrl);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二：分块上传生成临时密钥预签名

```php
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$tmpSecretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$tmpSecretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管
```

```
理，https://console.intl.cloud.tencent.com/cam/capi
$tmpToken = "COS_TOKEN"; //使用临时密钥需要填入，临时密钥生成和使用指引参见https://cloud.tencent.com/document/product/436/14048
$region = "COS_REGION"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'signHost' => true, //默认签入Header Host；您也可以选择不签入Header Host，但可能导致请求失败或安全漏洞,若不签入host则填false
'credentials'=> array(
'secretId' => $tmpSecretId,
'secretKey' => $tmpSecretKey,
'token' => $tmpToken)));
### 分块上传预签名
try {
$signedUrl = $cosClient->getPreSignedUrl('uploadPart', array(
'Bucket' => "examplebucket-1250000000", //存储桶，格式：BucketName-APPID
'Key' => "exampleobject", //对象在存储桶中的位置，即对象键
'UploadId' => '', //上传ID
'PartNumber' => '1', //块号
'Body' => '',
'Params'=> array(), //http 请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此HTTP请求的参数,默认为空
'Headers'=> array(), //http 请求头部，传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的HTTP请求头部,默认已签入host
), '+10 minutes'); //签名的有效时间
// 请求成功
echo ($signedUrl);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

下载请求示例

示例一：简单下载生成临时密钥预签名

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$tmpSecretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$tmpSecretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$tmpToken = "COS_TOKEN"; //使用临时密钥需要填入，临时密钥生成和使用指引参见https://cloud.
```

```
tencent.com/document/product/436/14048
$region = "COS_REGION"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'signHost' => true, //默认签入Header Host；您也可以选择不签入Header Host，但可能导致请求失败或安全漏洞,若不签入host则填false
'credentials'=> array(
'secretId' => $tmpSecretId,
'secretKey' => $tmpSecretKey,
'token' => $tmpToken)));

### 简单下载预签名
try {
$signedUrl = $cosClient->getPreSignedUrl('getObject', array(
'Bucket' => "examplebucket-1250000000", //存储桶，格式：BucketName-APPID
'Key' => "exampleobject", //对象在存储桶中的位置，即对象键
'Params'=> array(), //http 请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此HTTP请求的参数,默认为空
'Headers'=> array(), //http 请求头部，传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的HTTP请求头部,默认已签入host
), '+10 minutes'); //签名的有效时间
// 请求成功
echo ($signedUrl);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二：使用封装的 getObjectUrl 获取下载签名生成临时密钥预签名

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$tmpSecretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$tmpSecretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$tmpToken = "COS_TOKEN"; //使用临时密钥需要填入，临时密钥生成和使用指引参见https://cloud.tencent.com/document/product/436/14048
$region = "COS_REGION"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
```

```
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $tmpSecretId,
'secretKey' => $tmpSecretKey,
'token' => $tmpToken)));

### 使用封装的 getObjectUrl 获取下载签名
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //对象在存储桶中的位置，即对象键
$signedUrl = $cosClient->getObjectUrl($bucket, $key, '+10 minutes'); //签名的有效时间
// 请求成功
echo $signedUrl;
} catch (\Exception $e) {
// 请求失败
print_r($e);
}
```

判断对象是否存在

最近更新时间：2022-05-19 14:24:28

简介

本文档提供快捷查询存储桶中某个对象是否存在的示例代码。

示例代码实际调用了 COS API HeadObject，是该接口的简化版。

HeadObject 除了检查对象是否存在，主要功能为返回对象元数据。包含了 HeadObject 完整功能的 SDK 接口，可参见 查询对象元数据。

查询对象元数据

功能说明

检查存储桶中是否存在某个对象。

方法原型

```
public Guzzle\Service\Resource\Model doesObjectExist(array $args = array());
```

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->doesObjectExist(
```

```
'examplebucket-125000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'exampleobject' //对象名
);
// 请求成功
print_r($result);//返回布尔值
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

服务端加密

最近更新时间：2022-05-19 14:24:28

如果需要对上传的对象进行加密，我们支持以下加密方式。

使用 COS 托管加密密钥的服务端加密（SSE-COS）保护数据

由腾讯云 COS 托管主密钥和管理数据。COS 会帮助您在数据写入数据中心时自动加密，并在您取用该数据时自动解密。目前支持使用 COS 主密钥对数据进行 AES-256 加密。

```
try {
$result = $cosClient->putObject(array(
'Bucket' => 'examplebucket-125000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject'
'Body' => 'string',
'ServerSideEncryption' => 'AES256', //SSE-COS加密
),
);
print_r ($result);
} catch (Qcloud\Cos\Exception\ServiceResponseException $e) {
echo $e;
}
```

使用客户提供的加密密钥的服务端加密 （SSE-C）保护数据

加密密钥由用户自己提供，用户在上传对象时，COS 将使用用户提供的加密密钥对用户的数据进行 AES-256 加密。SDK 通过在上传时设置 `SSE` 相关头部字段来完成。

注意：

- 该加密所运行的服务需要使用 HTTPS 请求。
- customerKey：用户提供的密钥，传入一个32字节的字符串，支持数字、字母、字符的组合，不支持中文。
- 如果上传的源文件调用了该方法，那么在使用 GET（下载）、HEAD（查询）时对源对象操作的时候也要调用该方法。

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
```

版权所有：腾讯云计算（北京）有限责任公司

```
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

$bucket = 'examplebucket-125000000'; //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
$key = 'exampleobject';
try {
$customerKey = 'abcdefghijklmnopqrstuvwxyz123456'; //32字节的字符串，支持数字、字母、字符的组合，不支持中文
$SSECustomerKey = base64_encode($customerKey);
$SSECustomerKeyMd5 = base64_encode(md5($customerKey, true));
$result = $cosClient->putObject(array(
'Bucket' => $bucket,
'Key' => $key,
'Body' => 'string',
'SSECustomerAlgorithm' => 'AES256',
'SSECustomerKey' => $SSECustomerKey,
'SSECustomerKeyMD5' => $SSECustomerKeyMd5,
));
print_r ($result);
} catch (Qcloud\Cos\Exception\ServiceResponseException $e) {
echo $e;
}
```

查询对象元数据

最近更新时间：2022-05-19 14:24:28

简介

本文档提供关于查询对象元数据操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

查询对象元数据

功能说明

查询 Object 的 Meta 信息（HEAD Object）。

方法原型

```
public Guzzle\Service\Resource\Model headObject(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->headObject(array(
```

```
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
VersionId	String	开启版本控制后，指定文件的具体版本	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[DeleteMarker] =>
[AcceptRanges] =>
[Expiration] =>
[Restore] =>
[LastModified] => Tue, 02 Apr 2019 12:38:09 GMT
[ContentLength] => 238186
[ETag] => "af9f3b8eaf64473278909183abba1e31"
[MissingMeta] =>
[VersionId] =>
[CacheControl] =>
[ContentDisposition] =>
[ContentEncoding] =>
[ContentLanguage] =>
[ContentType] => text/plain; charset=utf-8
```

版权所有：腾讯云计算（北京）有限责任公司

```
[Expires] =>
[ServerSideEncryption] =>
[Metadata] => Array
(
[md5] => af9f3b8eaf64473278909183abba1e31
)
[SSECustomerAlgorithm] =>
[SSECustomerKeyMD5] =>
[SSEKMSKeyId] =>
[StorageClass] =>
[RequestCharged] =>
[ReplicationStatus] =>
[RequestId] => NWNhMzU3Y2ZfMzFhYzM1MGFfODdhMF8xOTExM2U=
[CRC] => 16749565679157681890
)
)
```

返回结果说明

参数名称	类型	描述	父节点
CacheControl	String	缓存策略，设置 Cache-Control	无
ContentDisposition	String	文件名称，设置 Content-Disposition	无
ContentEncoding	String	编码格式，设置 Content-Encoding	无
ContentLanguage	String	语言类型，设置 Content-Language	无
ContentLength	Int	设置传输长度	无
ContentType	String	内容类型，设置 Content-Type	无
Metadata	Array	用户自定义的文件元信息	无
StorageClass	String	文件的存储类型，例如 STANDARD、STANDARD_IA、ARCHIVE。更多存储类型请参见 存储类型概述	无
ServerSideEncryption	String	服务端加密方法	无
ETag	String	文件的 MD5 值	无
Restore	String	归档文件的回热信息	无
CRC	String	CRC64 检验码来进行 数据校验	无

存储桶操作
创建存储桶

最近更新时间：2023-06-16 15:21:34

简介

本文档提供关于创建存储桶的 API 概览以及 SDK 示例代码。

注意：

- 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄露目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

API	操作名	操作描述
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶

创建存储桶

功能说明

在指定账号下创建一个存储桶。同一用户账号下，可以创建多个存储桶，数量上限是200个（不区分地域），存储桶中的对象数量没有限制。创建存储桶是低频操作，一般建议在控制台创建 Bucket，在 SDK 进行 Object 的操作。

方法原型

```
public Guzzle\Service\Resource\Model createBucket(array $args = array());
```

请求示例

```
<?php

require dirname(__FILE__) . '/../vendor/autoload.php';

$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
```

```
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->createBucket(array(
'Bucket' => 'examplebucket-125000000' //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	父节点	描述	类型
Bucket	无	存储桶名称，格式：BucketName-APPID	String

删除存储桶

最近更新时间：2022-03-02 16:10:36

简介

本文档提供关于删除存储桶的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

删除存储桶

功能说明

删除指定的存储桶（DELETE Bucket）。

> 注意：
> 删除存储桶前，请确保存储桶内的数据和未完成上传的分块数据已全部清空，否则会无法删除存储桶。

示例代码

方法原型

```
public Guzzle\Service\Resource\Model deleteBucket(array $args = array());
```

请求示例

```
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
```

```
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->deleteBucket(array(
'Bucket' => 'examplebucket-125000000' //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	父节点	描述	类型
Bucket	无	存储桶名称，格式：BucketName-APPID	String

删除所有存储桶

功能说明

删除所有的存储桶（DELETE Buckets）。

示例代码

方法原型

```
public Guzzle\Service\Resource\Model deleteBuckets(array $args = array());
```

请求示例

```
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，htt
```

```
ps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
//列出所有buckets
$buckets = $cosClient->listBuckets();
//若bucket列表不为空则进行删除逻辑，先删除对象，再判断是否有上传的任务进行删除，最后删除桶
//可以在foreach中添加exclude逻辑(如建立一个需保留的桶数组)，保留无需保留的桶
if (!empty($buckets['Buckets'][0])) {
foreach ($buckets['Buckets'][0]['Bucket'] as $key => $value) {
$result = $cosClient->listObjects(array('Bucket' => $value['Name']));
if (isset($result['Contents'])) {
foreach ($result['Contents'] as $content) {
$cosClient->deleteObject(array('Bucket' => $value['Name'], 'Key' => $content['Ke
y']));
}
}
while(True){
$result = $cosClient->ListMultipartUploads(
array('Bucket' => $value['Name']));
if ($result['Uploads'] == array()) {
break;
}
foreach ($result['Uploads'] as $upload) {
try {
$cosClient->AbortMultipartUpload(
array('Bucket' => $value['Name'],
'Key' => $upload['Key'],
'UploadId' => $upload['UploadId']));
} catch (\Exception $e) {
print_r($e);
}
}
}
$cosClient->deleteBucket(array('Bucket' => $value['Name']));
}
}
print_r('DELETE ALL BUCKETS SUCCEED!');
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	父节点	描述	类型
Bucket	无	存储桶名称，格式：BucketName-APPID	String

版权所有：腾讯云计算（北京）有限责任公司

查询存储桶列表

最近更新时间：2022-05-19 14:33:07

简介

本文档提供关于查询存储桶列表的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Service（List Buckets）	查询存储桶列表	查询指定账号下所有的存储桶列表

查询存储桶列表

功能说明

用于查询指定账号下所有存储桶列表。

方法原型

```
public Guzzle\Service\Resource\Model listBuckets();
```

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->listBuckets();
// 请求成功
```

```
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

返回结果示例

```
Array
(
[data:protected] => Array
(
[Owner] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => 100000000001
)
[Buckets] => Array
(
[0] => Array
(
[Name] => examplebucket-1250000000
[Location] => ap-beijing
[CreationDate] => 2016-07-29T03:09:54Z
)
[1] => Array
(
[Name] => examplebucket2-1250000000
[Location] => ap-beijing
[CreationDate] => 2017-08-02T04:00:24Z
)
)
[RequestId] => NWE3YzgxZmFfYWZhYzM1MGFfMzc3MF9iOGY5OQ==
)
)
```

返回结果说明

参数名称	父节点	描述	类型
Owner	无	存储桶所有者信息	Array
ID	Owner	存储桶所有者 ID	String
DisplayName	Owner	存储桶所有者的名字信息	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	父节点	描述	类型
Buckets	无	存储桶列表	Array
Bucket	Buckets	存储桶信息	Array
Name	Bucket	存储桶名称	String
Location	Bucket	存储桶所在的地域名	String
CreationDate	Bucket	存储桶创建的时间	String

版权所有：腾讯云计算（北京）有限责任公司

检索存储桶

最近更新时间：2022-05-19 14:33:07

简介

本文档提供关于检索存储桶的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问

检索存储桶及其权限

功能说明

HEAD Bucket 请求可以确认该存储桶是否存在，是否有权限访问。有以下几种情况：

- 存储桶存在且有读取权限，返回 HTTP 状态码为200。
- 无存储桶读取权限，返回 HTTP 状态码为403。
- 存储桶不存在，返回 HTTP 状态码为404。

方法原型

```
public Guzzle\Service\Resource\Model headBucket(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
```

```
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->headBucket(array(
'Bucket' => 'examplebucket-125000000' //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	父节点	描述	类型
Bucket	无	存储桶名称，格式：BucketName-APPID	String

判断存储桶是否存在

最近更新时间：2022-05-19 14:33:07

简介

本文档提供快捷查询某个存储桶是否存在的示例代码。

示例代码实际调用了COS API HeadBucket，是该接口的简化版。

HeadBucket除了检查存储桶是否存在，还可以判断是否有权限访问存储桶，有以下几种情况：

- 存储桶存在且有读取权限，返回 HTTP 状态码为200。
- 无存储桶读取权限，返回 HTTP 状态码为403。
- 存储桶不存在，返回 HTTP 状态码为404。

检查存储桶是否存在

功能说明

检查存储桶是否存在。

方法原型

```
public Guzzle\Service\Resource\Model doesBucketExist(array $args = array());
```

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
```

```
'secretKey' => $secretKey)));
try {
$result = $cosClient->doesBucketExist(
'examplebucket-125000000'//存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
); ;
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

数据管理
生命周期

最近更新时间：2022-01-05 14:54:55

本文档提供关于生命周期的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket lifecycle	设置生命周期	设置存储桶的生命周期管理的配置
GET Bucket lifecycle	查询生命周期	查询存储桶生命周期管理的配置
DELETE Bucket lifecycle	删除生命周期	删除存储桶生命周期管理的配置

设置生命周期

功能说明

设置指定存储桶的生命周期配置信息（PUT Bucket lifecycle）。

方法原型

```
public Guzzle\Service\Resource\Model putBucketLifecycle(array $args = array());
```

请求示例

示例一：全部对象生成1天后删除

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
```

```
'secretKey' => $secretKey)));
try {
$result = $cosClient->putBucketLifecycle(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Rules' => array(
array(
'Expiration' => array(
'Days' => 1,
),
'ID' => 'rule01',
'Filter' => array(
'Prefix' => ''
),
'Status' => 'Enabled',
),
)
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

示例二：某前缀下对象生成1天沉降为归档

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->putBucketLifecycle(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
```

```
'Rules' => array(
array(
'ID' => 'rule01',
'Filter' => array(
'Prefix' => 'prefix01/'
),
'Status' => 'Enabled',
'Transitions' => array(
array(
'Days' => 1,
'StorageClass' => 'Archive'
),
),
),
)
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	设置生命周期的存储桶，格式：BucketName-APPID	是
Rules	Array	生命周期信息列表	是
Rule	Array	生命周期信息	是
Expiration	Array	设置 Object 过期规则，可以指定天数 Days 或者指定日期 Date	否
Transition	Array	设置 Object 转换存储类型规则	否
NoncurrentVersionExpiration	Array	设置 历史 Object 过期规则	否
NoncurrentVersionTransition	Array	设置 历史 Object 转换存储类型规则	否
Filter	Array	用于描述规则影响的 Object 集合	是
Prefix	String	过滤的对象的前缀	是
Status	String	设置 Rule 是否启用，可选值为 Enabled 、 Disabled	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	必填
ID	String	配置规则的 ID	是
Days	Int	设置生效的天数	否
Date	Int / String	设置生效的日期	否
NoncurrentDays	Int	设置非多版本对象生效的天数	否
StorageClass	String	转换的文件的存储类型，STANDARD 、 STANDARD_IA 、 ARCHIVE，默认值：STANDARD	是

查询生命周期

功能说明

查询存储桶的生命周期管理配置（GET Bucket lifecycle）。

方法原型

```
public Guzzle\Service\Resource\Model getBucketLifecycle(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->getBucketLifecycle(array(
'Bucket' => 'examplebucket-1250000000' //存储桶名称，由BucketName-Appid 组成，可以在C
```

```
OS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	查询生命周期的存储桶，格式：BucketName-APPID	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[data:protected] => Array
(
[Rules] => Array
(
[0] => Array
(
[ID] => id1
[Filter] => Array
(
[Prefix] => documents/
)
[Status] => Enabled
[Transition] => Array
(
[Days] => 200
[StorageClass] => Standard_IA
)
[Expiration] => Array
(
[Days] => 1000
)
)
)
[RequestId] => NWE3YzhlZjNfY2FhMzNiMGFfNDVkNF8yZDIxODE=
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Rules	Array	生命周期信息列表	无
Rule	Array	生命周期信息	Rules
Expiration	Array	设置 Object 过期规则，可以指定天数 Days 或者指定日期 Date	Rule
Transition	Array	设置 Object 转换存储类型规则	Rule
Filter	Array	用于描述规则影响的 Object 集合	Rule
Prefix	String	过滤的对象的前缀	Filter
Status	String	设置 Rule 是否启用，可选值为 Enabled 、 Disabled	Rule
ID	String	配置规则的 ID	Rule
Days	Int	设置生效的天数	Expiration / Transition
Date	Int / String	设置生效的日期	Expiration / Transition
StorageClass	String	转换的文件的存储类型，STANDARD 、 STANDARD_IA 、 ARCHIVE，默认值：STANDARD	Transition

删除生命周期

功能说明

删除 Bucket 生命周期管理的配置（DELETE Bucket lifecycle）。

方法原型

```
public Guzzle\Service\Resource\Model deleteBucketLifecycle(array $args = array
());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
```

```
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->deleteBucketLifecycle(array(
'Bucket' => 'examplebucket-1250000000' //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	被删除生命周期配置的存储桶，格式：BucketName-APPID	是

日志管理

最近更新时间：2022-01-11 12:39:36

本文档提供关于日志管理的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket logging	设置日志管理	为源存储桶开启日志记录
GET Bucket logging	查询日志管理	查询源存储桶的日志配置信息

设置日志管理

功能说明

PUT Bucket logging 用于为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中。

方法原型

```
public Guzzle\Service\Resource\Model PutBucketLogging(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->putBucketLogging(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'LoggingEnabled' => array(
```

```
'TargetBucket' => 'examplebucket2-1250000000',
'TargetPrefix' => '',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	描述	类型
Bucket	开启日志功能的源存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
TargetBucket	存放日志的目标存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
TargetPrefix	日志存放在目标存储桶的指定路径	String

查询日志管理

功能说明

GET Bucket logging 用于查询指定存储桶的日志配置信息。

方法原型

```
public Guzzle\Service\Resource\Model GetBucketLogging(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
```

```
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->getBucketLogging(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	描述	类型
Bucket	源存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[LoggingEnabled] => Array
(
[TargetBucket] => examplebucket2-1250000000
[TargetPrefix] =>
)
[RequestId] => NWRmMWJjOThfMjZiMjU4NjRfODY4X2ExMjcy****
)
```

返回结果说明

成员变量	描述	类型
TargetBucket	日志存放的目标存储桶	String
TargetPrefix	日志存放的目标存储桶路径	String

存储桶标签

最近更新时间：2022-01-05 14:54:55

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket tagging	设置存储桶标签	为已存在的存储桶设置标签
GET Bucket tagging	查询存储桶标签	查询指定存储桶下已有的存储桶标签
DELETE Bucket tagging	删除存储桶标签	删除指定的存储桶标签

设置存储桶标签

功能说明

PUT Bucket tagging 用于为已存在的存储桶设置标签。

方法原型

```
public Guzzle\Service\Resource\Model PutBucketTagging(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->putBucketTagging(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在
```

```
COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'TagSet' => array(
array('Key'=>'key1',
'Value'=>'value1',
),
array('Key'=>'key2',
'Value'=>'value2',
),
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	描述	类型
Bucket	设置标签的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
Key	标签的键	String
Value	标签的值	String

查询存储桶标签

功能说明

GET Bucket tagging 用于查询指定存储桶下已有的存储桶标签。

方法原型

```
public Guzzle\Service\Resource\Model GetBucketTagging(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
```

```
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，htt
ps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->getBucketTagging(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在
COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	描述	类型
Bucket	查询标签的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[TagSet] => Array
(
[0] => Array
(
[Key] => key1
[Value] => value1
)
[1] => Array
(
[Key] => key2
[Value] => value2
)
)
```

```
[RequestId] => NWRmMWVkMjFfMjJiMjU4NjRfNWQ3X2EwMWVj****
)
```

返回结果说明

成员变量	描述	类型
Key	标签的键	String
Value	标签的值	String

删除存储桶标签

功能说明

DELETE Bucket tagging 用于删除指定存储桶下已有的存储桶标签。

方法原型

```
public Guzzle\Service\Resource\Model DeleteBucketTagging(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->deleteBucketTagging(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
);
```

```
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	描述	类型
Bucket	被删除标签的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String

对象标签

最近更新时间：2022-05-19 14:40:49

简介

本文档提供关于对象标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Object tagging	设置对象标签	为对象设置标签
GET Object tagging	查询对象标签	查询指定对象下已有的对象标签
DELETE Object tagging	删除对象标签	删除指定对象下已有的对象标签

设置对象标签

功能说明

PUT Object tagging 用于为对象设置标签。

方法原型

```
public Guzzle\Service\Resource\Model PutObjectTagging(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
```

```
'secretKey' => $secretKey)));
try {
$result = $cosClient->putObjectTagging(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'TagSet' => array(
array('Key'=>'key1',
'Value'=>'value1',
),
array('Key'=>'key2',
'Value'=>'value2',
),
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	描述	类型
Bucket	设置标签的对象所在的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
Key	设置标签的对象键，对象键（Key）是对象在存储桶中的唯一标识，详情请参见 对象键	String
TagSet	对象的标签配置集合	Array

TagSet 成员说明：

参数名称	描述	类型
Key	标签的 key	String
Value	标签的 value	String

查询对象标签

功能说明

GET Object tagging 用于查询指定对象下已有的对象标签。

方法原型

```
public Guzzle\Service\Resource\Model GetObjectTagging(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->getObjectTagging(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	描述	类型
Bucket	查询标签的对象所在的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
Key	查询标签的对象键，对象键（Key）是对象在存储桶中的唯一标识，详情请参见 对象键	String

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[TagSet] => Array
(
[0] => Array
(
[Key] => key1
[Value] => value1
)
[1] => Array
(
[Key] => key2
[Value] => value2
)
)
[RequestId] => NWRmMWVkMjFfMjJiMjU4NjRfNWQ3X2EwMWVj****
)
```

返回结果说明

成员变量	描述	类型
Key	标签的键	String
Value	标签的值	String

删除对象标签

功能说明

DELETE Object tagging 用于删除指定对象的已有标签。

方法原型

```
public Guzzle\Service\Resource\Model DeleteObjectTagging(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
```

```
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->deleteObjectTagging(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
);
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	描述	类型
Bucket	删除标签的对象所在的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
Key	删除标签的对象键，对象键（Key）是对象在存储桶中的唯一标识，详情请参见 对象键	String

静态网站

最近更新时间：2022-01-05 14:54:56

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket website	设置静态网站	设置存储桶的静态网站配置
GET Bucket website	查询静态网站配置	查询存储桶的静态网站配置
DELETE Bucket website	删除静态网站配置	删除存储桶的静态网站配置

设置静态网站

功能说明

PUT Bucket website 用于为存储桶配置静态网站。

方法原型

```
public Guzzle\Service\Resource\Model PutBucketWebsite(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->putBucketWebsite(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在
```

```
COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'IndexDocument' => array(
'Suffix' => 'index.html',
),
'RedirectAllRequestsTo' => array(
'Protocol' => 'https',
),
'ErrorDocument' => array(
'Key' => 'Error.html',
),
'RoutingRules' => array(
array(
'Condition' => array(
'HttpErrorCodeReturnedEquals' => '405',
),
'Redirect' => array(
'Protocol' => 'https',
'ReplaceKeyWith' => '404.html',
),
),
// ... repeated
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	父节点	描述	类型	是否必选
Bucket	无	设置静态网站的存储桶，格式为 BucketName-APPID，详情请参见 命名规范	String	是
IndexDocument	无	索引文档	Array	是
Suffix	IndexDocument	指定索引文档	String	是
ErrorDocument	无	错误文档	Array	否
Key	ErrorDocument	指定通用错误返回	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	父节点	描述	类型	是否必选
RedirectAllRequestsTo	无	重定向所有请求	Array	否
Protocol	RedirectAllRequestsTo	指定全站重定向的协议，只能设置为 https	String	否
RoutingRules	无	设置重定向规则，最多设置100条 RoutingRule	Array	否
RoutingRule	RoutingRules	设置单条重定向规则，包括前缀匹配重定向和错误码重定向	Array	否
Condition	RoutingRule	指定重定向发生的条件，前缀匹配重定向和错误码重定向只能指定一个	Array	否
HttpErrorCodeReturnedEquals	Condition	指定重定向错误码，只支持配置4XX返回码，优先级高于 ErrorDocument	Interger	否
KeyPrefixEquals	Condition	指定前缀重定向的路径，替换指定的 folder/	String	否
Redirect	RoutingRule	指定满足重定向 conditon 时重定向的具体替换规则	Array	否
ReplaceKeyWith	Redirect	替换整个 Key 为指定的内容	String	否
ReplaceKeyPrefixWith	Redirect	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String	否

查询静态网站配置

功能说明

GET Bucket website 用于查询与存储桶关联的静态网站配置信息。

版权所有：腾讯云计算（北京）有限责任公司

方法原型

```
public Guzzle\Service\Resource\Model GetBucketWebsite(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->getBucketWebsite(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	描述	类型
bucket	查询静态网站配置的存储桶，格式为 BucketName-APPID，详情请参见 命名规范	String

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[RedirectAllRequestsTo] => Array
(
```

```
[Protocol] => https
)
[IndexDocument] => Array
(
[Suffix] => index.html
)
[ErrorDocument] => Array
(
[Key] => Error.html
)
[RoutingRules] => Array
(
[0] => Array
(
[Condition] => Array
(
[HttpErrorCodeReturnedEquals] => 405
)
[Redirect] => Array
(
[Protocol] => https
[ReplaceKeyWith] => 404.html
)
)
)
[RequestId] => NWRmMzQ3YjlfMTlhYTk0MGFfNzMzYl84YWIy****
)
```

返回结果说明

参数名称	描述	类型
Bucket	存储桶名称格式为 BucketName-APPID ，详情请参见 命名规范	String
IndexDocument	索引文档	Array
Suffix	指定索引文档	String
ErrorDocument	错误文档	Array
Key	指定通用错误返回	String
RedirectAllRequestsTo	重定向所有请求	Array
Protocol	指定全站重定向的协议，只能设置为 https	String
RoutingRules	设置重定向规则，最多设置100条 RoutingRule	Array

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
RoutingRule	设置单条重定向规则，包括前缀匹配重定向和错误码重定向	Array
Condition	指定重定向发生的条件，前缀匹配重定向和错误码重定向只能指定一个	Array
HttpErrorCodeReturnedEquals	指定重定向错误码，只支持配置4XX返回码，优先级高于ErrorDocument	Interger
KeyPrefixEquals	指定前缀重定向的路径，替换指定的 folder/	String
Redirect	指定满足重定向 conditon 时重定向的具体替换规则	Array
ReplaceKeyWith	替换整个 Key 为指定的内容	String
ReplaceKeyPrefixWith	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String

删除静态网站配置

功能说明

DELETE Bucket website 用于删除存储桶中的静态网站配置。

方法原型

```
public Guzzle\Service\Resource\Model DeleteBucketWebsite(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
```

```
'secretKey' => $secretKey)));

try {
$result = $cosClient->deleteBucketWebsite(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	描述	类型
bucket	被删除静态网站配置的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String

清单

最近更新时间：2021-05-26 15:49:48

简介

本文档提供关于清单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket inventory	设置清单任务	设置存储桶的清单任务
GET Bucket inventory	查询清单任务	查询存储桶的清单任务
DELETE Bucket inventory	删除清单任务	删除存储桶的清单任务

设置清单任务

功能说明

PUT Bucket inventory 用于在存储桶中创建清单任务。

方法原型

```
public Guzzle\Service\Resource\Model PutBucketInventory(array $args = array());
```

请求示例

```
try {
$result = $cosClient->putBucketInventory(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Id' => 'string',
'Destination' => array(
'COSBucketDestination'=>array(
'Format' => 'CSV',
'AccountId' => '100000000001',
'Bucket' => 'qcs::cos:ap-chengdu::examplebucket-1250000000',
'Prefix' => 'string',
)
),
'IsEnabled' => 'True',
'Schedule' => array(
```

```
'Frequency' => 'Daily',
),
'Filter' => array(
'Prefix' => 'string',
),
'IncludedObjectVersions' => 'Current',
'OptionalFields' => array(
'Size',
'ETag',
)
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	父节点	描述	类型	是否必选
Bucket	无	设置清单任务的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String、Array	是
Id	无	清单的名称，与请求参数中的 id 对应	Array	是
IsEnabled	无	清单是否启用的标识： • 如果设置为 true，清单功能将生效 • 如果设置为 false，将不生成任何清单	String	是
IncludedObjectVersions	无	是否在清单中包含对象版本： • 如果设置为 All，清单中将会包含所有对象版本，并在清单中增加 VersionId，IsLatest，DeleteMarker 这几个字段 • 如果设置为 Current，则清单中不包含对象版本信息	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	父节点	描述	类型	是否必选
Filter	无	筛选待分析对象。清单功能将分析符合 Filter 中设置的前缀的对象	Array	否
Prefix	Filter	需要分析的对象的前缀	String	否
OptionalFields	无	清单结果中可选包含的分析项目名称，可选字段包括：Size，LastModifiedDate，StorageClass，ETag，IsMultipartUploaded，ReplicationStatus	Array	否
Schedule	无	配置清单任务周期	Array	是
Frequency	Schedule	清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly	String	是
Destination	无	描述存放清单结果的信息	Array	是
COSBucketDestination	Destination	清单结果导出后存放的存储桶信息	Array	是
Bucket	COSBucketDestination	清单分析结果的存储桶名	String	是
AccountId	COSBucketDestination	存储桶的所有者 UIN，例如 100000000001	String	否
Prefix	COSBucketDestination	清单分析结果的前缀	String	否
Format	COSBucketDestination	清单分析结果的文件形式，可选项为 CSV 格式	String	是
Encryption	COSBucketDestination	为清单结果提供服务端加密的选项	Array	否
SSE-COS	Encryption	COS 托管密钥的加密方式，无需填充	String	否

其他清单配置参数请参考API文档。

错误码说明

该请求可能会发生的一些常见的特殊错误如下：

错误码	描述	状态码
InvalidArgument	不合法的参数值	HTTP 400 Bad Request
TooManyConfigurations	清单数量已经达到1000条的上限	HTTP 400 Bad Request
AccessDenied	未授权的访问。您可能不具备访问该存储桶的权限	HTTP 403 Forbidden

查询清单任务

功能说明

GET Bucket inventory 接口用于查询存储桶中用户的清单任务信息。

方法原型

```
public Guzzle\Service\Resource\Model GetBucketInventory(array $args = array());
```

请求示例

```
try {
$result = $cosClient->getBucketInvnetory(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Id' => 'string',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	描述	类型
bucket	查询清单任务的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	bucket
Id	清单任务的名称，合法字符：a-z，A-Z，0-9，-，_，.	String

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[Destination] => Array
(
[COSBucketDestination] => Array
(
[Format] => CSV
[AccountId] => 100000000001
[Bucket] => qcs::cos:ap-chengdu::examplebucket-1250000000
[Prefix] => String
)
)
[Schedule] => Array
(
[Frequency] => Daily
)
[OptionalFields] => Array
(
[0] => Size
[1] => ETag
)
[IsEnabled] => true
[Id] => string
[IncludedObjectVersions] => Current
[RequestId] => NWRmMzQwMDVfMjNiMjU4NjRfOGQ4MV9iN2Jk****
)
```

返回结果说明

参数名称	描述	类型
Bucket	存储桶名称格式为 BucketName-APPID ， 详情请参见 命名规范	String、Array
Id	清单的名称，与请求参数中的 id 对应	Array
IsEnabled	清单是否启用的标识： • 如果设置为 true，清单功能将生效 • 如果设置为 false，将不生成任何清单	String
IncludedObjectVersions	是否在清单中包含对象版本： • 如果设置为 All，清单中将会包含所有对象版本，并在清单中增加 VersionId，IsLatest，DeleteMarker 这几个字段 • 如果设置为 Current，则清单中不包含对象版本信息	String
Filter	筛选待分析对象。清单功能将分析符合 Filter 中设置的前缀的对象	Array

参数名称	描述	类型
Prefix	需要分析的对象的前缀	String
OptionalFields	清单结果中可选包含的分析项目名称，可选字段包括：Size，LastModifiedDate，StorageClass，ETag，IsMultipartUploaded，ReplicationStatus	Array
Schedule	配置清单任务周期	Array
Frequency	清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly	String
Destination	描述存放清单结果的信息	Array
COSBucketDestination	清单结果导出后存放的存储桶信息	Array
Bucket	清单分析结果的存储桶名	String
AccountId	存储桶的所有者 UIN，例如100000000001	String
Prefix	清单分析结果的前缀	String
Format	清单分析结果的文件形式	String
Encryption	为清单结果提供服务端加密的选项	Array
SSE-COS	COS 托管密钥的加密方式，无需填充	String

删除清单任务

功能说明

DELETE Bucket inventory 删除存储桶中指定的清单任务。

方法原型

```
public Guzzle\Service\Resource\Model DeleteBucketInventory(array $args = array())
;
```

请求示例

```
try {
$result = $cosClient->deleteBucketInvnetory(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Id' => 'string',
```

```
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	描述	类型
bucket	存储桶名称的格式为 BucketName-APPID ， 详情请参见 命名规范	String
Id	清单任务的名称，合法字符：a-z，A-Z，0-9，-，_，.	String

访问管理
跨域访问

最近更新时间：2022-01-05 14:36:26

简介

本文档提供关于跨域访问的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域名访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域名访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域名访问配置信息

设置跨域配置

功能说明

设置指定存储桶的跨域名访问配置信息（PUT Bucket cors）。

方法原型

```
public Guzzle\Service\Resource\Model putBucketCors(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
```

```
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->putBucketCors(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'CORSRules' => array(
array(
'AllowedHeaders' => array('*',),
'AllowedMethods' => array('PUT', ),
'AllowedOrigins' => array('*', ),
'ExposeHeaders' => array('*', ),
'MaxAgeSeconds' => 1,
),
// ... repeated
)
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	设置跨域配置的存储桶，格式：BucketName-APPID	是
CORSRules	Array	跨域信息列表	是
CORSRule	Array	跨域信息	是
AllowedMethods	String	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	是
AllowedOrigins	String	允许的访问来源，支持通配符 `*`，格式：`协议://域名[:端口]`，例如 `http://www.qq.com`	是
AllowedHeaders	String	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 `*`	否
ExposeHeaders	String	设置浏览器可以接收到来自服务器端的自定义头部信息	否
MaxAgeSeconds	Int	设置 OPTIONS 请求得到结果的有效期	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	是否必填
ID	String	配置规则的 ID	是

查询跨域配置

功能说明

查询指定存储桶的跨域名访问配置信息（GET Bucket cors）。

方法原型

```
public Guzzle\Service\Resource\Model getBucketCors(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->getBucketCors(array(
'Bucket' => 'examplebucket-1250000000' //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	查询跨域配置的存储桶，格式：BucketName-APPID	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[data:protected] => Array
(
[CORSRules] => Array
(
[0] => Array
(
[ID] => 1234
[AllowedHeaders] => Array
(
[0] => *
)
[AllowedMethods] => Array
(
[0] => PUT
)
[AllowedOrigins] => Array
(
[0] => http://www.qq.com
)
)
)
[RequestId] => NWE3YzhkMmRfMTdiMjk0MGFfNTQzZl8xNWUw****
)
)
```

返回结果说明

参数名称	类型	描述	父节点
CORSRules	Array	跨域信息列表	无
CORSRule	Array	跨域信息	CORSRules
AllowedMethods	String	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	CORSRule

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	父节点
AllowedOrigins	String	允许的访问来源，支持通配符 `*`，格式：`协议://域名[:端口]`，例如 `http://www.qq.com`	CORSRule
AllowedHeaders	String	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 `*`	CORSRule
ExposeHeaders	String	设置浏览器可以接收到来自服务器端的自定义头部信息	CORSRule
MaxAgeSeconds	Int	设置 OPTIONS 请求得到结果的有效期	CORSRule
ID	String	配置规则的 ID	CORSRule

删除跨域配置

功能说明

删除指定存储桶的跨域名访问配置（DELETE Bucket cors）。

方法原型

```
public Guzzle\Service\Resource\Model deleteBucketCors(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->deleteBucketCors(array(
```

```
'Bucket' => 'examplebucket-1250000000' //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	被删除跨域配置的存储桶，格式：BucketName-APPID	是

自定义域名

最近更新时间：2022-01-05 14:36:26

简介

本文档提供关于自定义域名的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket domain	设置自定义域名	设置存储桶的自定义域名信息
GET Bucket domain	查询自定义域名	查询存储桶的自定义域名信息

设置自定义域名

功能说明

PUT Bucket domain 用于为存储桶配置自定义域名。

方法原型

```
public Guzzle\Service\Resource\Model PutBucketDomain(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
```

```
$result = $cosClient->putBucketDomain(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'DomainRules' => array(
array(
'Name' => 'www.qq.com',
'Status' => 'ENABLED',
'Type' => 'REST',
'ForcedReplacement' => 'CNAME',
),
// ... repeated
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	描述	类型
Bucket	设置自定义域名的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String
Name	自定义域名	String
Status	域名上线状态，可选值有 ENABLED、DISABLED	String
Type	绑定的源站类型，可选值有 REST、WEBSITE	String
ForcedReplacement	强制覆盖已存在的配置，可选值有 CNAME、TXT	String

返回错误码说明

该请求可能会发生的一些常见的特殊错误如下：

状态码	说明
HTTP 409 Conflict	该域名记录已存在，且请求中没有设置强制覆盖。或者该域名记录不存在，且请求中设置了强制覆盖
HTTP 451 Unavailable For Legal Reasons	该域名是中国境内域名，并且没有备案

版权所有：腾讯云计算（北京）有限责任公司

查询自定义域名

功能说明

GET Bucket domain 用于查询存储桶的自定义域名信息。

方法原型

```
public Guzzle\Service\Resource\Model GetBucketDomain(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->getBucketDomain(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	描述	类型
Bucket	查询自定义域名的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[DomainRules] => Array
(
[0] => Array
(
[Status] => ENABLED
[Name] => www.qq.com
[Type] => REST
)
)
[DomainTxtVerification] => tencent-cloud-cos-verification=9d2258433b1f38c7dd4b29fe272d2128
)
```

返回结果说明

参数名称	描述	类型
Name	自定义域名	String
Status	域名上线状态，可选值有 ENABLED、DISABLED	String
Type	绑定的源站类型，可选值有 REST、WEBSITE	String
ForcedReplacement	强制覆盖已存在的配置，可选值有 CNAME、TXT	String
DomainTXTVerification	域名校验信息，该字段是一个 MD5 校验值，原串格式为：`cos[Region][BucketName-APPID][BucketCreateTime]`，其中 Region 为存储桶所在地域，BucketCreateTime 为存储桶 GMT 创建时间	String

版权所有：腾讯云计算（北京）有限责任公司

访问控制

最近更新时间：2022-01-05 14:36:26

简介

本文档提供关于存储桶、对象的访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

存储桶 ACL

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶的访问权限控制列表（ACL）
GET Bucket acl	查询存储桶 ACL	获取指定存储桶的访问权限控制列表（ACL）

对象 ACL

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置 Bucket 中某个 Object （文件/对象）的 ACL
GET Object acl	查询对象 ACL	查询 Object（文件/对象）的 ACL

存储桶 ACL

设置存储桶 ACL

功能说明

设置指定存储桶的访问权限控制列表（ACL）。

方法原型

```
public Guzzle\Service\Resource\Model putBucketAcl(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
```

```php
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->putBucketAcl(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'ACL' => 'private',
'Grants' => array(
array(
'Grantee' => array(
'DisplayName' => 'qcs::cam::uin/100000000001:uin/100000000001',
'ID' => 'qcs::cam::uin/100000000001:uin/100000000001',
'Type' => 'CanonicalUser',
),
'Permission' => 'FULL_CONTROL',
),
// ... repeated
),
'Owner' => array(
'DisplayName' => 'qcs::cam::uin/100000000001:uin/100000000001',
'ID' => 'qcs::cam::uin/100000000001:uin/100000000001',
)));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Grants	Array	ACL权限列表	无

参数名称	类型	描述	父节点
Grant	Array	ACL权限信息	Grants
Grantee	Array	ACL权限信息	Grant
Type	String	所有者权限类型	Grantee
Permission	String	权限类型，可选值： FULL_CONTROL 、 WRITE 、 READ	Grant
ACL	String	整体权限类型，可选值：private 、 public-read 、public-read-write	无
Owner	String	存储桶所有者信息	无
DisplayName	String	权限所有者的名字信息	Grantee/Owner
ID	String	权限所有者 ID	Grantee/Owner

查询存储桶 ACL

功能说明

获取指定存储桶的访问权限控制列表（ACL）。

方法原型

```
public Guzzle\Service\Resource\Model getBucketAcl(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

```
try {
$result = $cosClient->getBucketAcl(array(
'Bucket' => 'examplebucket-1250000000' //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

返回结果示例

```
Array
(
[data:protected] => Array
(
[Owner] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => qcs::cam::uin/100000000001:uin/100000000001
)
[Grants] => Array
(
[0] => Array
(
[Grantee] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => qcs::cam::uin/100000000001:uin/100000000001
)
[Permission] => FULL_CONTROL
)
)
[RequestId] => NWE3YzhjMTRfYzdhMzNiMGFfYjdiOF8yYzZmMzU=
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Grants	Array	ACL 权限列表	无

参数名称	类型	描述	父节点
Grant	Array	ACL 权限信息	Grants
Grantee	Array	ACL 权限信息	Grant
Permission	String	权限类型，可选值：FULL_CONTROL 、WRITE 、 READ	Grant
Owner	String	存储桶所有者信息	无
DisplayName	String	权限所有者的名字信息	Grantee/Owner
ID	String	权限所有者 ID	Grantee/Owner

对象 ACL

设置对象 ACL

功能说明

设置指定对象访问权限控制列表（ACL）（PUT Object acl）。

方法原型

```
public Guzzle\Service\Resource\Model putObjectAcl(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->putObjectAcl(array(
```

```
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'ACL' => 'private',
'Grants' => array(
array(
'Grantee' => array(
'DisplayName' => 'qcs::cam::uin/100000000001:uin/100000000001',
'ID' => 'qcs::cam::uin/100000000001:uin/100000000001',
'Type' => 'CanonicalUser',
),
'Permission' => 'FULL_CONTROL',
),
// ... repeated
),
'Owner' => array(
'DisplayName' => 'qcs::cam::uin/100000000001:uin/100000000001',
'ID' => 'qcs::cam::uin/100000000001:uin/100000000001',
)));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
Grants	Array	ACL权限列表	否
Grant	Array	ACL权限信息	否
Grantee	Array	ACL权限信息	否
Type	String	所有者权限类型	否
Permission	String	权限类型，可选值：FULL_CONTROL、WRITE、READ	否
ACL	String	整体权限类型，可选值：private、 public-read	否
Owner	String	存储桶所有者信息	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	是否必填
DisplayName	String	权限所有者的名字信息	否
ID	String	权限所有者 ID	否

查询对象 ACL

功能说明

查询指定对象的访问权限控制列表（GET Object acl）。

方法原型

```
public Guzzle\Service\Resource\Model getObjectAcl(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->getObjectAcl(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
```

```
echo($e);
}
```

返回结果示例

```
Array
(
[data:protected] => Array
(
[Owner] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => qcs::cam::uin/100000000001:uin/100000000001
)
[Grants] => Array
(
[0] => Array
(
[Grantee] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => qcs::cam::uin/100000000001:uin/100000000001
)
[Permission] => FULL_CONTROL
)
)
[RequestId] => NWE3YzhjMTRfYzdhMzNiMGFfYjdiOF8yYzZmMzU=
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Grants	Array	ACL权限列表	无
Grant	Array	ACL权限信息	Grants
Grantee	Array	ACL权限信息	Grant
Permission	String	权限类型，可选值：FULL_CONTROL 、WRITE 、 READ	Grant
Owner	String	存储桶所有者信息	无
DisplayName	String	权限所有者的名字信息	Grantee / Owner

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	父节点
ID	String	权限所有者 ID	Grantee / Owner

版权所有：腾讯云计算（北京）有限责任公司

防盗链

最近更新时间：2022-01-05 14:36:27

简介

本文档提供关于存储桶 Referer 白名单或者黑名单的 API 概览以及 SDK 示例代码。

> 注意：
> 需要 COS PHP SDK v2.3.2 及以上版本。旧版本可能存在 bug，使用时建议升级到 最新版本。

API	操作名	操作描述
PUT Bucket referer	设置存储桶 Referer	设置存储桶 Referer 白名单或者黑名单
GET Bucket referer	查询存储桶 Referer	查询存储桶 Referer 白名单或者黑名单

设置存储桶 Referer

功能说明

设置存储桶 Referer 白名单或者黑名单（PUT Bucket referer）。

方法原型

```
public Guzzle\Service\Resource\Model putBucketReferer(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
```

```
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->putBucketReplication(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Status' => 'Enabled', //是否开启防盗链，枚举值：Enabled、Disabled
'RefererType' => 'White-List', //防盗链类型，枚举值：Black-List、White-List
'DomainList' => array(
'Domains' => array(
'*.qq.com', //单条生效域名
'*.qcloud.com', //单条生效域名
)
), //生效域名列表
'EmptyReferConfiguration' => 'Allow', //是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny
),
);
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Status	是否开启防盗链，枚举值：Enabled、Disabled	String	是
RefererType	防盗链类型，枚举值：Black-List、White-List	String	是
DomainList	生效域名列表	Array	是
EmptyReferConfiguration	是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny	String	否

查询存储桶 Referer

功能说明

查询存储桶 Referer 白名单或者黑名单（GET Bucket referer）。

方法原型

```
public Guzzle\Service\Resource\Model getBucketReferer(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->getBucketReferer(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[RequestId] => NjE0NTk2ZWVfZmQzNDJjMGJfMWNiNmVfN2Y1MWQx
[ContentType] => application/xml
[ContentLength] => 260
[Status] => Enabled
[RefererType] => White-List
[EmptyReferConfiguration] => Allow
[DomainList] => Array
(
[Domain] => Array
(
[0] => *.qq.com
[1] => *.qcloud.com
)
)
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/
)
```

异常处理

最近更新时间：2022-01-05 14:36:27

简介

调用 SDK 接口请求对象存储（Cloud Object Storage，COS）服务失败时，例如返回码为4xx或者5xx，系统将抛出（Qcloud\Cos\Exception\ServiceResponseException）异常。

服务端异常

CosServerException 包含了服务端返回的状态码、requestid 和出错明细等。捕获异常后，建议对整个异常进行打印，异常包含了必须的排查因素。以下是异常成员变量的描述以及异常捕获示例：

成员	描述	类型
requestId	请求 ID，用于表示一个请求，对于排查问题十分重要	string
statusCode	response 的 status 状态码，更多详情请参见 错误码	string
errorCode	请求失败时 body 返回的 Error Code，更多详情请参见 错误码	string
errorMessage	请求失败时 body 返回的 Error Message，更多详情请参见 错误码	string

异常捕获示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

```
try {
$cosClient->listBuckets()
} catch (Qcloud\Cos\Exception\ServiceResponseException $e) {
$statusCode = $e->getStatusCode(); // 获取错误码
$errorMessage = $e->getMessage(); // 获取错误信息
$requestId = $e->getRequestId(); // 获取错误的 requestId
$errorCode = $e->getCosErrorCode(); // 获取错误名称
$request = $e->getRequest(); // 获取完整的请求
$response = $e->getResponse(); // 获取完整的响应
echo ($e);
} catch (\Exception $e) {
}
```

注意：
如您使用 Phar 方式或源码方式安装 SDK，返回的异常信息提示会更加准确。如您使用 Composer 方式安装 SDK，返回的异常信息如不符合您的需求，可通过修改 vendor/guzzlehttp/guzzle-services/src/SchemaValidator.php 来自定义一些错误信息。

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。
   COS 自助诊断工具
   点击自助诊断
   输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**。
4. 稍候片刻，便能看到相应的智能诊断结果。

设置访问域名（CDN/全球加速）

最近更新时间：2022-01-06 16:16:46

简介

本文档提供关于如何使用非默认域名请求对象存储（Cloud Object Storage，COS）服务。

相关参数说明

通过初始化参数，来控制请求域名，相关的参数说明如下：

参数名	参数描述	类型	是否必填
Domain	调用操作存储桶和对象的 API 时自定义请求域名。可以使用模板，如"{$Bucket}.cos.{$Region}.myqcloud.com"，即在调用 API 时会使用参数中传入的 Bucket 和 Region 进行替换。	String	否
Protocol	发请求时用的协议，可选项https:、http:，默认判断当前页面是http: 时使用http:，否则使用https:	String	否

默认 CDN 加速域名

关于如何开启默认加速域名请参考 开启默认 CDN 加速域名。

以下代码展示了如何使用默认加速域名访问 COS 服务。

```
$bucket = "examplebucket-1250000000";
$cosClient = new Qcloud\Cos\Client(
array(
'domain' => $bucket . '.file.myqcloud.com', //默认加速域名
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId,
'secretKey' => $secretKey
)
)
);
```

版权所有：腾讯云计算（北京）有限责任公司

自定义 CDN 加速域名

关于如何开启 CDN 自定义加速域名请参考 开启自定义 CDN 加速域名。

以下代码展示了如何使用自定义加速域名访问 COS 服务。

```
$cosClient = new Qcloud\Cos\Client(
array(
'domain' => 'example-cdn-domain.com', //默认加速域名
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId,
'secretKey' => $secretKey
)
)
);
```

自定义源站域名

关于如何设置自定义源站域名请参考 自定义源站域名。

以下代码展示了如何使用自定义源站域名访问 COS 服务。

```
$cosClient = new Qcloud\Cos\Client(
array(
'domain' => 'example-cos-domain.com', //默认加速域名
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId,
'secretKey' => $secretKey
)
)
);
```

全球加速域名

关于全球加速功能请参考 全球加速功能概述。

以下代码展示了如何使用全球加速域名访问 COS 服务。

版权所有：腾讯云计算（北京）有限责任公司

```php
//{$bucket}.cos.{$region}.myqcloud.com/key 转换为
//{$bucket}.cos.accelerate.myqcloud.com/key
$cosClient = new Qcloud\Cos\Client(
array(
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId,
'secretKey' => $secretKey
),
'allow_accelerate' => true // 指定 true，使用全球加速域名请求
)
);
```

图片处理
基础图片处理

最近更新时间：2022-01-23 14:47:38

简介

腾讯云对象存储（Cloud Object Storage，COS）集成了 数据万象（Cloud Infinite，CI）专业的一体化多媒体解决方案，涵盖以下图片处理功能，详情可见 图片处理概述。

服务	功能	说明
基础图片处理服务	缩放	等比缩放、设定目标宽高缩放等多种方式
	裁剪	普通裁剪、缩放裁剪、内切圆、人脸智能裁剪
	旋转	自适应旋转、普通旋转
	格式转换	格式转换、GIF 格式优化、渐进显示
	质量变换	针对 JPG 和 WEBP 图片进行质量变换
	高斯模糊	对图片进行模糊处理
	亮度	调节图片亮度
	对比度	调节图片对比度
	锐化	对图片进行锐化处理
	添加水印	图片水印、文字水印
	获取图片信息	基本信息、EXIF 信息、主色调
	去除元信息	包括 EXIF 信息
	快速缩略模板	快速实现图片格式转换、缩略、剪裁等功能，生成缩略图
	管道操作符	实现对图片按顺序进行多种处理

实例化 Template

版权所有：腾讯云计算（北京）有限责任公司

在使用基础图片处理功能时，首先实例化 Template 类，调用方法生成万象图片处理参数后，获取文件资源时传入，下面所有操作一致，不再重复说明。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$imageMogrTemplate = new Qcloud\Cos\ImageParamTemplate\ImageMogrTemplate();//创建基础图片处理参数模版实例
$imageMogrTemplate->thumbnailByScale(50);//指定图片的宽高为原图的 50%
$result = $cosClient->getObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'ImageHandleParam' => $imageMogrTemplate->queryString(),//生成基础图片处理参数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	是否必填
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
ImageHandleParam	String	万象图片处理参数，例如：imageMogr2/thumbnail/!50p	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Body] =>
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[ContentLength] => 100
[CacheControl] => max-age=2592000
[ContentType] => image/jpeg
[Key] => exampleobject
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Body	File/String	下载内容	无
RequestId	String	请求 ID 标识	无
CacheControl	String	缓存策略，设置 Cache-Control	无
ContentType	String	内容类型，设置 Content-Type	无
ContentLength	Int	响应体长度	无
Key	String	对象键	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	父节点
Location	String	请求资源地址	无

缩放

按百分比缩放

```
$imageMogrTemplate->thumbnailByScale(50);//指定图片的宽高为原图的 50%
$imageMogrTemplate->thumbnailByWidthScale(50);//指定图片的宽为原图的 50%，高度不变
$imageMogrTemplate->thumbnailByHeightScale(50);//指定图片的高为原图的 50%，宽度不变
```

指定宽高缩放

```
$imageMogrTemplate->thumbnailByWidth(50);//指定目标图片宽度为50，高度等比压缩
$imageMogrTemplate->thumbnailByHeight(50);//指定目标图片高度为50，宽度等比压缩
$imageMogrTemplate->thumbnailByMaxWH(50, 50);//限定缩略图的宽度和高度的最大值分别为50和50，进行等比缩放
$imageMogrTemplate->thumbnailByMinWH(50, 50);//限定缩略图的宽度和高度的最小值分别为50和50，进行等比缩放
$imageMogrTemplate->thumbnailByWH(50, 50);//忽略原图宽高比例，指定图片宽度为50，高度为50 ，强行缩放图片，可能导致目标图片变形
```

等比缩放

```
$imageMogrTemplate->thumbnailByPixel(1000);//等比缩放图片，缩放后的图像，总像素数量不超过1000
```

裁剪

普通裁剪

```
$imageMogrTemplate->cut(100, 300, 30, 30);//指定目标图片宽度、高度、相对于图片左上顶点水平向右偏移、相对于图片左上顶点水平向下偏移进行裁剪
```

缩放裁剪

方位图说明，请参见 九宫格方位图。

```
$imageMogrTemplate->cropByWidth(100);//按照指定目标宽度进行缩放裁剪
$imageMogrTemplate->cropByWidth(100, 'center');//按照指定目标宽度进行缩放裁剪，指定操作的起点位置
$imageMogrTemplate->cropByHeight(100);//按照指定目标高度进行缩放裁剪
$imageMogrTemplate->cropByHeight(100, 'center');//按照指定目标高度进行缩放裁剪，指定操作的起点位置
$imageMogrTemplate->cropByWH(100, 100);//按照指定目标宽度和高度进行缩放裁剪
$imageMogrTemplate->cropByWH(100, 100, 'center');//按照指定目标宽度和高度进行缩放裁剪，指定操作的起点位置
```

内切圆裁剪

radius 是内切圆的半径，取值范围为大于0且小于原图最小边一半的整数。内切圆的圆心为图片的中心。图片格式为 GIF 时，不支持该操作。

```
$imageMogrTemplate->iradius(100);//指定半径100为例
```

圆角裁剪

radius 为图片圆角边缘的半径，取值范围为大于0且小于原图最小边一半的整数。圆角与原图边缘相切。图片格式为 GIF 时，不支持该操作。

```
$imageMogrTemplate->rradius(1000);// 指定圆角半径100为例
```

人脸智能裁剪

基于图片中的人脸位置进行缩放裁剪。目标图片的宽度为 Width、高度为 Height。

```
$imageMogrTemplate->scrop(100, 100);// 裁剪人脸并宽高指定100缩放为例；
```

旋转

普通旋转

图片顺时针旋转角度，取值范围0 - 360，默认不旋转。

```
$imageMogrTemplate->rotate(45);//以旋转45度为例
```

自适应旋转

根据原图 EXIF 信息将图片自适应旋转回正。

```
$imageMogrTemplate->autoOrient();
```

格式转换

普通转换

目标缩略图的图片格式可为：jpg，bmp，gif，png，webp，yjpeg 等，其中 yjpeg 为针对 jpeg 格式进行的优化，本质为 jpg 格式；缺省为原图格式。

```
$imageMogrTemplate->format('png');//格式转换png
```

GIF 格式优化

该功能只针对原图为 gif 格式，对 gif 图片格式进行的优化，降帧降颜色。分为以下两种情况：

- FrameNumber=1，则按照默认帧数30处理，如果图片帧数大于该帧数则截取。
- FrameNumber 取值(1,100]，则将图片压缩到指定帧数 （FrameNumber）。

```
$imageMogrTemplate->gifOptimization(50);//压缩到指定帧数50
```

输出渐进式 JPG 格式

该操作仅在输出图片格式为 jpg 格式时有效。如果输出非 jpg 图片格式，会忽略该参数。取值范围0或1，0：表示不开启渐进式；1：开启渐进式，默认值0。

```
$imageMogrTemplate->jpegInterlaceMode(1);//开启渐进式。
```

质量变换

质量变换功能提供三种变换类型：绝对变换、相对变换、最低质量变换，可调节图片质量。该功能仅适用于 JPG 和 WEBP 格式的图片。

绝对变换

设置图片的绝对质量，取值范围0 - 100，默认值为原图质量；取原图质量和指定质量的最小值。
是否强制使用指定值，取值范围0或1，0：表示不强制使用；1：表示强制使用，默认值0。

版权所有：腾讯云计算（北京）有限责任公司

```
$imageMogrTemplate->quality(90, 1);//强制指定质量90
```

相对变换

设置图片的相对质量，取值范围0 - 100，数值以原图质量为标准。例如原图质量为80，将 rquality 设置为80后，得到处理结果图的图片质量为64（80x80%）。

```
$imageMogrTemplate->relativelyQuality(80);
```

最低质量变换

图片的最低质量，取值范围为0 - 100，设置结果图的质量参数最小值。

- 例如原图质量为85，将 lquality 设置为80后，处理结果图的图片质量为85。
- 例如原图质量为60，将 lquality 设置为80后，处理结果图的图片质量会被提升至80。

```
$imageMogrTemplate->lowestQuality(90);
```

高斯模糊

高斯模糊功能，模糊半径，取值范围为1 - 50；正态分布的标准差，必须大于0。图片格式为 gif 时，不支持该参数。

```
$imageMogrTemplate->blur(20, 20);// 模糊半径20，正态分布的标准差20为例
```

亮度

图片亮度调节功能，取值范围为[-100, 100]的整数。

```
//取值＜0：降低图片亮度。
//取值 = 0：不调整图片亮度。
//取值＞0：提高图片亮度。
$imageMogrTemplate->bright(50);// 提高图片亮度50
```

对比度

图片对比度调节功能，取值范围为[-100, 100]的整数。

```
//取值＜0：降低图片对比度。
//取值 = 0：不调整图片对比度。
//取值＞0：提高图片对比度。
$imageMogrTemplate->contrast(50);// 提高图片对比度50
```

锐化

图片锐化功能，取值范围为10 - 300间的整数，推荐使用70。参数值越大，锐化效果越明显。

```
$imageMogrTemplate->sharpen(70);// 以锐化值70为例
```

添加水印

方位图说明，请参见 九宫格方位图。

图片水印

指定的水印图片必须同时满足如下3个条件：

1. 水印图片与源图片必须位于同一个存储桶下。
2. URL 需使用 COS 域名（不能使用 CDN 加速域名，例如 examplebucket-1250000000.file.myqcloud.com/shuiyin_2.png 不可用），且需保证水印图可访问。如果水印图读取权限为私有，则需要携带有效签名。
3. URL 必须以 `http://` 开始，不能省略 HTTP 头，也不能填 HTTPS 头，例如 `examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/shuiyin_2.png`，`https://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/shuiyin_2.png` 为非法的水印 URL。

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
```

```
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$imageWatermarkTemplate = new Qcloud\Cos\ImageParamTemplate\ImageWatermarkTemplate();//创建图片水印参数模版实例
$imageWatermarkTemplate->setImage("imageurl");//设置水印图片地址
$imageWatermarkTemplate->setGravity('center');//设置水印九宫格位置
$imageWatermarkTemplate->setDx(10);//设置水平（横轴）边距
$imageWatermarkTemplate->setDy(10);//设置垂直（纵轴）边距
$imageWatermarkTemplate->setBlogo(1);//设置水印图适配功能，适用于水印图尺寸过大的场景（如水印墙）共有两种类型：设置为1时，水印图会被缩放至与原图相似大小后添加；设置为2时，水印图会被直接裁剪至与原图相似大小后添加。
$imageWatermarkTemplate->setScatype(1);//根据原图大小调整水印图大小，1:按宽缩放，2:按高缩放，3:按面积缩放
$imageWatermarkTemplate->setSpcent(100);//设置水印图片占原图的千分比为100/1000,取值范围为[1,1000]，默认为原水印图大小
$result = $cosClient->getObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'ImageHandleParam' => $imageWatermarkTemplate->queryString(),//生成图片水印参数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

文字水印

COS 提供实时文字水印处理功能。水印字体列表可参见 支持字体列表。字体颜色列表可参见 RGB 编码表。

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
```

```
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$textWatermarkTemplate = new Qcloud\Cos\ImageParamTemplate\TextWatermarkTemplate
();//创建文字水印参数模版实例
$textWatermarkTemplate->setText("test");//设置水印内容
$textWatermarkTemplate->setFont("tahoma.ttf");//设置水印字体,默认值 tahoma.ttf
$textWatermarkTemplate->setFontsize(30);//设置水印文字字体大小，单位为磅，缺省值13
$textWatermarkTemplate->setFill("#3D3D3D");//设置字体颜色，需设置为十六进制 RGB 格式
（例如 #FF0000），默认值为 #3D3D3D(灰色)
$textWatermarkTemplate->setDissolve(90);//设置文字透明度，取值1 - 100 ，默认90（完全不
透明）
$textWatermarkTemplate->setGravity('center');//设置文字水印位置，九宫格位置（参见九宫格
方位图），默认值 southeast
$textWatermarkTemplate->setDx(10);//设置水平（横轴）边距，单位为像素，缺省值为0
$textWatermarkTemplate->setDy(10); //设置垂直（纵轴）边距，单位为像素，默认值为0
$textWatermarkTemplate->setBatch(1);//设置平铺水印功能，可将文字水印平铺至整张图片。当设
置为1时，开启平铺水印功能
$textWatermarkTemplate->setDegree(45);//设置文字水印的旋转角度设置，取值范围为0 - 360,
默认0
$result = $cosClient->getObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在
COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'ImageHandleParam' => $textWatermarkTemplate->queryString(),//生成文字水印参数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

获取图片信息

获取图片基本信息

查询图片基本信息。

示例代码

```php
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->ImageInfo(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
```

版权所有：腾讯云计算（北京）有限责任公司

```
(
[Body] =>
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[ContentLength] => 100
[ContentType] => application/json
[Key] => exampleobject
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
[Data] => {"format": "jpeg", "width": "400", "height": "400"}
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Body	File/String	返回体	无
RequestId	String	请求 ID 标识	无
ContentType	String	内容类型，设置 Content-Type	无
ContentLength	Int	响应体长度	无
Key	String	对象键	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无
Data	Json/String	图片基本信息	无

获取图片 EXIF

获取图片 EXIF 信息。如图片无 exif 信息，将返回{"error" : "no exif data"}。

示例代码

```php
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
```

```
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->ImageExif(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Body] =>
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[ContentLength] => 32
[ContentType] => application/json
[Key] => exampleobject
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
[Data] => {"error" : "no exif data"}
```

```
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Body	File/String	返回体	无
RequestId	String	请求 ID 标识	无
ContentType	String	内容类型，设置 Content-Type	无
ContentLength	Int	响应体长度	无
Key	String	对象键	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无
Data	Json/String	图片 EXIF 信息	无

获取图片主色调

获取图片主色调信息。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->ImageAve(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在
```

```
COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Body] =>
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[ContentLength] => 19
[ContentType] => application/json
[Key] => exampleobject
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
[Data] => {"RGB": "0x483519"}
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Body	File/String	返回体	无

参数名称	类型	描述	父节点
RequestId	String	请求 ID 标识	无
ContentType	String	内容类型，设置 Content-Type	无
ContentLength	Int	响应体长度	无
Key	String	对象键	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无
Data	Json/String	图片主色调信息	无

去除元信息

去除图片元信息，包括 EXIF 信息。

```
$imageMogrTemplate->strip();//去除元信息
```

快速缩略模板

提供常用图片处理模板，生成相应的缩略图。

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
```

```
$imageViewTemplate = new Qcloud\Cos\ImageParamTemplate\ImageViewTemplate();//创建快速缩略模板参数模版实例
$imageViewTemplate->setMode(1);//设置缩略模式，取值范围[1，5]:1:限定缩略图的宽高最小值。该操作会将图像等比缩放直至某一边达到设定最小值，之后将另一边居中裁剪至设定值。若只指定一边，则表示宽高相等的正方形。2:限定缩略图的宽高最大值。该操作会将图像等比缩放至宽高都小于设定最大值。3:限定缩略图的宽高最小值。该操作会将图像等比缩放至宽高都大于设定最小值。4:限定缩略图的长边和短边的最小值，进行等比压缩；如果只指定一边，代表另外一边为同样的值。5:限定缩略图的长边和短边的最大值，进行等比压缩，居中裁剪；如果只指定一边，则表示宽高相等的正方形；缩放后其中一边多余的部分会被裁剪掉。
$imageViewTemplate->setWidth(400);//设置宽度
$imageViewTemplate->setHeight(600);//设置高度
$imageViewTemplate->setFormat();//设置目标缩略图的图片格式，可为:jpg，bmp，gif，png，webp，缺省为原图格式
$imageViewTemplate->setQuality(1, 85);//设置图片质量类型和图片质量，图片质量类型可为1:绝对质量，2:相对质量，3:最低质量。
$imageViewTemplate->setQuality(1, 85, 1);//若图片质量类型为绝对质量时，可强制使用指定值。
$result = $cosClient->getObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'ImageHandleParam' => $imageViewTemplate->queryString(),//生成快速缩略模板参数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

管道操作符

CIParamTransformation 支持多操作组。

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
```

```
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$ciParamTransformation = new Qcloud\Cos\ImageParamTemplate\CIParamTransformation();//创建管道参数模版实例
$ciParamTransformation = new Qcloud\Cos\ImageParamTemplate\CIParamTransformation("#");//若管道操作符不为"|",需创建实例时指定
$ciParamTransformation->addRule($imageMogrTemplate);//设置基础图片处理参数
$ciParamTransformation->addRule($imageWatermarkTemplate);//设置图片水印处理参数
$ciParamTransformation->addRule($textWatermarkTemplate);//设置文字水印处理参数
$ciParamTransformation->addRule($imageViewTemplate);//设置快速缩略模板参数
$result = $cosClient->getObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'ImageHandleParam' => $ciParamTransformation->queryString(),//生成管道拼接参数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

图片样式

最近更新时间：2022-01-04 12:30:50

简介

本文档提供关于图片样式相关的 API 概览以及 SDK 示例代码。

API	说明
增加样式	增加存储桶样式
查询样式	查询存储桶样式
删除样式	删除存储桶样式

增加样式

功能说明

对某一个存储桶设置样式功能，后续上传到该存储桶的图片文件都会被添加指定的样式。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->PutBucketImageStyle(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
```

```
'StyleName' => 'style_name',//样式名称
'StyleBody' => 'imageMogr2/thumbnail/!50px', //样式信息
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
StyleName	String	样式名称	是
StyleBody	String	样式信息	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.pic.ap-beijing.myqcloud.com/
)
)
```

返回结果说明

参数名称	类型	描述	父节点
RequestId	String	请求 ID 标识	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无

查询样式

版权所有：腾讯云计算（北京）有限责任公司

功能说明

查询某个存储桶下已有的样式。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->GetBucketImageStyle(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'StyleName' => 'style_name', //样式名称
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
StyleName	String	样式名称	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
```

版权所有：腾讯云计算（北京）有限责任公司

```
[structure:protected] =>
[data:protected] => Array
(
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.pic.ap-beijing.myqcloud.com/
[StyleRule] => Array(
[0] => Array(
[StyleName] => style_name
[StyleBody] => imageMogr2/thumbnail/!50px
)
)
)
)
```

返回结果说明

参数名称	类型	描述	父节点
RequestId	String	请求 ID 标识	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无
StyleRule	Array	样式信息列表	无

删除样式

功能说明

删除某一特定样式。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
```

```
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->DeleteBucketImageStyle(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'StyleName' => 'style_name', //样式名称
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
StyleName	String	样式名称	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.pic.ap-beijing.myqcloud.com/
)
)
```

返回结果说明

参数名称	类型	描述	父节点
RequestId	String	请求 ID 标识	无

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无

版权所有：腾讯云计算（北京）有限责任公司

图片持久化处理

最近更新时间：2022-01-04 12:30:51

简介

本文档提供关于图片持久化处理的 API 概览以及 SDK 示例代码。

API	说明
图片持久化处理	对象存储（Cloud Object Storage，COS）提供的上传时处理功能可以帮助使用者在上传时实现图片处理。此外能够对已存储在 COS 的图片进行相应处理操作，并将结果存入到 COS

上传时处理

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$imageMogrTemplate = new Qcloud\Cos\ImageParamTemplate\ImageMogrTemplate();//创建基础图片处理参数模板实例
$imageMogrTemplate->thumbnailByScale(50);//指定图片的宽高为原图的 50%
$picOperationsTemplate = new \Qcloud\Cos\ImageParamTemplate\PicOperationsTransformation();//创建图片持久化处理参数模版实例
$picOperationsTemplate->setIsPicInfo(1);//设置是否返回原图信息，0不返回原图信息，1返回原图信息，默认为0
$picOperationsTemplate->addRule($imageMogrTemplate, "resultobject");//设置图片处理
```

```
规则
$result = $cosClient->putObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Body' => fopen('path/to/localFile', 'rb'),
'PicOperations' => $picOperationsTemplate->queryString(),//生成图片持久化处理参数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
Body	File/String	上传的内容	是
PicOperations	Json/String	图片持久化处理信息	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Body] =>
[ETag] => "698d51a19d8a121ce581499d7b701668"
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[ContentLength] => 238186
[Key] => exampleobject
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
[Data] => Array
```

```
(
[OriginalInfo] => Array
(
[Key] => exampleobject
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
[ETag] => "7037fb6fb4cca43b958a28789605e73d98088720"
[ImageInfo] => Array
(
[Format] => JPEG
[Width] => 600
[Height] => 500
[Quality] => 90
[Ave] => 0x46442e
[Orientation] => 0
)
)
[ProcessResults] => Array
(
[Object] => Array
(
[0] => Array(
[Key] => resultobject
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/resultobject
[Format] => JPEG
[Width] => 300
[Height] => 200
[Size] => 30000
[Quality] => 90
[ETag] => "87c153bc2909aa0ba111ca126b675c510d36b817"
)
)
)
)
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Body	File/String	返回体	无
ETag	String	文件的 MD5 值	无
RequestId	String	请求 ID 标识	无

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	父节点
ContentLength	Int	响应体长度	无
Key	String	对象键	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无
Data	Array	图片处理结果信息	无

云上数据处理

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$imageMogrTemplate = new Qcloud\Cos\ImageParamTemplate\ImageMogrTemplate();//创建基础图片处理参数模版实例
$imageMogrTemplate->thumbnailByScale(50);//指定图片的宽高为原图的 50%
$picOperationsTemplate = new \Qcloud\Cos\ImageParamTemplate\PicOperationsTransformation();//创建图片持久化处理参数模版实例
$picOperationsTemplate->setIsPicInfo(1);//设置是否返回原图信息，0不返回原图信息，1返回原图信息，默认为0
$picOperationsTemplate->addRule($imageMogrTemplate, "resultobject");//设置图片处理规则
$result = $cosClient->ImageProcess(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
```

```
'Key' => 'exampleobject',
'PicOperations' => $picOperationsTemplate->queryString(),//生成图片持久化处理参数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
PicOperations	Json/String	图片持久化处理信息	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[Key] => exampleobject
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
[OriginalInfo] => Array
(
[Key] => exampleobject
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
[ETag] => "7037fb6fb4cca43b958a28789605e73d98088720"
[ImageInfo] => Array
(
[Format] => JPEG
[Width] => 600
[Height] => 500
[Quality] => 90
```

版权所有：腾讯云计算（北京）有限责任公司

```
[Ave] => 0x46442e
[Orientation] => 0
)
)
[ProcessResults] => Array
(
[Object] => Array
(
[0] => Array(
[Key] => resultobject
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/resultobject
[Format] => JPEG
[Width] => 300
[Height] => 200
[Size] => 30000
[Quality] => 90
[ETag] => "87c153bc2909aa0ba111ca126b675c510d36b817"
)
)
)
)
)
```

返回结果说明

参数名称	类型	描述	父节点
RequestId	String	请求 ID 标识	无
Key	String	对象键	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无
OriginalInfo	Array	源图片信息	无
ProcessResults	Array	图片处理结果信息	无

版权所有：腾讯云计算（北京）有限责任公司

盲水印

最近更新时间：2022-04-06 16:53:38

简介

本文档提供关于盲水印的相关的 API 概览以及 SDK 示例代码。

API	操作描述
盲水印	对本地图片添加或提取盲水印并上传至存储桶

添加盲水印

上传时添加

指定的水印图片必须同时满足如下3个条件：

1. 水印图片与源图片必须位于同一个存储桶下。
2. URL 需使用 COS 域名（不能使用 CDN 加速域名，例如 examplebucket-1250000000.file.myqcloud.com/shuiyin_2.png 不可用），且需保证水印图可访问。如果水印图读取权限为私有，则需要携带有效签名。
3. URL 必须以 `http://` 开始，不能省略 HTTP 头，也不能填 HTTPS 头，例如 `examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/shuiyin_2.png`，`https://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/shuiyin_2.png` 为非法的水印 URL。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
```

```
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$blindWatermarkTemplate = new Qcloud\Cos\ImageParamTemplate\BlindWatermarkTemplate();//创建盲水印参数模版实例
$blindWatermarkTemplate->setType(3);//盲水印类型，有效值：1 半盲；2 全盲；3 文字
$blindWatermarkTemplate->setImage("imageUrl");//设置盲水印图片地址
$blindWatermarkTemplate->setText("Test");//设置盲水印文字
$blindWatermarkTemplate->setLevel(3);//只对全盲水印（type=2）有效。level 的取值范围为{1,2,3}，默认值为1，level 值越大则图片受影响程度越大、盲水印效果越好。
$picOperationsTemplate = new \Qcloud\Cos\ImageParamTemplate\PicOperationsTransformation();//创建图片持久化处理参数模版实例
$picOperationsTemplate->setIsPicInfo(1);//设置是否返回原图信息，0不返回原图信息，1返回原图信息，默认为0
$picOperationsTemplate->addRule($blindWatermarkTemplate, "resultobject");//设置图片持久化处理参数
$result = $cosClient->putObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'Body' => fopen('path/to/localFile', 'rb'),
'PicOperations' => $picOperationsTemplate->queryString(),//生成图片持久化处理参数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
Body	File/String	上传的内容	是
PicOperations	Json/String	图片持久化处理信息	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Body] =>
[ETag] => "698d51a19d8a121ce581499d7b701668"
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[ContentLength] => 10000
[Key] => exampleobject
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
[Data] => Array
(
[OriginalInfo] => Array
(
[Key] => exampleobject
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
[ETag] => "7037fb6fb4cca43b958a28789605e73d98088720"
[ImageInfo] => Array
(
[Format] => JPEG
[Width] => 600
[Height] => 500
[Quality] => 90
[Ave] => 0x46442e
[Orientation] => 0
)
)
[ProcessResults] => Array
(
[Object] => Array
(
[0] => Array(
[Key] => resultobject
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/resultobject
[Format] => JPEG
[Width] => 300
[Height] => 200
[Size] => 30000
[Quality] => 90
[ETag] => "87c153bc2909aa0ba111ca126b675c510d36b817"
)
)
)
```

```
)
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Body	File/String	返回体	无
ETag	String	文件的 MD5 值	无
RequestId	String	请求 ID 标识	无
ContentLength	Int	响应体长度	无
Key	String	对象键	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无
Data	Array	图片处理结果信息	无

下载时添加

指定的水印图片必须同时满足如下3个条件：

1. 水印图片与源图片必须位于同一个存储桶下。
2. URL 需使用 COS 域名（不能使用 CDN 加速域名，例如 examplebucket-1250000000.file.myqcloud.com/shuiyin_2.png 不可用），且需保证水印图可访问。如果水印图读取权限为私有，则需要携带有效签名。
3. URL 必须以 `http://` 开始，不能省略 HTTP 头，也不能填 HTTPS 头，例如 `examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/shuiyin_2.png`，`https://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/shuiyin_2.png` 为非法的水印 URL。

示例代码

```php
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，htt
```

```
ps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$blindWatermarkTemplate = new Qcloud\Cos\ImageParamTemplate\BlindWatermarkTemplat
e();//创建盲水印参数模版实例
$blindWatermarkTemplate->setType(3);//盲水印类型，有效值：1 半盲；2 全盲；3 文字
$blindWatermarkTemplate->setImage("imageUrl");//设置盲水印图片地址
$blindWatermarkTemplate->setText("Test");//设置盲水印文字
$blindWatermarkTemplate->setLevel(3);//只对全盲水印（type=2）有效。level 的取值范围为
{1,2,3}，默认值为1，
$result = $cosClient->getObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在
COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'ImageHandleParam' => $blindWatermarkTemplate->queryString(),//生成盲水印参数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
ImageHandleParam	String	盲水印参数	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Body] =>
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[ContentLength] => 100
[CacheControl] => max-age=2592000
[ContentType] => image/jpeg
[Key] => exampleobject
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Body	File/String	下载内容	无
RequestId	String	请求 ID 标识	无
CacheControl	String	缓存策略，设置 Cache-Control	无
ContentType	String	内容类型，设置 Content-Type	无
ContentLength	Int	响应体长度	无
Key	String	对象键	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无

提取盲水印

图片地址，根据盲水印类型填写：
1.当半盲类型时，则盲水印图片地址必填，且为原图图片地址。
2.当全盲类型时，则盲水印图片地址必填，且为水印图地址。
3.当文字类型时，则盲水印图片地址无需填写（无效）。

指定的水印图片必须同时满足如下3个条件：

1. 水印图片与源图片必须位于同一个存储桶下。
2. URL 需使用 COS 域名（不能使用 CDN 加速域名，例如 examplebucket-1250000000.file.myqcloud.com/shuiyin_2.png 不可用），且需保证水印图可访问。如果水印图读取权限为私有，则需要携带有效签名。
3. URL 必须以 `http://` 开始，不能省略 HTTP 头，也不能填 HTTPS 头，例如 `examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/shuiyin_2.png`，`https://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/shuiyin_2.png` 为非法的水印 URL。

示例代码

```php
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$blindWatermarkTemplate = new Qcloud\Cos\ImageParamTemplate\BlindWatermarkTemplate();//创建盲水印参数模版实例
$blindWatermarkTemplate->setPick();//设置为提取盲水印
$blindWatermarkTemplate->setType(3);//盲水印类型，有效值：1 半盲；2 全盲；3 文字
$blindWatermarkTemplate->setImage("imageUrl");//设置盲水印图片地址
$blindWatermarkTemplate->setText("Test");//设置盲水印文字
$blindWatermarkTemplate->setLevel(3);//只对全盲水印（type=2）有效。level 的取值范围为{1,2,3}，默认值为1，level 值越大则图片受影响程度越大、盲水印效果越好。
$picOperationsTemplate = new \Qcloud\Cos\ImageParamTemplate\PicOperationsTransformation();//创建图片持久化处理参数模版实例
$picOperationsTemplate->setIsPicInfo(1);//设置是否返回原图信息，0不返回原图信息，1返回原图信息，默认为0
$picOperationsTemplate->addRule($blindWatermarkTemplate, "resultobject");//设置图片持久化处理参数
$result = $cosClient->putObject(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
```

```
'Body' => fopen('path/to/localFile', 'rb'),
'PicOperations' => $picOperationsTemplate->queryString(),//生成图片持久化处理参数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
Body	File/String	上传的内容	是
PicOperations	Json/String	图片持久化处理信息	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Body] =>
[ETag] => "698d51a19d8a121ce581499d7b701668"
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[ContentLength] => 10000
[Key] => exampleobject
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
[Data] => Array
(
[OriginalInfo] => Array
(
[Key] => exampleobject
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
```

```
[ETag] => "7037fb6fb4cca43b958a28789605e73d98088720"
[ImageInfo] => Array
(
[Format] => JPEG
[Width] => 600
[Height] => 500
[Quality] => 90
[Ave] => 0x46442e
[Orientation] => 0
)
)
[ProcessResults] => Array
(
[Object] => Array
(
[0] => Array(
[Key] => resultobject
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/resultobject
[Format] => JPEG
[Width] => 300
[Height] => 200
[Size] => 30000
[Quality] => 90
[WatermarkStatus] => 90
[ETag] => "87c153bc2909aa0ba111ca126b675c510d36b817"
)
)
)
)
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Body	File/String	返回体	无
ETag	String	文件的 MD5 值	无
RequestId	String	请求 ID 标识	无
ContentLength	Int	响应体长度	无
Key	String	对象键	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	父节点
Location	String	请求资源地址	无
Data	Array	图片处理结果信息	无

版权所有：腾讯云计算（北京）有限责任公司

图片高级压缩

最近更新时间：2022-01-04 12:30:51

简介

本文档提供关于图片高级压缩的 API 概览以及 SDK 示例代码。

API	操作描述
图片高级压缩	图片高级压缩可以更加高效地将图片转换为 TPG 或 HEIF 等高压缩比格式，有效降低图片传输链路及加载耗时，降低带宽及流量成本

图片高级压缩

功能说明

通过数据万象 imageMogr2 接口提供图片高级压缩功能。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$imageMogrTemplate = new Qcloud\Cos\ImageParamTemplate\ImageMogrTemplate();//创建基础图片处理参数模版实例
$imageMogrTemplate->format('tpg');//将原图转换为 TPG 格式，
$imageMogrTemplate->format('heif');//将原图转换为 HEIF 格式
$result = $cosClient->getObject(array(
```

```
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'exampleobject',
'ImageHandleParam' => $imageMogrTemplate->queryString(),//生成基础图片处理参数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	此处的 Key 为对象键，对象键是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 `doc/pic.jpg`	是
ImageHandleParam	String	万象图片处理参数，例如：imageMogr2/format/tpg	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Body] =>
[ETag] => "698d51a19d8a121ce581499d7b701668"
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[ContentLength] => 100
[ContentType] => image/jpeg
[Key] => exampleobject
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/exampleobject
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Body	File/String	下载内容	无
ETag	String	文件的 MD5 值	无
ContentLength	Int	响应体长度	无
ContentType	String	内容类型，设置 Content-Type	无
RequestId	String	请求 ID 标识	无
Key	String	对象键	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无

版权所有：腾讯云计算（北京）有限责任公司

Guetzli 压缩

最近更新时间：2022-04-06 16:53:38

简介

本文档提供关于 Guetzli 压缩的相关的 API 概览以及 SDK 示例代码。

API	操作描述
开通 Guetzli 压缩	对 Bucket 开通 Guetzli 压缩功能
查询 Guetzli 状态	用于查询 Guetzli 压缩功能是否开启
关闭 Guetzli 压缩	用于关闭 Guetzli 压缩功能

开通 Guetzli 压缩

功能说明

对 Bucket 开通 Guetzli 压缩功能。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->PutBucketGuetzli(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
```

```
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.pic.ap-beijing.myqcloud.com/
)
)
```

返回结果说明

参数名称	类型	描述	父节点
RequestId	String	请求 ID 标识	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无

查询 Guetzli 状态

功能说明

用于查询 Guetzli 压缩功能是否开启。

示例代码

```php
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->GetBucketGuetzli(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.pic.ap-beijing.myqcloud.com/
[GuetzliStatus] => on
```

版权所有：腾讯云计算（北京）有限责任公司

```
)
)
```

返回结果说明

参数名称	类型	描述	父节点
RequestId	String	请求 ID 标识	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无
GuetzliStatus	String	Guetzli 状态，on：开启，off：关闭	无

关闭 Guetzli 压缩

用于关闭 Guetzli 压缩功能。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
$result = $cosClient->DeleteBucketGuetzli(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
```

```
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.pic.ap-beijing.myqcloud.com/
)
)
```

返回结果说明

参数名称	类型	描述	父节点
RequestId	String	请求 ID 标识	无
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Location	String	请求资源地址	无

版权所有：腾讯云计算（北京）有限责任公司

媒体处理
队列接口

最近更新时间：2023-04-12 14:30:48

简介

本文档提供关于媒体处理队列接口的 API 概览和 SDK 示例代码。

API	操作描述
搜索媒体处理队列	搜索媒体处理队列。
更新媒体处理队列	更新媒体处理队列。

搜索媒体处理队列

功能说明

搜索媒体处理队列。

方法原型

```
public Guzzle\Service\Resource\Model describeMediaQueues(array $args = array());
```

请求示例

```
<?php

require dirname(__FILE__, 2) . '/vendor/autoload.php';

$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
```

```
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
// https://cloud.tencent.com/document/product/436/54045 搜索媒体处理队列
$result = $cosClient->describeMediaQueues(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
// 'QueueIds' => '', // 可选 队列 ID，以“,”符号分割字符串
// 'Category' => 'Transcoding', // 可选 CateAll：所有类型；Transcoding：媒体处理队列；SpeedTranscoding：媒体处理倍速转码队列；默认为 Transcoding。
// 'State' => 'Paused', // 可选 1. Active 表示队列内的作业会被媒体转码服务调度转码执行 2. Paused 表示队列暂停，作业不再会被媒体转码调度转码执行，队列内的所有作业状态维持在暂停状态，已经处于转码中的任务将继续转码，不受影响
// 'PageNumber' => '1', // 可选 第几页
// 'PageSize' => '2', // 可选 每页个数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

请求参数描述如下：

参数名称（关键字）	描述	类型	是否必选
QueueIds	队列 ID，以“,”符号分割字符串。	String	否
State	• Active 表示队列内的作业会被媒体处理服务调度执行。 • Paused 表示队列暂停，作业不再会被媒体处理调度执行，队列内的所有作业状态维持在暂停状态，已经执行中的任务不受影响。	String	否
Category	• CateAll：所有类型。 • Transcoding：媒体处理队列。 • SpeedTranscoding：媒体处理倍速转码队列。 • 默认为 Transcoding。	String	否
PageNumber	第几页，默认值1。	String	否
PageSize	每页个数，默认值10。	String	否

版权所有：腾讯云计算（北京）有限责任公司

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[RequestId] => NjNkOGM2NmVfZTYxNmY5MSAASJPOJCIANSIDNAI=
[ContentType] => application/xml
[ContentLength] => 751
[TotalCount] => 1
[PageNumber] => 1
[PageSize] => 10
[QueueList] => Array
(
[0] => Array
(
[BucketId] => examplebucket-1250000000
[QueueId] => pcc3ae89sa9d807fs89dg789sdg
[Name] => queue-1
[State] => Active
[MaxSize] => 10000
[MaxConcurrent] => 10
[Category] => Transcoding
[UpdateTime] => 2022-07-13T10:41:34+0800
[CreateTime] => 2022-07-13T10:41:34+0800
[NotifyConfig] => Array
(
[Url] =>
[State] => Off
[Type] =>
[Event] =>
[ResultFormat] => XML
)

)

)

[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-guangzhou.myqcloud.com/queue
)
```

更新媒体处理队列

功能说明

更新媒体处理队列。

方法原型

```
public Guzzle\Service\Resource\Model updateMediaQueue(array $args = array());
```

请求示例

```
<?php

require dirname(__FILE__, 2) . '/vendor/autoload.php';

$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

try {
// https://cloud.tencent.com/document/product/436/54046 更新媒体处理队列
$result = $cosClient->updateMediaQueue(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'xxx', // queueId
'Name' => '', // 模板名称，长度限制100字符
'State' => 'Active', // 管道状态
'NotifyConfig' => array(
'State' => 'Off',
// 'Event' => '',
// 'ResultFormat' => '',
// 'Type' => '',
// 'Url' => '',
// 'MqMode' => '',
// 'MqRegion' => '',
// 'MqName' => '',
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
```

```
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选	默认值	限制
Name	Request	队列名称，长度不超过128	String	是	无	无
State	Request	• Active 表示队列内的作业会被媒体处理服务调度执行 • Paused 表示队列暂停，作业不再会被媒体处理调度执行，队列内的所有作业状态维持在暂停状态，已经执行中的任务不受影响	String	是	无	无
NotifyConfig	Request	回调配置	Container	是	无	无

Container 节点 NotifyConfig 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选	默认值	限制
State	Request.NotifyConfig	回调开关，Off/On	String	否	Off	On/Off
Event	Request.NotifyConfig	回调事件	String	当 State=On 时，必选	无	任务完成：TaskFinish；工作流完成：WorkflowFinish
ResultFormat	Request.NotifyConfig	回调格式	String	否	XML	JSON/XML
Type	Request.NotifyConfig	回调类型	String	当 State=On 时，必选	无	Url 或 TDMQ
Url	Request.NotifyConfig	回调地址	String	当 State=On，且 Type=Url 时，必选	无	不能为内网地址

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选	默认值	限制
MqMode	Request.NotifyConfig	TDMQ 使用模式	String	当 State=On, 且 Type=TDMQ 时, 必选	Queue	主题订阅：Topic 队列服务: Queue
MqRegion	Request.NotifyConfig	TDMQ 所属园区	String	当 State=On, 且 Type=TDMQ 时, 必选	无	目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）
MqName	Request.NotifyConfig	TDMQ 主题名称	String	当 State=On, 且 Type=TDMQ 时, 必选	无	无

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[Body] => GuzzleHttp\Psr7\Stream Object
(
[stream:GuzzleHttp\Psr7\Stream:private] => Resource id #88
[size:GuzzleHttp\Psr7\Stream:private] =>
[seekable:GuzzleHttp\Psr7\Stream:private] => 1
[readable:GuzzleHttp\Psr7\Stream:private] => 1
[writable:GuzzleHttp\Psr7\Stream:private] => 1
[uri:GuzzleHttp\Psr7\Stream:private] => php://temp
[customMetadata:GuzzleHttp\Psr7\Stream:private] => Array
(
)

)

[RequestId] => NjNkOGM2NmVfZTYxNmY5MSAASJPOJCIANSIDNAI=
[ContentType] => application/xml
[ContentLength] => 668
[Key] => fcd32dbdaa13b11esa9ds8g0d98gd0h85
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-guangzhou.myqcloud.com/queue/fcd32db
daa13b11esa9ds8g0d98gd0h85
[Response] => Array
```

```
(
[RequestId] => NjNkOGM2NmVfZTYxNmY5MSAASJPOJCIANSIDNAI=
[Queue] => Array
(
[QueueId] => fcd32dbdaa13b11esa9ds8g0d98gd0h85
[Name] => media-queue-1
[State] => Active
[NotifyConfig] => Array
(
[Url] =>
[Event] =>
[Type] =>
[State] => Off
[ResultFormat] => XML
[MqMode] =>
[MqName] =>
[MqRegion] =>
)

[MaxSize] => 10000
[MaxConcurrent] => 10
[CreateTime] => 2022-07-13T10:41:34+0800
[UpdateTime] => 2023-01-31T17:32:18+0800
[BucketId] => examplebucket-1250000000
[Category] => Transcoding
)

)

)
```

媒体 bucket 接口

最近更新时间：2022-05-23 14:50:09

简介

本文档提供关于查询媒体处理开通状态的 API 概览和 SDK 示例代码。

API	操作描述
查询媒体处理开通状态	查询媒体处理开通状态

查询媒体处理开通状态

功能说明

查询已经开通媒体处理功能的存储桶。

方法原型

```
public Guzzle\Service\Resource\Model describeMediaBuckets(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->describeMediaBuckets(array(
'Regions' => '', // 可选 地域信息，例如 ap-shanghai、ap-beijing，若查询多个地域以“,”分隔
```

```
字符串
'BucketNames' => '', // 可选 存储桶名称，以“,”分隔，支持多个存储桶，精确搜索
'BucketName' => '', // 可选 存储桶名称前缀，前缀搜索
'PageNumber' => 1, // 可选 第几页
'PageSize' => 20, // 可选 每页个数
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

名称	描述	类型	是否必选
regions	地域信息，例如 ap-shanghai、ap-beijing，若查询多个地域以“,”分隔字符串，支持中国大陆地域，详情请参见 地域与域名	string	否
bucketNames	存储桶名称，以“,”分隔，支持多个存储桶，精确搜索	string	否
bucketName	存储桶名称前缀，前缀搜索	string	否
pageNumber	第几页	string	否
pageSize	每页个数	string	否

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[RequestId] => NjI2MTE5OTlfNzgSxDxIHF0ZF8xZjk0MmQ=
[ContentType] => application/xml
[ContentLength] => 4744
[TotalCount] => 51
[PageNumber] => 1
[PageSize] => 5
[MediaBucketList] => Array
(
[0] => Array
(
[BucketId] => examplebucket-1250000000
[Name] => examplebucket-1250000000
```

```
[Region] => ap-guangzhou
[CreateTime] => 2022-04-07T22:06:57+0800
)
[1] => Array
(
[BucketId] => examplebucket-1250000000
[Name] => examplebucket-1250000000
[Region] => ap-guangzhou
[CreateTime] => 2022-04-07T22:06:57+0800
)
[2] => Array
(
[BucketId] => examplebucket-1250000000
[Name] => examplebucket-1250000000
[Region] => ap-guangzhou
[CreateTime] => 2022-04-07T22:06:57+0800
)
[3] => Array
(
[BucketId] => examplebucket-1250000000
[Name] => examplebucket-1250000000
[Region] => ap-guangzhou
[CreateTime] => 2022-04-07T22:06:57+0800
)
[4] => Array
(
[BucketId] => examplebucket-1250000000
[Name] => examplebucket-1250000000
[Region] => ap-guangzhou
[CreateTime] => 2022-04-07T22:06:57+0800
)
)
[Location] => ci.ap-guangzhou.myqcloud.com/mediabucket
)
```

版权所有：腾讯云计算（北京）有限责任公司

私有 M3U8 接口

最近更新时间：2022-05-23 14:50:09

简介

本文档提供关于私有 M3U8 接口的 API 概览和 SDK 示例代码。

API	操作描述
GetPrivateM3U8	用于获取私有 M3U8 ts 资源的下载授权。（此方式通过对象存储转发请求至数据万象）。

GetPrivateM3U8

功能说明

用于获取私有 M3U8 ts 资源的下载授权。（此方式通过对象存储转发请求至数据万象）。

方法原型

```
public Guzzle\Service\Resource\Model getPrivateM3U8(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->GetPrivateM3U8(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在
```

```
COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'xxx.m3u8', // 桶文件
'ci-process' => 'pm3u8', // 操作类型，固定使用 pm3u8
'expires' => '3600', // 私有 ts 资源 url 下载凭证的相对有效期，单位为秒，范围为[3600, 43200]
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

参数名称	描述	类型	是否必选
ci-process	操作类型，固定使用 pm3u8	String	是
expires	私有 ts 资源 url 下载凭证的相对有效期，单位为秒，范围为[3600, 43200]	String	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[Body] => GuzzleHttp\Psr7\Stream Object
(
[stream:GuzzleHttp\Psr7\Stream:private] => Resource id #89
[size:GuzzleHttp\Psr7\Stream:private] =>
[seekable:GuzzleHttp\Psr7\Stream:private] => 1
[readable:GuzzleHttp\Psr7\Stream:private] => 1
[writable:GuzzleHttp\Psr7\Stream:private] => 1
[uri:GuzzleHttp\Psr7\Stream:private] => php://temp
[customMetadata:GuzzleHttp\Psr7\Stream:private] => Array
(
)
)
[RequestId] => NjI2MTFjYSDHIUsdsdSJDOIDTRfMjVjYzcw
[ContentType] => application/x-mpegURL
[ContentLength] => 4310
[Key] => xxx.m3u8
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-beijing.myqcloud.com/xxx.m3u8
```

```
[Data] => m3u8文件内容
)
```

版权所有：腾讯云计算（北京）有限责任公司

文档处理
文档转码
同步请求接口

最近更新时间：2022-10-19 14:31:24

简介

本文档提供关于文档预览的相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
文档转码	文档转码同步请求	获取文档转码的请求 URL

文档转码同步请求

功能说明

文档转码同步请求，获取文档转码的请求 URL。

注意：

- 使用此接口前，请确保已打开官网控制台中数据处理下的文档处理开关，否则会报错 `doc bucket unbinded, bucket's host is unavailable` 。
- 如需使用文档转 HTML 功能，可直接将请求参数置于 URL 后方，SDK 仅负责签名计算操作。

示例代码

```
<?php
require dirname(__FILE__, 2) . '/vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
```

版权所有：腾讯云计算（北京）有限责任公司

```
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
// 1. 文档转码 https://cloud.tencent.com/document/product/460/47074
$bucket = 'examplebucket-1250000000';
$key = 'exampleobject';
$url = $cosClient->getObjectUrl($bucket, $key);
$params = array(
'ci-process' => 'doc-preview',
// 'srcType' => '',
'page' => 3,
'dstType' => 'png',
// 'password' => '',
// 'comment' => '',
// 'sheet' => '',
// 'excelPaperDirection' => '',
// 'excelRow' => '',
// 'excelCol' => '',
// 'excelPaperSize' => '',
// 'txtPagination' => '',
'ImageParams' => 'imageMogr2/thumbnail/!50p',
// 'quality' => '',
// 'scale' => '',
// 'imageDpi' => '',
);
$query = http_build_query($params);
echo $url . $query; // 生成的可访问链接
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

通用请求参数

名称	描述	类型	是否必选
Key	对象文件名，例如 folder/document.pdf	String	是
ci-process	数据万象处理能力，文档预览固定为 doc-preview	String	是

名称	描述	类型	是否必选
srcType	源数据的后缀类型，当前文档转换根据 COS 对象的后缀名来确定源数据类型。当 COS 对象没有后缀名时，可以设置该值	String	否
page	需转换的文档页码，默认从1开始计数；表格文件中 page 表示转换的第 X 个 sheet 的第 X 张图	Int	否
dstType	转换输出目标文件类型： png，转成 png 格式的图片文件；jpg，转成 jpg 格式的图片文件； pdf，转成 pdf 格式文件；无法选择页码，page 参数不生效；txt，转成 txt 格式文件。如果传入的格式未能识别，默认使用 jpg 格式	String	否
password	Office 文档的打开密码，如果需要转换有密码的文档，请设置该字段	String	否
comment	是否隐藏批注和应用修订，默认为0。0：隐藏批注，应用修订 1：显示批注和修订	Int	否

适用于表格文件（Excel）的参数

名称	描述	类型	是否必选
sheet	表格文件参数，转换第 X 个表，默认为1	Int	否
excelPaperDirection	表格文件转换纸张方向，0代表垂直方向，非0代表水平方向，默认为0	Int	否
excelRow	值为 `1` 表示将所有列放到1页进行排版，默认值为0	Int	否
excelCol	值为 `1` 表示将所有行放到1页进行排版，默认值为0	Int	否
excelPaperSize	设置纸张（画布）大小，对应信息为： `0 → A4` 、 `1 → A2` 、 `2 → A0` ，默认 A4 纸张 （需配合 `excelRow` 或 `excelCol` 一起使用）	Int	否

适用于转码成 txt 的参数

名称	描述	类型	是否必选
txtPagination	是否转换成长文本，设置为 `true` 时，可以将需要导出的页中的文字合并导出，分页范围可以通过 Ranges 控制。默认值为 `false` ，按页导出 txt。（ `ExportType="txt"` 时生效）	Bool	否

适用于转码成 png/jpg 图片的参数

版权所有：腾讯云计算（北京）有限责任公司

名称	描述	类型	是否必选
ImageParams	转换后的图片处理参数，支持 基础图片处理 所有处理参数，多个处理参数可通过 管道操作符 分隔，从而实现在一次访问中按顺序对图片进行不同处理	String	否
quality	生成预览图的图片质量，取值范围为 [1, 100]，默认值100。 例如取值为100，代表生成图片质量为100%	Int	否
scale	预览图片的缩放参数，取值范围为 [10, 200]， 默认值100。 例如取值为200，代表图片缩放比例为200% 即放大两倍	Int	否
imageDpi	按指定 dpi 渲染图片，该参数与 `scale` 共同作用，取值范围 `96-600` ，默认值为 `96` 。转码后的图片单边宽度需小于65500像素	Int	否

注意：

- 目前支持的输入文件类型包含如下格式：
  - 演示文件：pptx、ppt、pot、potx、pps、ppsx、dps、dpt、pptm、potm、ppsm。
  - 文字文件：doc、dot、wps、wpt、docx、dotx、docm、dotm。
  - 表格文件：xls、xlt、et、ett、xlsx、xltx、csv、xlsb、xlsm、xltm、ets。
    表格文件，一张表可能分割为多页转换，生成多张图片。
  - 其他格式文件：pdf、lrc、c、cpp、h、asm、s、java、asp、bat、bas、prg、cmd、rtf、txt、log、xml、htm、html。
- 输入文件大小限制在200MB之内。
- 输入文件页数限制在5000页之内。

返回结果示例

同步请求URL

```
https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/example.ppt?sign=q-sign-algorithmxxxxxxxxxxxxx&ci-process=doc-preview&page=1&dstType=png
```

异步处理任务接口

最近更新时间：2022-11-07 14:24:33

简介

本文档提供关于异步处理任务接口的相关的 API 概览以及 SDK 示例代码。

注意：
需要 COS PHP SDK v2.4.0 及以上版本。旧版本可能存在 bug，使用时建议升级到 最新版本。

API	操作名	操作描述
CreateDocProcessJobs	提交文档预览任务	用于提交一个文档预览任务
DescribeDocProcessJob	查询文档预览任务	查询指定的文档预览任务
DescribeDocProcessJobs	拉取文档预览任务	拉取符合条件的文档预览任务

提交文档预览任务

功能说明

CreateDocProcessJobs 接口用于提交一个文档预览任务。

注意：

- 使用此接口前，请确保已打开官网控制台中数据处理下的文档处理开关，否则会报错 `doc bucket unbinded, bucket's host is unavailable` 。
- 如需使用文档转 HTML 功能，可直接将请求参数置于 URL 后方，SDK 仅负责签名计算操作。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
```

```php
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->createDocProcessJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'DocProcess', //任务的 Tag：DocProcess 固定值
'QueueId' => 'pd8e422a2ea134165a92f2012ea43****', //任务所在的队列 ID
'Input' => array(
'Object' => 'Append功能测试.pdf' //待操作的文件对象
),
'Operation' => array(
'DocProcess' => array(
'SrcType' => 'pdf', //源数据的后缀类型
'TgtType' => 'png', //转换输出目标文件类型
'SheetId' => 0, //表格文件参数，转换第 x 个表，默认为1
'StartPage' => 1, //从第 x 页开始转换，默认为1
'EndPage' => 3, //转换至第 x 页，默认为-1，即转换全部页
'ImageParams' => '', //转换后的图片处理参数
'DocPassword' => '', //Office 文档的打开密码
'Comments' => 0, //是否隐藏批注和应用修订，默认为 0
'PaperDirection' => 0, //表格文件转换纸张方向，默认为0
'Quality' => 100, //生成预览图的图片质量，取值范围 [1-100]，默认值100
'Zoom' => 100, //预览图片的缩放参数，取值范围[10-200]， 默认值100
),
'Output' => array(
'Region' => $region, //存储桶的地域
'Bucket' => 'examplebucket-1250000000', //存储结果的存储桶
'Object' => 'pic-${Page}.jpg', //输出文件路径
),
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Operation	Array	操作规则	是
Tag	String	创建任务的 Tag，目前仅支持：DocProcess	是
Input	Array	待操作的文件对象	是
QueueId	String	任务所在的队列 ID	是

Input 中的具体数据描述如下：

参数名称	类型	描述	是否必填
Object	String	文件在 COS 上的文件路径，Bucket 由 Host 指定	是

Operation 中的具体数据描述如下：

参数名称	类型	描述	是否必填
DocProcess	Array	当 Tag 为 DocProcess 时有效，指定该任务的参数	是
Output	Array	结果输出地址	是

DocProcess 中的具体数据描述如下：

参数名称	类型	描述	是否必填
srcType	String	源数据的后缀类型，当前文档转换根据 cos 对象的后缀名来确定源数据类型，当 cos 对象没有后缀名时，可以设置该值	否
tgtType	String	转换输出目标文件类型： jpg，转成 jpg 格式的图片文件；如果传入的格式未能识别，默认使用 jpg 格式 png，转成 png 格式的图片文件 pdf，转成 pdf 格式文件（暂不支持指定页数）	否
sheetId	Integer	表格文件参数，转换第 X 个表，默认为1；设置 SheetId 为0，即转换文档中全部表	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述	是否必填
startPage	Integer	从第 X 页开始转换；在表格文件中，一张表可能分割为多页转换，生成多张图片。StartPage 表示从指定 SheetId 的第 X 页开始转换。默认为1	否
endPage	Integer	转换至第 X 页；在表格文件中，一张表可能分割为多页转换，生成多张图片。EndPage 表示转换至指定 SheetId 的第 X 页。默认为-1，即转换全部页	否
imageParams	String	转换后的图片处理参数，支持 基础图片处理 所有处理参数，多个处理参数可通过 管道操作符 分隔，从而实现在一次访问中按顺序对图片进行不同处理	否
docPassword	String	Office 文档的打开密码，如果需要转换有密码的文档，请设置该字段	否
comments	Integer	是否隐藏批注和应用修订，默认为 0；0：隐藏批注，应用修订；1：显示批注和修订	否
paperDirection	Integer	表格文件转换纸张方向，0代表垂直方向，非0代表水平方向，默认为0	否
quality	Integer	生成预览图的图片质量，取值范围 [1-100]，默认值100。 例：值为100，代表生成图片质量为100%	否
zoom	Integer	预览图片的缩放参数，取值范围[10-200]， 默认值100。 例：值为200，代表图片缩放比例为200% 即放大两倍	否

Output 中的具体数据描述如下：

参数名称	类型	描述	是否必填
Region	String	存储桶的地域	是
Bucket	String	存储结果的存储桶	是
Object	String	输出文件路径	是

返回参数说明

参数名称	类型	描述
Code	String	错误码，只有 State 为 Failed 时有意义
Message	String	错误描述，只有 State 为 Failed 时有意义

版权所有：腾讯云计算（北京）有限责任公司

参数名称	类型	描述
JobId	String	新创建任务的 ID
Tag	String	新创建任务的 Tag：DocProcess
State	String	任务的状态，为 Submitted、Running、Success、Failed、Pause、Cancel 其中一个
CreationTime	String	任务的创建时间
QueueId	String	任务所属的队列 ID
Input	Array	该任务的输入文件路径
Operation	Array	该任务的规则

查询文档预览任务

功能说明

DescribeDocProcessJob 用于查询指定的文档预览任务。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->describeDocProcessJob(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'examplejobid', // JobId
```

```
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述
Bucket	COS 存储桶名称	String
Key	创建任务的 ID	String

返回参数说明

参数名称	类型	描述
jobsDetail	记录任务的详细信息	Array

拉取文档预览任务

功能说明

DescribeDocProcessJobs 用于拉取符合条件的文档预览任务。

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
```

```
'secretKey' => $secretKey)));
try {
$result = $cosClient->getDescribeDocProcessJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'QueueId' => 'pd8e422a2ea134165a92f2012ea43****', //拉取该队列 ID 下的任务
'Tag' => 'DocProcess', //任务的 Tag：DocProcess 固定值
// 'NextToken' => '143486', //请求的上下文，用于翻页。上次返回的值
// 'OrderByTime' => 'Desc', //Desc 或者 Asc。默认为 Desc
// 'Size' => 2, //拉取的最大任务数。默认为10。最大为100
// 'States' => 'All', //拉取该状态的任务，以,分割，支持多状态：All、Submitted、Running、Success、Failed、Pause、Cancel。默认为 All。
// 'StartCreationTime' => '2021-10-10T16:20:07+0800', //拉取创建时间大于该时间的任务
// 'EndCreationTime' => '2021-10-10T16:20:07+0800', //拉取创建时间小于该时间的任务
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	COS 存储桶名称	是
QueueId	String	拉取该队列 ID 下的任务	是
Tag	String	任务的 Tag：DocProcess	是
OrderByTime	String	Desc 或者 Asc，默认为 Desc	否
NextToken	String	请求的上下文，用于翻页，设置为上次返回的值	否
Size	Integer	拉取的最大任务数，默认为10，最大为100	否
States	String	拉取该状态的任务，以 `,` 分割，支持多状态：All、Submitted、Running、Success、Failed、Pause、Cancel，默认为 All	否
StartCreationTime	String	拉取创建时间大于该时间的任务，格式为：`%Y-%m-%dT%H:%m:%S%z`	否
EndCreationTime	String	拉取创建时间小于该时间的任务，格式为：`%Y-%m-%dT%H:%m:%S%z`	否

版权所有：腾讯云计算（北京）有限责任公司

返回参数说明

参数名称	类型	描述
JobDetail	Array	记录任务列表
nextToken	String	翻页的上下文 Token

异步处理队列接口

最近更新时间：2022-11-07 14:24:34

简介

本文档提供关于异步处理队列接口相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DescribeDocProcessQueues	查询文档转码队列	用于查询文档转码队列
UpdateDocProcessQueue	更新文档转码队列	用于更新文档转码队列

查询文档转码队列

功能说明

DescribeDocProcessQueues 接口用于查询文档转码队列。（可通过该接口获取桶对应的队列 ID）

示例代码

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->describeDocProcessQueues(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
));
// 请求成功
```

```
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述
Bucket	COS 存储桶名称	String

返回参数说明

Response 中的具体数据描述如下：

参数名称	类型	描述
RequestId	String	请求的唯一 ID
TotalCount	Integer	队列总数
PageNumber	Integer	当前页数，同请求中的 pageNumber
PageSize	Integer	每页个数，同请求中的 pageSize
QueueList	Array	队列数组

QueueList 中的具体数据描述如下：

参数名称	类型	描述
QueueId	String	队列 ID
Name	String	队列名字
State	String	当前状态，Active 或者 Paused
NotifyConfig	Array	回调配置
MaxSize	Integer	队列最大长度
MaxConcurrent	Integer	当前队列最大并行执行的任务数
UpdateTime	String	更新时间
CreateTime	String	创建时间

版权所有：腾讯云计算（北京）有限责任公司

NotifyConfig 中的具体数据描述如下：

参数名称	类型	描述
Url	String	回调地址
State	String	开关状态，On 或者 Off
Type	String	回调类型，Url
Event	String	触发回调的事件

更新文档转码队列

功能说明

UpdateDocProcessQueue 接口用于更新文档转码队列。

示例代码

```
<?php
require dirname(__FILE__, 2) . '/vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
// 更新文档转码队列 https://cloud.tencent.com/document/product/460/46947
$result = $cosClient->updateDocProcessQueue(array(
'Bucket' => 'examplebucket-125000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => '', // QueueID
'Name' => '',
'QueueID' => '',
'State' => '',
'NotifyConfig' => array(
```

```
'Url' => '',
'Type' => '',
'Event' => '',
'State' => '',
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选	限制
Name	Request	队列名称	String	是	长度限制100字符
QueueID	Request	队列ID	String	是	无
State	Request	队列状态	String	是	1. Active 表示队列内的作业会被文档预览服务调度转码执行 2. Paused 表示队列暂停，作业不再会被文档预览服务调度执行，队列内的所有作业状态维持在已提交状态，已经处于执行中的任务将继续执行，不受影响
NotifyConfig	Request	通知渠道	Container	是	第三方回调 Url

Container 类型 NotifyConfig 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选	限制
Url	Request.NotifyConfig	回调配置	String	否	长度限制100字符
Type	Request.NotifyConfig	回调类型，普通回调：Url	String	否	长度限制100字符

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选	限制
Event	Request.NotifyConfig	回调事件，文档预览任务完成	String	否	长度限制100字符
State	Request.NotifyConfig	回调开关，Off，On	String	否	长度限制100字符

返回参数说明

Response 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型
RequestId	Response	请求的唯一 ID	String
Queue	Response	队列信息，详情请参见 DescribeDocProcessQueues 中的 Response.QueueList	Container

版权所有：腾讯云计算（北京）有限责任公司

文件处理
多文件打包压缩

最近更新时间：2023-03-28 14:38:57

简介

本文档提供关于多文件打包压缩接口的 API 概览和 SDK 示例代码。

API	操作描述
提交多文件打包压缩任务	多文件打包压缩功能可以将您的多个文件，打包为 zip 等压缩包格式，以提交任务的方式进行多文件打包压缩，异步返回打包后的文件，该接口属于 POST 请求。
查询多文件打包压缩结果	本接口用于主动查询指定的多文件打包压缩任务结果。

提交多文件打包压缩任务

功能说明

多文件打包压缩功能可以将您的多个文件，打包为 zip 等压缩包格式，以提交任务的方式进行多文件打包压缩，异步返回打包后的文件，该接口属于 POST 请求。

方法原型

```
public Guzzle\Service\Resource\Model createFileCompressJobs(array $args = array
());
```

请求示例

```
<?php

require dirname(__FILE__, 2) . '/vendor/autoload.php';

$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
```

```
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，h
ttps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
// https://intl.cloud.tencent.com/document/product/436/53995 提交多文件打包压缩任务-
异步
$result = $cosClient->createFileCompressJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以
在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'FileCompress',
// 'QueueId' => 'pcc3ae89sa9d807fs89dg789sdg',
'Operation' => array(
'UserData' => 'xxx',
'FileCompressConfig' => array(
'Flatten' => '0',
'Format' => 'zip',
// 'UrlList' => 'test/index.csv',
// 'Prefix' => 'test/',
'Keys' => array(
'object1', // 待压缩桶文件
'object2', // 待压缩桶文件
'object3', // 待压缩桶文件
),
),
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以
在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'output/test.zip',
),
),
// 'CallBackFormat' => '',
// 'CallBackType' => '',
// 'CallBack' => '',
// 'CallBackMqConfig' => array(
// 'MqRegion' => '',
// 'MqMode' => '',
// 'MqName' => '',
// ),
));
```

```
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	表示任务的类型，多文件打包压缩默认为：FileCompress。	String	是
Operation	Request	包含文件打包压缩的处理规则。	Container	是
QueueId	Request	任务所在的队列 ID。	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式。	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型。	String	否
CallBack	Request	任务回调的地址，优先级高于队列的回调地址。	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情请参见 CallBackMqConfig	Container	否

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
FileCompressConfig	Request.Operation	指定文件打包压缩的处理规则。	Container	是
UserData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否
Output	Request.Operation	指定打包压缩后的文件保存的地址信息。	Container	是

Container 类型 FileCompressConfig 的具体数据描述如下：

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型
Flatten	Request.Operation.FileCompressConfig	文件打包时，是否需要去除源文件已有的目录结构，有效值： • 0：不需要去除目录结构，打包后压缩包中的文件会保留原有的目录结构； • 1：需要，打包后压缩包内的文件会去除原有的目录结构，所有文件都在同一层级。例如：源文件 URL 为 `https://domain/source/test.mp4`，则源文件路径为 source/test.mp4，如果为1，则 ZIP 包中该文件路径为 test.mp4；如果为0， ZIP 包中该文件路径为 source/test.mp4。	String
Format	Request.Operation.FileCompressConfig	打包压缩的类型，有效值：zip、tar、tar.gz。	String
UrlList	Request.Operation.FileCompressConfig	• 支持将需要打包的文件整理成索引文件，后台将根据索引文件内提供的文件url，打包为一个压缩包文件。索引文件需要保存在当前存储桶中，本字段需要提供索引文件的对象地址，例如：/test/index.csv。 • 索引文件格式：仅支持 CVS文件，一行一条URL（仅支持本存储桶文件），如有多列字段，默认取第一列作为URL。最多不超过10000个文件, 总大小不超过50G, 否则会导致任务失败。	String
Prefix	Request.Operation.FileCompressConfig	支持对存储桶中的某个前缀进行打包，如果需要对某个目录进行打包，需要加/，例如test目录打包，则值为：test/。最多不超过10000个文件，总大小不超过50G，否则会导致任务失败。	String
Key	Request.Operation.FileCompressConfig	支持对存储桶中的多个文件进行打包，个数不能超过 1000, 总大小不超过50G，否则会导致任务失败。	String 数组

注意：

UrlList、Prefix、Key 三者仅能选择一个，不能都为空，也不会同时生效。如果填了多个，会按优先级 UrlList > Prefix > Key 取最高优先级执行。

Container 类型 Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域。	String	是
Bucket	Request.Operation.Output	保存压缩后文件的存储桶。	String	是
Object	Request.Operation.Output	压缩后文件的文件名	String	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[RequestId] => NjNkOGM2NmVfZTYxNmY5MSAASJPOJCIANSIDNAI=
[ContentType] => application/xml
[ContentLength] => 827
[JobsDetail] => Array
(
[Progress] => 0
[Code] => Success
[Message] =>
[JobId] => fcd32dbdaa13b11esa9ds8g0d98gd0h85
[Tag] => FileCompress
[State] => Submitted
[CreationTime] => 2023-01-31T16:17:21+0800
[StartTime] => -
[EndTime] => -
[QueueId] => pcc3ae89sa9d807fs89dg789sdg
[Operation] => Array
(
[JobLevel] => 0
[UserData] => xxx
[Output] => Array
(
[Region] => ap-guangzhou
[Bucket] => examplebucket-1250000000
[Object] => tmp/test.zip
)

[FileCompressConfig] => Array
(
```

```
[Flatten] => 0
[Format] => zip
[UrlList] =>
[Prefix] =>
[Key] => Array
(
[0] => test1.png
[1] => test2.jpeg
)

)

)

)

[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-guangzhou.myqcloud.com/file_jobs
)
```

查询多文件打包压缩结果

功能说明

本接口用于主动查询指定的多文件打包压缩任务结果。

方法原型

```
public Guzzle\Service\Resource\Model getFileCompressResult(array $args = array
());
```

请求示例

```
<?php

require dirname(__FILE__, 2) . '/vendor/autoload.php';

$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，h
ttps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
```

```
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
// https://intl.cloud.tencent.com/document/product/436/53996 查询多文件打包压缩结果
$result = $cosClient->getFileCompressResult(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => '', // jobId
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	jobId	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[RequestId] => NjNkOGM2NmVfZTYxNmY5MSAASJPOJCIANSIDNAI=
[ContentType] => application/xml
[ContentLength] => 1093
[JobsDetail] => Array
(
[Progress] => 100
[Code] => Success
[Message] => success
[JobId] => fcd32dbdaa13b11esa9ds8g0d98gd0h85
[Tag] => FileCompress
[State] => Success
[CreationTime] => 2023-01-31T16:17:21+0800
```

版权所有：腾讯云计算（北京）有限责任公司

```
[StartTime] => 2023-01-31T16:17:21+0800
[EndTime] => 2023-01-31T16:17:22+0800
[QueueId] => pcc3ae89sa9d807fs89dg789sdg
[Operation] => Array
(
[JobLevel] => 0
[UserData] => xxx
[Output] => Array
(
[Region] => ap-guangzhou
[Bucket] => examplebucket-1250000000
[Object] => tmp/test.zip
)

[FileCompressConfig] => Array
(
[Flatten] => 0
[Format] => zip
[UrlList] =>
[Prefix] =>
[Key] => Array
(
[0] => test1.png
[1] => test2.jpeg
)

)

[FileCompressResult] => Array
(
[CompressFileCount] => 2
[Region] => ap-guangzhou
[Bucket] => examplebucket-1250000000
[Object] => tmp/test.zip
)

)

)

[Key] => fcd32dbdaa13b11esa9ds8g0d98gd0h85
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-guangzhou.myqcloud.com/file_jobs/fcd
32dbdaa13b11esa9ds8g0d98gd0h85
)
```

文件哈希值计算

最近更新时间：2023-03-28 14:38:57

简介

本文档提供关于文件哈希值计算接口的 API 概览和 SDK 示例代码。

API	操作描述
哈希值计算同步请求	以同步请求的方式进行文件哈希值计算，实时返回计算得到的哈希值，该接口属于 GET 请求。
提交哈希值计算任务	以提交任务的方式进行文件哈希值计算，异步返回计算得到的哈希值，该接口属于 POST 请求。
查询哈希值计算结果	本接口用于主动查询指定的文件哈希值计算任务结果。

哈希值计算同步请求

功能说明

以同步请求的方式进行文件哈希值计算，实时返回计算得到的哈希值，该接口属于 GET 请求。

方法原型

```
public Guzzle\Service\Resource\Model FileJobs4Hash(array $args = array());
```

请求示例

```
<?php

require dirname(__FILE__, 2) . '/vendor/autoload.php';

$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
```

```
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
// https://intl.cloud.tencent.com/document/product/436/53990 哈希值计算同步请求
$result = $cosClient->FileJobs4Hash(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => 'for-test.mp4', // 桶文件
'Type' => 'md5', // 支持的哈希算法类型，有效值：md5、sha1、sha256
// 'AddToHeader' => 'true', // 是否将计算得到的哈希值，自动添加至文件的自定义 header，格式为：x-cos-meta-md5/sha1/sha256; 有效值： true、false，不填则默认为 false。
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
/**
* 可能出现的 Exception
* 1. Error Message: file processing is not active yet, please apply for file processing service first
* 解决方式：开通文件处理服务
*/
}
```

参数说明

Request 中的具体数据描述如下：

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	桶文件	是
Type	String	支持的哈希算法类型，有效值：md5、sha1、sha256	是
AddToHeader	String	是否将计算得到的哈希值，自动添加至文件的自定义header，格式为：x-cos-meta-md5/sha1/sha256; 有效值： true、false，不填则默认为 false。	否

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[RequestId] => NjNkOGM2NmVfZTYxNmY5MSAASJPOJCIANSIDNAI
[ContentType] => application/xml
[ContentLength] => 421
[FileHashCodeResult] => Array
(
[Etag] => "1f3ede740849eaa2e79e9b269ea3db2a"
[LastModified] => 2023-01-29T11:31:02+0000
[MD5] => 1f3ede740849eaa2e79e9b269ea3db2a
[FileSize] => 42120
)

[Input] => Array
(
[Region] => ap-guangzhou
[Bucket] => examplebucket-1250000000
[Object] => sound01.mp3
)

[Key] => sound01.mp3
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sound01.mp3
)
```

提交哈希值计算任务

功能说明

以提交任务的方式进行文件哈希值计算，异步返回计算得到的哈希值，该接口属于 POST 请求。

方法原型

```
public Guzzle\Service\Resource\Model createFileHashCodeJobs(array $args = array
());
```

请求示例

```
<?php

require dirname(__FILE__, 2) . '/vendor/autoload.php';

$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
```

```
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，h
ttps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
// https://intl.cloud.tencent.com/document/product/436/53991 提交哈希值计算任务-异步
$result = $cosClient->createFileHashCodeJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以
在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'FileHashCode',
// 'QueueId' => 'pcc3ae89sa9d807fs89dg789sdg',
'Input' => array(
'Object' => 'test.mp4',
),
'Operation' => array(
'UserData' => 'xxx',
'FileHashCodeConfig' => array(
'Type' => 'MD5',
'AddToHeader' => 'true',
),
),
// 'CallBackFormat' => '',
// 'CallBackType' => '',
// 'CallBack' => '',
// 'CallBackMqConfig' => array(
// 'MqRegion' => '',
// 'MqMode' => '',
// 'MqName' => '',
// ),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	表示任务的类型，哈希值计算默认为：FileHashCode。	String	是
Input	Request	包含待操作的文件信息。	Container	是
Operation	Request	包含哈希值计算的处理规则。	Container	是
QueueId	Request	任务所在的队列 ID。	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式。	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型。	String	否
CallBack	Request	任务回调的地址，优先级高于队列的回调地址。	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	文件名，取值为文件在当前存储桶中的完整名称。	String	是

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
FileHashCodeConfig	Request.Operation	指定哈希值计算的处理规则。	Container	是
UserData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否

Container 类型 FileHashCodeConfig 的具体数据描述如下：

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Type	Request.Operation.FileHashCodeConfig	哈希值的算法类型，有效值：MD5、SHA1、SHA256。	String	是
AddToHeader	Request.Operation.FileHashCodeConfig	是否将计算得到的哈希值添加至文件自定义header，有效值：true、false，默认值为 false。自定义header根据 `Type` 的值变化，例如 `Type` 值为MD5时，自定义header为 x-cos-meta-md5。	String	否

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[RequestId] => NjNkOGM2NmVfZTYxNmY5MSAASJPOJCIANSIDNAI=
[ContentType] => application/xml
[ContentLength] => 738
[JobsDetail] => Array
(
[Progress] => 0
[Code] => Success
[Message] =>
[JobId] => fcd32dbdaa13b11esa9ds8g0d98gd0h85
[Tag] => FileHashCode
[State] => Submitted
[CreationTime] => 2023-01-31T15:49:33+0800
[StartTime] => -
[EndTime] => -
[QueueId] => pcc3ae89sa9d807fs89dg789sdg
[Input] => Array
(
[BucketId] => examplebucket-1250000000
[Region] => ap-guangzhou
[Object] => sound01.mp3
)

[Operation] => Array
(
```

```
[JobLevel] => 0
[UserData] => xxx
[FileHashCodeConfig] => Array
(
[Type] => MD5
[AddToHeader] => true
)

)

)

[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-guangzhou.myqcloud.com/file_jobs
)
```

查询哈希值计算结果

功能说明

本接口用于主动查询指定的文件哈希值计算任务结果。

方法原型

```
public Guzzle\Service\Resource\Model getFileHashCodeResult(array $args = array
());
```

请求示例

```
<?php

require dirname(__FILE__, 2) . '/vendor/autoload.php';

$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，h
ttps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
```

```
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
// https://intl.cloud.tencent.com/document/product/436/53992 查询哈希值计算结果
$result = $cosClient->getFileHashCodeResult(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => '', // jobId
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	jobId	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[RequestId] => NjNkOGM2NmVfZTYxNmY5MSAASJPOJCIANSIDNAI=
[ContentType] => application/xml
[ContentLength] => 1048
[JobsDetail] => Array
(
[Progress] => 100
[Code] => Success
[Message] => success
[JobId] => fcd32dbdaa13b11esa9ds8g0d98gd0h85
[Tag] => FileHashCode
[State] => Success
[CreationTime] => 2023-01-31T15:49:33+0800
[StartTime] => 2023-01-31T15:49:33+0800
[EndTime] => 2023-01-31T15:49:34+0800
[QueueId] => pcc3ae89sa9d807fs89dg789sdg
[Input] => Array
```

版权所有：腾讯云计算（北京）有限责任公司

```
(
[BucketId] => examplebucket-1250000000
[Region] => ap-guangzhou
[Object] => sound01.mp3
)

[Operation] => Array
(
[JobLevel] => 0
[UserData] => xxx
[FileHashCodeConfig] => Array
(
[Type] => MD5
[AddToHeader] => true
)

[FileHashCodeResult] => Array
(
[MD5] => 1f3ede740849eaa2e79e9b269ea3db2a
[LastModified] => 2023-01-29T11:31:02+0000
[Etag] => "1f3ede740849eaa2e79e9b269ea3db2a"
[FileSize] => 42120
)

)

)

[Key] => fcd32dbdaa13b11esa9ds8g0d98gd0h85
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-guangzhou.myqcloud.com/file_jobs/fcd
32dbdaa13b11esa9ds8g0d98gd0h85
)
```

版权所有：腾讯云计算（北京）有限责任公司

文件解压

最近更新时间：2023-03-28 14:38:56

简介

本文档提供关于文件解压接口的 API 概览和 SDK 示例代码。

API	操作描述
提交文件解压任务	以提交任务的方式进行文件解压缩，异步返回压缩包内被解压出来的文件，该接口属于 POST 请求。
查询文件解压结果	本接口用于主动查询指定的文件解压任务结果。

提交文件解压任务

功能说明

以提交任务的方式进行文件解压缩，异步返回压缩包内被解压出来的文件，该接口属于 POST 请求。

方法原型

```
public Guzzle\Service\Resource\Model createFileUncompressJobs(array $args = array
());
```

请求示例

```
<?php

require dirname(__FILE__, 2) . '/vendor/autoload.php';

$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
```

```
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
// https://intl.cloud.tencent.com/document/product/436/53993 提交文件解压任务-异步
$result = $cosClient->createFileUncompressJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'FileUncompress',
// 'QueueId' => 'pcc3ae89sa9d807fs89dg789sdg',
'Input' => array(
'Object' => 'test.zip',
),
'Operation' => array(
'UserData' => 'xxx',
'FileUncompressConfig' => array(
'Prefix' => 'prefix',
'PrefixReplaced' => '1',
),
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
),
),
// 'CallBackFormat' => '',
// 'CallBackType' => '',
// 'CallBack' => '',
// 'CallBackMqConfig' => array(
// 'MqRegion' => '',
// 'MqMode' => '',
// 'MqName' => '',
// ),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	表示任务的类型，文件解压默认为：FileUncompress。	String	是
Input	Request	包含待操作的文件信息。	Container	是
Operation	Request	包含文件解压的处理规则。	Container	是
QueueId	Request	任务所在的队列 ID。	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式。	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型。	String	否
CallBack	Request	任务回调的地址，优先级高于队列的回调地址。	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	文件名，取值为文件在当前存储桶中的完整名称。	String	是

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
FileUncompressConfig	Request.Operation	指定文件解压的处理规则。	Container	是
UserData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否
Output	Request.Operation	指定解压后的文件保存的存储桶信息。	Container	是

Container 类型 FileUncompressConfig 的具体数据描述如下：

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Prefix	Request.Operation.FileUncompressConfig	指定解压后输出文件的前缀，不填则默认保存在存储桶根路径。	String	否
PrefixReplaced	Request.Operation.FileUncompressConfig	指定解压后的文件路径是否需要替换前缀，有效值： • 0：不添加额外的前缀，解压缩将保存在Prefix指定的路径下（不会保留压缩包的名称，仅将压缩包内的文件保存至指定的路径）。 • 1：以压缩包本身的名称作为前缀，解压缩将保存在Prefix指定的路径下。 • 2：以压缩包完整路径作为前缀，此时如果不指定Prefix，就是解压到压缩包所在的当前路径（包含压缩包本身名称）。 • 默认值为0。	String	否

说明：
假设压缩包名为 test.zip，压缩包内的文件为“图片.jpg”，存储在了存储桶 A 的 123 目录下，则压缩包的完整路径为：123/test.zip。
对该压缩包进行解压缩，指定解压缩到存储桶A，并指定 `Prefix` 为 456，则不同的 `PrefixReplaced` 值对应解压后的文件保存情况如下：

- 0：“图片.jpg”会保存在 456 目录下，“图片.jpg”的完整路径为 456/图片.jpg；
- 1：“图片.jpg”会以 test 为前缀保存在 456 目录下，“图片.jpg”的完整路径为 456/test/图片.jpg；
- 2：“图片.jpg”会以 123/test 为前缀保存在 456 目录下，“图片.jpg”的完整路径为 456/123/test/图片.jpg。

Container 类型 Output 的具体数据描述如下：

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域。	String	是
Bucket	Request.Operation.Output	保存解压后文件的存储桶。	String	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[RequestId] => NjNkOGM2NmVfZTYxNmY5MSAASJPOJCIANSIDNAI=
[ContentType] => application/xml
[ContentLength] => 872
[JobsDetail] => Array
(
[Progress] => 0
[Code] => Success
[Message] =>
[JobId] => fcd32dbdaa13b11esa9ds8g0d98gd0h85
[Tag] => FileUncompress
[State] => Submitted
[CreationTime] => 2023-01-31T16:05:31+0800
[StartTime] => -
[EndTime] => -
[QueueId] => pcc3ae89sa9d807fs89dg789sdg
[Input] => Array
(
[BucketId] => examplebucket-1250000000
[Region] => ap-guangzhou
[Object] => test.zip
)

[Operation] => Array
(
[JobLevel] => 0
[UserData] => xxx
[Output] => Array
(
[Region] => ap-guangzhou
[Bucket] => examplebucket-1250000000
)

[FileUncompressConfig] => Array
(
[Prefix] => prefix
[PrefixReplaced] => 1
```

```
)

)

)

[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-guangzhou.myqcloud.com/file_jobs
)
```

查询文件解压结果

功能说明

本接口用于主动查询指定的文件解压任务结果。

方法原型

```
public Guzzle\Service\Resource\Model getFileUncompressResult(array $args = array
());
```

请求示例

```
<?php

require dirname(__FILE__, 2) . '/vendor/autoload.php';

$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，h
ttps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
// https://intl.cloud.tencent.com/document/product/436/53994 查询文件解压结果
$result = $cosClient->getFileUncompressResult(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以
```

```
在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Key' => '', // jobId
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	jobId	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[RequestId] => NjNkOGM2NmVfZTYxNmY5MSAASJPOJCIANSIDNAI=
[ContentType] => application/xml
[ContentLength] => 1092
[JobsDetail] => Array
(
[Progress] => 100
[Code] => Success
[Message] => success
[JobId] => fcd32dbdaa13b11esa9ds8g0d98gd0h85
[Tag] => FileUncompress
[State] => Success
[CreationTime] => 2023-01-31T16:05:31+0800
[StartTime] => 2023-01-31T16:05:31+0800
[EndTime] => 2023-01-31T16:05:31+0800
[QueueId] => pcc3ae89sa9d807fs89dg789sdg
[Input] => Array
(
[BucketId] => examplebucket-1250000000
[Region] => ap-guangzhou
[Object] => test.zip
)
```

版权所有：腾讯云计算（北京）有限责任公司

```
[Operation] => Array
(
[JobLevel] => 0
[UserData] => xxx
[Output] => Array
(
[Region] => ap-guangzhou
[Bucket] => examplebucket-1250000000
)

[FileUncompressConfig] => Array
(
[Prefix] => prefix
[PrefixReplaced] => 1
)

[FileUncompressResult] => Array
(
[Region] => ap-guangzhou
[Bucket] => examplebucket-1250000000
[FileCount] => 6
)

)

)

[Key] => fcd32dbdaa13b11esa9ds8g0d98gd0h85
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-guangzhou.myqcloud.com/file_jobs/fcd
32dbdaa13b11esa9ds8g0d98gd0h85
)
```

任务接口
动图任务接口

最近更新时间：2023-02-07 15:10:16

简介

本文档提供关于提交动图任务的 API 概览和 SDK 示例代码。

API	操作描述
提交动图任务	提交一个动图任务

提交动图任务

功能说明

用于提交一个动图任务。

方法原型

```
public Guzzle\Service\Resource\Model createMediaAnimationJobs(array $args = array
());
```

请求示例

示例一： 使用模板

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，h
ttps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
```

```
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->createMediaAnimationJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'Animation',
'QueueId' => 'p81e648af2aee49688xxxxxxxxxxxxxxx',
'Input' => array(
'Object' => 'video01.mp4'
),
'Operation' => array(
'TemplateId' => 't1de276cbdab16xxxxxxxxxxxxxxxxxxxxx',
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'Animation.gif',
),
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
),
'CallBack' => '',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二： 自定义参数

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
```

```
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->createMediaAnimationJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'Animation',
'QueueId' => 'p81e648af2aee49688xxxxxxxxxxxxxxx',
'Input' => array(
'Object' => 'video01.mp4'
),
'Operation' => array(
'Animation' => array(
'Container' => array(
'Format' => '',
),
'Video' => array(
'Codec' => '',
'Width' => '',
'Height' => '',
'Fps' => '',
'AnimateOnlyKeepKeyFrame' => '',
'AnimateTimeIntervalOfFrame' => '',
'AnimateFramesPerSecond' => '',
'Quality' => '',
),
'TimeInterval' => array(
'Start' => '',
'Duration' => '',
),
),
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'Animation.gif',
),
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
),
'CallBack' => '',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
```

```
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	创建任务的 Tag：Animation	String	是
Input	Request	待操作的媒体信息	Container	是
Operation	Request	操作规则	Container	是
QueueId	Request	任务所在的队列 ID	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型	String	否
CallBack	Request	任务回调地址，优先级高于队列的回调地址。设置为 no 时，表示队列的回调地址不产生回调	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	媒体文件名	String	是

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Animation	Request.Operation	指定该任务的参数	Container	否
TemplateId	Request.Operation	指定的模板 ID	String	否
Output	Request.Operation	结果输出地址	Container	是

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
UserData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否
JobLevel	Request.Operation	任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0	String	否

注意：

优先使用 TemplateId，无 TemplateId 时使用 Animation。

Container 类型 Animation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Container	Request.Operation.Animation	同动图模板 CreateMediaTemplate 接口中的 Request.Container	Container	否
Video	Request.Operation.Animation	同动图模板 CreateMediaTemplate 接口中的 Request.Video	Container	否
TimeInterval	Request.Operation.Animation	同动图模板 CreateMediaTemplate 接口中的 Request.TimeInterval	Container	否

Container 类型 Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域	String	是
Bucket	Request.Operation.Output	存储结果的存储桶	String	是
Object	Request.Operation.Output	输出结果的文件名	String	是

返回结果示例

版权所有：腾讯云计算（北京）有限责任公司

```
GuzzleHttp\Command\Result Object
(
[Body] => GuzzleHttp\Psr7\Stream Object
(
[stream:GuzzleHttp\Psr7\Stream:private] => Resource id #88
[size:GuzzleHttp\Psr7\Stream:private] =>
[seekable:GuzzleHttp\Psr7\Stream:private] => 1
[readable:GuzzleHttp\Psr7\Stream:private] => 1
[writable:GuzzleHttp\Psr7\Stream:private] => 1
[uri:GuzzleHttp\Psr7\Stream:private] => php://temp
[customMetadata:GuzzleHttp\Psr7\Stream:private] => Array
(
)
)
[RequestId] => NjI2MTQHOAIHDOIDUOIA7879ADJOIMDc2YTk=
[ContentType] => application/xml
[ContentLength] => 797
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-beijing.myqcloud.com/jobs
[Response] => Array
(
[JobsDetail] => Array
(
[Code] => Success
[CreationTime] => 2022-04-21T19:29:36+0800
[EndTime] => -
[Input] => Array
(
[BucketId] => examplebucket-1250000000
[Object] => video01.mp4
[Region] => ap-beijing
)
[JobId] => j5318zx9c890zx8c0z98c0fa6355174de
[Message] =>
[Operation] => Array
(
[Output] => Array
(
[Bucket] => examplebucket-1250000000
[Object] => Animation.gif
[Region] => ap-beijing
)
[TemplateId] => t1de276cbdz78xcz09xc80zx901af71
[TemplateName] => Animation-gif-1
[UserData] => xxx
[JobLevel] => 0
```

```
)
[QueueId] => p81e648af2aeez87c6z78xc7z98xc79ca86
[StartTime] => -
[State] => Submitted
[Tag] => Animation
)
)
)
```

截图任务接口

最近更新时间：2023-02-07 15:10:16

简介

本文档提供关于提交截图任务的 API 概览和 SDK 示例代码。

API	操作描述
提交截图任务	提交一个截图任务

提交截图任务

功能说明

用于提交一个截图任务。

方法原型

```
public Guzzle\Service\Resource\Model createMediaSnapshotJobs(array $args = array
());
```

请求示例

示例一： 使用模板

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，h
ttps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

```
try {
$result = $cosClient->createMediaSnapshotJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'Snapshot',
'QueueId' => 'asdadadfafsdkjhfjghdfjg',
'CallBack' => 'https://example.com/callback',
'Input' => array(
'Object' => 'video01.mp4'
),
'Operation' => array(
'TemplateId' => 'asdfafiahfiushdfisdhfuis',
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'snapshot-${Number}.jpg',
// 'SpriteObject' => 'sprite-${Number}.jpg',
),
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
),
// 'CallBackMqConfig' => array(
// 'MqRegion' => '',
// 'MqMode' => '',
// 'MqName' => '',
// ),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二： 自定义参数

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
```

```
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->createMediaSnapshotJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'Snapshot',
'QueueId' => 'asdadadfafsdkjhfjghdfjg',
'CallBack' => 'https://example.com/callback',
'Input' => array(
'Object' => 'video01.mp4'
),
'Operation' => array(
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'snapshot-${Number}.jpg',
),
'Snapshot' => array(
'Mode' => 'Average',
'Start' => 3,
'TimeInterval' => '',
'Count' => 3,
'Width' => '1280',
'Height' => '',
),
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	创建任务的 Tag：Snapshot	String	是
Input	Request	待操作的媒体信息	Container	是
Operation	Request	操作规则	Container	是
QueueId	Request	任务所在的队列 ID	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型	String	否
CallBack	Request	任务回调地址，优先级高于队列的回调地址。设置为 no 时，表示队列的回调地址不产生回调	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	媒体文件名	String	是

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Snapshot	Request.Operation	指定该任务的参数，同创建截图模板 CreateMediaTemplate 接口中的 Request.Snapshot	Container	否
TemplateId	Request.Operation	指定的模板 ID	String	否
Output	Request.Operation	结果输出地址	Container	是
UserData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
JobLevel	Request.Operation	任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0	String	否

注意：

优先使用 TemplateId，无 TemplateId 时使用 Snapshot。

Container 类型 Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域	String	是
Bucket	Request.Operation.Output	存储结果的存储桶	String	是
Object	Request.Operation.Output	结果文件的名字。**必须包含 ${Number} 参数。**如 Object 为 snapshot-${Number}.jpg	String	否
SpriteObject	Request.Operation.Output	雪碧图的名字。**必须包含 ${Number} 参数。**如 Object 为 sprite-${Number}.jpg ** 仅支持 jpg 格式	String	否

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[Body] => GuzzleHttp\Psr7\Stream Object
(
[stream:GuzzleHttp\Psr7\Stream:private] => Resource id #88
[size:GuzzleHttp\Psr7\Stream:private] =>
[seekable:GuzzleHttp\Psr7\Stream:private] => 1
[readable:GuzzleHttp\Psr7\Stream:private] => 1
[writable:GuzzleHttp\Psr7\Stream:private] => 1
[uri:GuzzleHttp\Psr7\Stream:private] => php://temp
[customMetadata:GuzzleHttp\Psr7\Stream:private] => Array
(
)
)
```

版权所有：腾讯云计算（北京）有限责任公司

```
[RequestId] => NjI2MTNkYAHDUIAHDRfNjQ0OF8yMGI4MjQ=
[ContentType] => application/xml
[ContentLength] => 829
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-beijing.myqcloud.com/jobs
[Response] => Array
(
[JobsDetail] => Array
(
[Code] => Success
[CreationTime] => 2022-04-21T19:19:25+0800
[EndTime] => -
[Input] => Array
(
[BucketId] => examplebucket-1250000000
[Object] => video01.mp4
[Region] => ap-beijing
)
[JobId] => je6dcd3d6a80d98a09ds8ac4214b9b
[Message] =>
[Operation] => Array
(
[Output] => Array
(
[Bucket] => examplebucket-1250000000
[Object] => snapshot-${Number}.jpg
[Region] => ap-beijing
[SpriteObject] =>
)
[TemplateId] => t0778zc908zc908z0c8z0adcffd54a
[TemplateName] => snapshot_1280*720_3
[UserData] => xxx
[JobLevel] => 0
)
[QueueId] => p81e648afz7zc08c0z8xc09e086
[StartTime] => -
[State] => Submitted
[Tag] => Snapshot
)
)
)
```

转码任务接口

最近更新时间：2023-02-07 15:10:16

简介

本文档提供关于提交转码任务的 API 概览和 SDK 示例代码。

API	操作描述
提交转码任务	提交转码任务

提交转码任务

功能说明

提交转码任务。

方法原型

```
public Guzzle\Service\Resource\Model createMediaTranscodeJobs(array $args = array
());
```

请求示例

示例一： 使用模板

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，h
ttps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

```
try {
$result = $cosClient->createMediaTranscodeJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'Transcode',
'QueueId' => 'paaf4fce5521a40888a3034a5de80f6ca',
'Input' => array(
'Object' => 'example.mp4'
),
'Operation' => array(
'TemplateId' => 't04e1ab86554984f1aa17c062fbf6c007c',
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'video02.mp4',
),
'Watermark' => array(
array(
'Type' => 'Text',
'LocMode' => 'Absolute',
'Dx' => '64',
'Dy' => '64',
'Pos' => 'TopRight',
'Text' => array(
'Text' => '第一个水印',
'FontSize' => '30',
'FontType' => 'simfang.ttf',
'FontColor' => '#99ff00',
'Transparency' => '100', // 不透明度
),
),
array(
'Type' => 'Text',
'LocMode' => 'Absolute',
'Dx' => '64',
'Dy' => '64',
'Pos' => 'TopLeft',
'Text' => array(
'Text' => '第二个水印',
'FontSize' => '30',
'FontType' => 'simfang.ttf',
'FontColor' => '#99ff00',
'Transparency' => '100', // 不透明度
),
```

```
),
),
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二： 自定义参数

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->createMediaTranscodeJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'Transcode',
'QueueId' => 'asdadadfafsdkjhfjghdfjg',
'CallBack' => 'https://example.com/callback',
'Input' => array(
'Object' => 'video01.mp4'
),
'Operation' => array(
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'video01.mkv',
```

```
),
'Transcode' => array(
'Container' => array(
'Format' => 'mp4'
),
'Video' => array(
'Codec' => 'H.264',
'Profile' => 'high',
'Bitrate' => '1000',
'Preset' => 'medium',
'Width' => '1280',
'Fps' => '30',
),
'Audio' => array(
'Codec' => 'aac',
'Samplerate' => '44100',
'Bitrate' => '128',
'Channels' => '4',
),
'TransConfig' => array(
'AdjDarMethod' => 'scale',
'IsCheckReso' => 'false',
'ResoAdjMethod' => '1',
),
'TimeInterval' => array(
'Start' => '0',
'Duration' => '60',
),
),
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	创建任务的 Tag：Transcode	String	是

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	待操作的媒体信息	Container	是
Operation	Request	操作规则	Container	是
QueueId	Request	任务所在的队列 ID	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型	String	否
CallBack	Request	任务回调地址，优先级高于队列的回调地址。设置为 no 时，表示队列的回调地址不产生回调	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	媒体文件名	String	是

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
TemplateId	Request.Operation	指定的模板 ID	String	否
Transcode	Request.Operation	指定转码模板参数	Container	否
WatermarkTemplateId	Request.Operation	指定的水印模板 ID，可以传多个水印模板 ID	String	否
Watermark	Request.Operation	指定水印模板参数，同创建水印模板 CreateMediaTemplate 接口中的 Request.Watermark	Container	否
RemoveWatermark	Request.Operation	指定去除水印参数, H265暂不支持次参数	Container	否
DigitalWatermark	Request.Operation	指定数字水印参数	Container	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Output	Request.Operation	结果输出地址	Container	是
UserData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否
JobLevel	Request.Operation	任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0	String	否

说明：

对于转码参数，优先使用优先使用 TemplateId，无 TemplateId 时使用 Transcode；对于水印参数，可以使用 WatermarkTemplateId 或 Watermark 设置，WatermarkTemplateId 优先级更高。

Container 类型 Transcode 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
TimeInterval	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.TimeInterval	Container	否
Container	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.Container	Container	否
Video	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.Video	Container	否
Audio	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.Audio	Container	否
TransConfig	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.TransConfig	Container	否
AudioMix	Request.Operation.Transcode	混音参数，详情见AudioMix	Container 数组	否

版权所有：腾讯云计算（北京）有限责任公司

Container 类型 RemoveWatermark 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
Dx	Request.Operation.RemoveWatermark	距离左上角原点 x 偏移，范围为[1, 4096]	string	是
Dy	Request.Operation.RemoveWatermark	距离左上角原点 y 偏移，范围为[1, 4096]	string	是
Width	Request.Operation.RemoveWatermark	宽，范围为[1, 4096]	string	是
Height	Request.Operation.RemoveWatermark	高，范围为[1, 4096]	string	是

Container 类型 DigitalWatermark 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
Message	Request.Operation.DigitalWatermark	数字水印嵌入的字符串信息，长度不超过64个字符，仅支持中文、英文、数字、_、-和*	string	是
Type	Request.Operation.DigitalWatermark	水印类型，当前仅可设置为 Text	String	是
Version	Request.Operation.DigitalWatermark	水印版本，当前仅可设置为 V1	String	是
IgnoreError	Request.Operation.DigitalWatermark	当添加水印失败是否忽略错误继续执行任务，限制为 true/false，默认为false	string	是

Container 类型 Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域	String	是
Bucket	Request.Operation.Output	存储结果的存储桶	String	是
Object	Request.Operation.Output	输出结果的文件名	String	是

返回结果示例

版权所有：腾讯云计算（北京）有限责任公司

```
GuzzleHttp\Command\Result Object
(
[Body] => GuzzleHttp\Psr7\Stream Object
(
[stream:GuzzleHttp\Psr7\Stream:private] => Resource id #88
[size:GuzzleHttp\Psr7\Stream:private] =>
[seekable:GuzzleHttp\Psr7\Stream:private] => 1
[readable:GuzzleHttp\Psr7\Stream:private] => 1
[writable:GuzzleHttp\Psr7\Stream:private] => 1
[uri:GuzzleHttp\Psr7\Stream:private] => php://temp
[customMetadata:GuzzleHttp\Psr7\Stream:private] => Array
(
)
)
[RequestId] => NjI2MTAzasdaHUISDHUINzE3YV8xZWVmNjU=
[ContentType] => application/xml
[ContentLength] => 1554
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/jobs
[Response] => Array
(
[JobsDetail] => Array
(
[Code] => Success
[CreationTime] => 2022-04-21T15:10:15+0800
[EndTime] => -
[Input] => Array
(
[BucketId] => examplebucket-1250000000
[Object] => video01.mp4
[Region] => ap-beijing
)
[JobId] => j1815a4c81h2kj31231gh31kj2f9e223404e
[Message] => Array
(
)
[Operation] => Array
(
[Output] => Array
(
[Bucket] => examplebucket-1250000000
[Object] => Transcode.flv
[Region] => ap-beijing
)
[TemplateId] => t0b612860as90d8a09sd65dca38
[TemplateName] => FLV-SD
```

```
[UserData] => xxx
[JobLevel] => 0
[Watermark] => Array
(
[0] => Array
(
[Dx] => 64
[Dy] => 64
[EndTime] => Array
(
)
[LocMode] => Absolute
[Pos] => TopRight
[StartTime] => Array
(
)
[Text] => Array
(
[FontColor] => #99ff00
[FontSize] => 30
[FontType] => simfang.ttf
[Text] => 第一个水印
[Transparency] => 100
)
[Type] => Text
)
[1] => Array
(
[Dx] => 64
[Dy] => 64
[EndTime] => Array
(
)
[LocMode] => Absolute
[Pos] => TopLeft
[StartTime] => Array
(
)
[Text] => Array
(
[FontColor] => #99ff00
[FontSize] => 30
[FontType] => simfang.ttf
[Text] => 第二个水印
[Transparency] => 100
)
[Type] => Text
```

```
)
)
)
[Progress] => 0
[QueueId] => p81e648a7s9d88a09sd89a0757be086
[StartTime] => -
[State] => Submitted
[Tag] => Transcode
)
)
)
```

拼接任务接口

最近更新时间：2023-02-07 15:10:16

简介

本文档提供关于提交拼接任务的 API 概览和 SDK 示例代码。

API	操作描述
提交拼接任务	提交一个拼接任务

提交拼接任务

功能说明

用于提交一个拼接任务。

方法原型

```
public Guzzle\Service\Resource\Model createMediaConcatJobs(array $args = array
());
```

请求示例

示例一： 使用模板

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，h
ttps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

```
try {
$result = $cosClient->createMediaConcatJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'Concat',
'QueueId' => 'asdadadfafsdkjhfjghdfjg',
'CallBack' => 'https://example.com/callback',
'Input' => array(
'Object' => 'video01.mp4'
),
'Operation' => array(
'TemplateId' => 'asdfafiahfiushdfisdhfuis',
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'concat-video02.mp4',
),
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二： 自定义参数

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

版权所有：腾讯云计算（北京）有限责任公司

```
try {
$result = $cosClient->createMediaConcatJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'Concat',
'QueueId' => 'asdadadfafsdkjhfjghdfjg',
'CallBack' => 'https://example.com/callback',
'Input' => array(
'Object' => 'video01.mp4'
),
'Operation' => array(
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'concat-video03.mp4',
),
'ConcatTemplate' => array(
'ConcatFragments' => array(
array(
'Url' => 'https://example.com/video01.mp4',
'Mode' => 'Start',
// 'StartTime' => '0',
// 'EndTime' => '7',
),
array(
'Url' => 'https://example.com/video02.mp4',
'Mode' => 'Start',
// 'StartTime' => '0',
// 'EndTime' => '10',
),
// ... repeated
),
'Index' => 1,
'Container' => array(
'Format' => 'mp4'
),
'Audio' => array(
'Codec' => 'mp3',
'Samplerate' => '',
'Bitrate' => '',
'Channels' => '',
),
'Video' => array(
'Codec' => 'H.264',
```

```
'Bitrate' => '1000',
'Width' => '1280',
'Height' => '',
'Fps' => '30',
),
),
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	创建任务的 Tag：Concat	String	是
Input	Request	待操作的媒体信息	Container	否
Operation	Request	操作规则	Container	是
QueueId	Request	任务所在的队列 ID	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型	String	否
CallBack	Request	任务回调地址，优先级高于队列的回调地址。设置为 no 时，表示队列的回调地址不产生回调	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	媒体文件名	String	是

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
ConcatTemplate	Request.Operation	指定拼接参数	Container	否
TemplateId	Request.Operation	指定的模板 ID	String	否
Output	Request.Operation	结果输出地址	Container	是
JobLevel	Request.Operation	任务优先级，级别限制：0、1、2。级别越大任务优先级越高，默认为0	String	否

注意：

优先使用 TemplateId，无 TemplateId 时使用 ConcatTemplate。

Container 类型 ConcatTemplate 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选	默
ConcatFragment	Request.Operation.ConcatTemplate	拼接节点	Container 数组	否	无
Audio	Request.Operation.ConcatTemplate	音频参数，同创建拼接模板 CreateMediaTemplate 接口中的 Request.ConcatTemplate.Audio	Container	否	无
Video	Request.Operation.ConcatTemplate	视频参数，同创建拼接模板 CreateMediaTemplate 接口中的 Request.ConcatTemplate.Video	Container	否	无
Container	Request.Operation.ConcatTemplate	封装格式，同创建拼接模板 CreateMediaTemplate 接口中的 Request.ConcatTemplate.Container	Container	是	无

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选	默
AudioMix	Request.Operation.ConcatTemplate	混音参数, 详情见 AudioMix	Container 数组	否	无
Index	Request.Operation.ConcatTemplate	Input 节点位于 ConcatFragment 序列索引	String	否	0
DirectConcat	Request.Operation.ConcatTemplate	简单拼接方式（不转码直接拼接），其他的视频和音频参数失效	String	否	fa

Container 类型 ConcatFragment 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选	默认值	限制
Url	Request.Operation.ConcatTemplate.ConcatFragment	拼接对象地址	String	是	无	同 bucket 对象文件
FragmentIndex	Request.Operation.ConcatTemplate.ConcatFragment	拼接对象的索引位置	String	否	0	大于等于0的整数
StartTime	Request.Operation.ConcatTemplate.ConcatFragment	开始时间	String	否	视频开始	[0 视频时长]单位为秒
EndTime	Request.Operation.ConcatTemplate.ConcatFragment	结束时间	String	否	视频结束	[0 视频时长]单位为秒

Container 类型 Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域	String	是
Bucket	Request.Operation.Output	存储结果的存储桶	String	是

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Operation.Output	输出结果的文件名	String	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[Body] => GuzzleHttp\Psr7\Stream Object
(
[stream:GuzzleHttp\Psr7\Stream:private] => Resource id #88
[size:GuzzleHttp\Psr7\Stream:private] =>
[seekable:GuzzleHttp\Psr7\Stream:private] => 1
[readable:GuzzleHttp\Psr7\Stream:private] => 1
[writable:GuzzleHttp\Psr7\Stream:private] => 1
[uri:GuzzleHttp\Psr7\Stream:private] => php://temp
[customMetadata:GuzzleHttp\Psr7\Stream:private] => Array
(
)
)
[RequestId] => NjI2MjFhOTHASIUDHAUISDHl8yMjMxNmE=
[ContentType] => application/xml
[ContentLength] => 792
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-beijing.myqcloud.com/jobs
[Response] => Array
(
[JobsDetail] => Array
(
[Code] => Success
[CreationTime] => 2022-04-22T11:01:42+0800
[EndTime] => -
[Input] => Array
(
[BucketId] => examplebucket-1250000000
[Object] => video01.mp4
[Region] => ap-beijing
)
[JobId] => j898f573asd780a8sd90a8sd09e30cf31f
[Message] =>
[Operation] => Array
(
[Output] => Array
(
[Bucket] => examplebucket-1250000000
[Object] => concat-video01.mp4
```

```
[Region] => ap-beijing
)
[TemplateId] => t17b9c2e7a0s8d0a8d0a9d43552f648e
[TemplateName] => concat-1
[UserData] => xxx
[TemplateName] => 0
)
[QueueId] => p81e648af2aeas78d0a8d09a8d09757be086
[StartTime] => -
[State] => Submitted
[Tag] => Concat
)
)
)
```

智能封面任务接口

最近更新时间：2023-05-26 17:37:04

简介

本文档提供关于提交智能封面任务的 API 概览和 SDK 示例代码。

API	操作描述
提交智能封面任务	提交一个智能封面任务

提交智能封面任务

功能说明

用于提交一个智能封面任务。

方法原型

```
public Guzzle\Service\Resource\Model createMediaSmartCoverJobs(array $args = arra
y());
```

请求示例

```
<?php

require dirname(__FILE__) . '/../vendor/autoload.php';

$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，h
ttps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

版权所有：腾讯云计算（北京）有限责任公司

```
try {
// 提交智能封面任务 https://intl.cloud.tencent.com/document/product/436/49054
$result = $cosClient->createMediaSmartCoverJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'SmartCover',
'QueueId' => 'p81e648afxxxxxxxxxxxxxxxxx',
'Input' => array(
'Object' => 'video01.mp4'
),
'Operation' => array(
// 'TemplateId' => '', // 使用模板
'SmartCover' => array(
'Format' => '',
'Width' => '',
'Height' => '',
'Count' => '',
'DeleteDuplicates' => '',
),
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'SmartCover-${Number}.jpg',
),
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
),
'CallBack' => '',
// 'CallBackFormat' => '',
// 'CallBackType' => '',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	创建任务的 Tag：SmartCover	String	是
Input	Request	待操作的媒体信息	Container	是
Operation	Request	操作规则	Container	是
QueueId	Request	任务所在的队列 ID	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型	String	否
CallBack	Request	任务回调地址，优先级高于队列的回调地址。设置为 no 时，表示队列的回调地址不产生回调	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	媒体文件名	String	是

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
TemplateId	Request.Operation	智能封面模板 ID	String	否
SmartCover	Request.Operation	封面配置	Container	否
Output	Request.Operation	结果输出地址	Container	是
UserData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否
JobLevel	Request.Operation	任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0	String	否

版权所有：腾讯云计算（北京）有限责任公司

说明：

如果同时设置了 TemplateId 和 SmartCover，则优先使用 TemplateId

Container 类型 SmartCover 的具体数据类型描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选	默认值	限制
Format	Request.Operation.SmartCover	封面图片类型	String	是	无	png、jpg、webp
Width	Request.Operation.SmartCover	封面图片宽度	String	是	无	1. 值范围：[128, 4096] 2. 单位：px
Height	Request.Operation.SmartCover	封面图片高度	String	是	无	1. 值范围：[128, 4096] 2. 单位：px
Count	Request.Operation.SmartCover	封面数量	String	否	3	1. 值范围：[1, 10]
DeleteDuplicates	Request.Operation.SmartCover	封面是否去重	String	否	false	true/false

Container 类型 Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域	String	是
Bucket	Request.Operation.Output	存储结果的存储桶	String	是

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Operation.Output	输出结果的文件名。 **必须包含 ${Number} 参数。** 如 Object 为 my-new-cover-${Number}.jpg，对应实际3张输出结果时，分别为 my-new-cover-0.jpg my-new-cover-1.jpg my-new-cover-2.jpg	String	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[Body] => GuzzleHttp\Psr7\Stream Object
(
[stream:GuzzleHttp\Psr7\Stream:private] => Resource id #88
[size:GuzzleHttp\Psr7\Stream:private] =>
[seekable:GuzzleHttp\Psr7\Stream:private] => 1
[readable:GuzzleHttp\Psr7\Stream:private] => 1
[writable:GuzzleHttp\Psr7\Stream:private] => 1
[uri:GuzzleHttp\Psr7\Stream:private] => php://temp
[customMetadata:GuzzleHttp\Psr7\Stream:private] => Array
(
)

)

[RequestId] => NjI2MASOIDUAOIDUJIAOJDIOADJyMjNjZDE=
[ContentType] => application/xml
[ContentLength] => 832
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-beijing.myqcloud.com/jobs
[Response] => Array
(
[JobsDetail] => Array
(
[Code] => Success
[CreationTime] => 2022-04-22T11:12:33+0800
[EndTime] => -
[Input] => Array
(
[BucketId] => examplebucket-1250000000
```

版权所有：腾讯云计算（北京）有限责任公司

```
[Object] => video01.mp4
[Region] => ap-beijing
)

[JobId] => j0d9bc17z890zx8c9s877z97187147
[Message] =>
[Operation] => Array
(
[Output] => Array
(
[Bucket] => examplebucket-1250000000
[Object] => SmartCover-${Number}.jpg
[Region] => ap-beijing
)

[SmartCover] => Array
(
[Count] =>
[DeleteDuplicates] => false
[Format] =>
[Height] =>
[Width] =>
)
[UserData] => xxx
[JobLevel] => 0
)

[QueueId] => p81e648afzx8c09z8xc09zx8c097be086
[StartTime] => -
[State] => Submitted
[Tag] => SmartCover
)

)

)
```

多任务接口

最近更新时间：2022-05-23 14:41:10

简介

本文档提供关于提交多任务的 API 概览和 SDK 示例代码。

API	操作描述
提交多任务	用于提交多个任务。

提交多任务

功能说明

用于提交多个任务。

方法原型

```
public Guzzle\Service\Resource\Model createMediaJobs(array $args = array());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
// 多任务接口 https://cloud.tencent.com/document/product/436/58335
$result = $cosClient->CreateMediaJobs(array(
```

```
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'Transcode', // 可选，单一任务时，优先以Operation.Tag为准，当Operation无Tag参数时，该参数生效
'QueueId' => 'paaf4fce5521a40888a303xxxxxxxxxxxxxx',
'CallBack' => '',
'Input' => array(
'Object' => 'example.mp4'
),
'Operation' => array(
array(
'Tag' => 'Transcode',
'TemplateId' => 't04e1ab86554984f1aa17cxxxxxxxxxxxxxx',
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'video01.mp4',
),
'WatermarkTemplateId' => array(
't112d18d9b2a9b430e91dxxxxxxxxxxxxxx',
),
),
array(
'Tag' => 'Transcode',
'TemplateId' => 't04e1ab86554984f1aa17xxxxxxxxxxxxxx',
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'video02.mp4',
),
'WatermarkTemplateId' => array(
't1bf713bb5c6a5496e859axxxxxxxxxxxxxx',
),
),
),
));
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	待操作的媒体信息	Container	是
Operation	Request	操作规则，个数不超过6个	Container	是
QueueId	Request	任务所在的队列 ID	String	是

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	媒体文件名	String	是

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request.Operation	任务类型: Animation、Transcode、SmartCover、Snapshot	Container	是
TemplateId	Request.Operation	指定的模板 ID	String	是
Output	Request.Operation	结果输出地址	Container	是

Container 类型 Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域	String	是
Bucket	Request.Operation.Output	存储结果的存储桶	String	是
Object	Request.Operation.Output	输出结果的文件名	String	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[Body] => GuzzleHttp\Psr7\Stream Object
(
[stream:GuzzleHttp\Psr7\Stream:private] => Resource id #88
[size:GuzzleHttp\Psr7\Stream:private] =>
```

版权所有：腾讯云计算（北京）有限责任公司

```
[seekable:GuzzleHttp\Psr7\Stream:private] => 1
[readable:GuzzleHttp\Psr7\Stream:private] => 1
[writable:GuzzleHttp\Psr7\Stream:private] => 1
[uri:GuzzleHttp\Psr7\Stream:private] => php://temp
[customMetadata:GuzzleHttp\Psr7\Stream:private] => Array
(
)
)
[RequestId] => NjI2MTAJOIDSjoiajsdoSDOASDHJoimY2YTQ=
[ContentType] => application/xml
[ContentLength] => 1731
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-beijing.myqcloud.com/jobs
[Response] => Array
(
[JobsDetail] => Array
(
[0] => Array
(
[Code] => Success
[CreationTime] => 2022-04-21T18:03:31+0800
[EndTime] => -
[Input] => Array
(
[BucketId] => examplebucket-1250000000
[Object] => video01.mp4
[Region] => ap-beijing
)
[JobId] => j4c211cas9a87sd90d80a988627cb30
[Message] =>
[Operation] => Array
(
[Output] => Array
(
[Bucket] => examplebucket-1250000000
[Object] => Transcode-01.flv
[Region] => ap-beijing
)
[TemplateId] => t0b6128z8c098z09xcsd89s0a5dca38
[TemplateName] => FLV-SD
[WatermarkTemplateId] => t185z90x8c9z0da5sd67a5a19c
)
[Progress] => 0
[QueueId] => pzx8c09za90sd890a8d09aa0757be086
[StartTime] => -
[State] => Submitted
[Tag] => Transcode
```

```
)
[1] => Array
(
[Code] => Success
[CreationTime] => 2022-04-21T18:03:31+0800
[EndTime] => -
[Input] => Array
(
[BucketId] => examplebucket-1250000000
[Object] => video01.mp4
[Region] => ap-beijing
)
[JobId] => j4cz9x7c8z9xcsda98sd09ad80627cb30
[Message] =>
[Operation] => Array
(
[Output] => Array
(
[Bucket] => examplebucket-1250000000
[Object] => Transcode-02.flv
[Region] => ap-beijing
)
[TemplateId] => t0bz98xc789zxs9d80a8sd90asd37f83c
[TemplateName] => FLV-HD
[WatermarkTemplateId] => t18zc8z90csd876a78sdas87d8a19c
)
[Progress] => 0
[QueueId] => p81zxc908zaf2aee496a09s8d90as8d90
[StartTime] => -
[State] => Submitted
[Tag] => Transcode
)
)
)
)
```

视频增强任务接口

最近更新时间：2023-02-07 15:10:16

简介

本文档提供关于提交视频增强任务的 API 概览和 SDK 示例代码。

API	操作描述
提交视频增强任务	提交一个视频增强任务

提交视频增强任务

功能说明

用于提交一个视频增强任务。

方法原型

```
public Guzzle\Service\Resource\Model createMediaVideoProcessJobs(array $args = array());
```

请求示例

示例一： 使用模板

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

```
try {
$result = $cosClient->createMediaVideoProcessJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'VideoProcess',
'QueueId' => 'p81e648afxxxxxxxxxxxxxxxxx',
'Input' => array(
'Object' => 'video01.mp4'
),
'Operation' => array(
'TemplateId' => 't13466f1ea41a14c0xxxxxxxxxxxxx', // 视频增强模板 ID
'TranscodeTemplateId' => 't0b6a845f5e42847bd81xxxxxxxxxxxx', // 转码模板 ID
'WatermarkTemplateId' => 't185e2e24551b24259a0xxxxxxxxxxxx', // 水印模板 ID
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'VideoProcess.flv',
),
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
),
'CallBack' => '',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二： 自定义参数

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
```

```
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->createMediaVideoProcessJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'VideoProcess',
'QueueId' => 'p81e648afxxxxxxxxxxxxxxxxx',
'Input' => array(
'Object' => 'video01.mp4'
),
'Operation' => array(
'VideoProcess' => array(
'ColorEnhance' => array(
'Enable' => '',
'Contrast' => '',
'Correction' => '',
'Saturation' => '',
),
'MsSharpen' => array(
'Enable' => '',
'SharpenLevel' => '',
),
),
'Transcode' => array(
'Tag' => '',
'Name' => '',
'Container' => array(
'Format' => '',
),
'Video' => array(
'Codec' => '',
'Width' => '',
'Height' => '',
'Fps' => '',
'Remove' => '',
'Profile' => '',
'Bitrate' => '',
'Crf' => '',
'Gop' => '',
'Preset' => '',
'Bufsize' => '',
'Maxrate' => '',
'HlsTsTime' => '',
'Pixfmt' => '',
'LongShortMode' => '',
),
```

```
'TimeInterval' => array(
'Start' => '',
'Duration' => '',
),
'Audio' => array(
'Codec' => '',
'Samplerate' => '',
'Bitrate' => '',
'Channels' => '',
'Remove' => '',
'KeepTwoTracks' => '',
'SwitchTrack' => '',
'SampleFormat' => '',
),
'TransConfig' => array(
'AdjDarMethod' => '',
'IsCheckReso' => '',
'ResoAdjMethod' => '',
'IsCheckVideoBitrate' => '',
'VideoBitrateAdjMethod' => '',
'IsCheckAudioBitrate' => '',
'AudioBitrateAdjMethod' => '',
'DeleteMetadata' => '',
'IsHdr2Sdr' => '',
'HlsEncrypt' => array(
'IsHlsEncrypt' => '',
'UriKey' => '',
),
),
),
'Watermark' => array(
'Type' => '',
'Pos' => '',
'LocMode' => '',
'Dx' => '',
'Dy' => '',
'StartTime' => '',
'EndTime' => '',
'Image' => array(
'Url' => '',
'Mode' => '',
'Width' => '',
'Height' => '',
'Transparency' => '',
'Background' => '',
),
'Text' => array(
```

```
'FontSize' => '',
'FontType' => '',
'FontColor' => '',
'Transparency' => '',
'Text' => '',
),
),
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'VideoProcess.flv',
),
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
),
'CallBack' => '',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	创建任务的 Tag：VideoProcess	String	是
Input	Request	待操作的媒体信息	Container	是
Operation	Request	操作规则	Container	是
QueueId	Request	任务所在的队列 ID	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型	String	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
CallBack	Request	任务回调地址，优先级高于队列的回调地址。设置为 no 时，表示队列的回调地址不产生回调	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	媒体文件名	String	是

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
VideoProcess	Request.Operation	指定视频增强模板参数，不能与 TemplateId 同时为空	Container	否
TemplateId	Request.Operation	指定的模板 ID，不能与 VideoProcess 同时为空	String	否
Transcode	Request.Operation	指定转码模板参数，不能与 TranscodeTemplateId 同时为空	Container	否
TranscodeTemplateId	Request.Operation	指定的转码模板 ID，优先使用模板 ID，不能与 Transcode 同时为空	String 数组	否
Watermark	Request.Operation	指定水印模板参数，同创建水印模板 CreateMediaTemplate 接口的 Request.Watermark, 最多传3个	Container 数组	否
WatermarkTemplateId	Request.Operation	指定的水印模板 ID，可以传多个水印模板 ID，最多传3个，优先使用模板 ID	String	否
DigitalWatermark	Request.Operation	指定数字水印参数	Container	否
Output	Request.Operation	结果输出地址	Container	是

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
JobLevel	Request.Operation	任务优先级，级别限制：0、1、2。级别越大任务优先级越高，默认为0	String	否

说明：

提交视频增强任务必须传入转码参数。对于视频增强参数，优先使用 TemplateId，无 TemplateId 时使用 VideoProcess；对于转码参数，优先使用 TranscodeTemplateId，无 TranscodeTemplateId 时使用 Transcode；对于水印参数，可以使用 WatermarkTemplateId 或 Watermark 设置，WatermarkTemplateId 优先级更高。

Container 类型 VideoProcess 的具体数据描述如下：

节点名称（关键字）	父节点	描述
ColorEnhance	Request.Operation.VideoProcess	同创建视频增强模板 CreateMediaTemplate 接口中的 Request.ColorEnhance
MsSharpen	Request.Operation.VideoProcess	同创建视频增强模板 CreateMediaTemplate 接口中的 Request.MsSharpen

Container 类型 Transcode 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
TimeInterval	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.TimeInterval	Container	否
Container	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.Container	Container	否
Video	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.Video	Container	否

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	必选
Audio	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.Audio	Container	否
TransConfig	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.TransConfig	Container	否
AudioMix	Request.Operation.Transcode	混音参数，详情见 AudioMix	Container 数组	否

Container 类型 DigitalWatermark 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
Message	Request.Operation.DigitalWatermark	数字水印嵌入的字符串信息，长度不超过64个字符，仅支持中文、英文、数字、_、-和*	string	是
Type	Request.Operation.DigitalWatermark	水印类型，当前仅可设置为 Text	String	是
Version	Request.Operation.DigitalWatermark	水印版本，当前仅可设置为 V1	String	是
IgnoreError	Request.Operation.DigitalWatermark	当添加水印失败是否忽略错误继续执行任务，限制为 true/false，默认为 false	string	是

Container 类型 Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域	String	是
Bucket	Request.Operation.Output	存储结果的存储桶	String	是
Object	Request.Operation.Output	输出结果的文件名	String	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
```

```
[Body] => GuzzleHttp\Psr7\Stream Object
(
[stream:GuzzleHttp\Psr7\Stream:private] => Resource id #88
[size:GuzzleHttp\Psr7\Stream:private] =>
[seekable:GuzzleHttp\Psr7\Stream:private] => 1
[readable:GuzzleHttp\Psr7\Stream:private] => 1
[writable:GuzzleHttp\Psr7\Stream:private] => 1
[uri:GuzzleHttp\Psr7\Stream:private] => php://temp
[customMetadata:GuzzleHttp\Psr7\Stream:private] => Array
(
)
)
[RequestId] => NjI2MjNjYHAOIHDOIADJHOIA18yMjFlMWE=
[ContentType] => application/xml
[ContentLength] => 965
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-beijing.myqcloud.com/jobs
[Response] => Array
(
[JobsDetail] => Array
(
[Code] => Success
[CreationTime] => 2022-04-22T11:19:15+0800
[EndTime] => -
[Input] => Array
(
[BucketId] => examplebucket-1250000000
[Object] => video01.mp4
[Region] => ap-beijing
)
[JobId] => jfd27519z8zx90c8z90c7809z9f35eb
[Message] =>
[Operation] => Array
(
[Output] => Array
(
[Bucket] => examplebucket-1250000000
[Object] => VideoProcess.flv
[Region] => ap-beijing
)
[TemplateId] => t13466f1eazx08c09zx8c09zx8e9cd
[TemplateName] => video-process-1
[TranscodeTemplateId] => t0b612860z087xc09zx8c09xz8c095dca38
[WatermarkTemplateId] => t185e2e24zxc780z8c0z98c323c6428a19c
[UserData] => xxx
[JobLevel] => 0
)
```

```
[QueueId] => p81e648af2aee496885zx098c09z8xc09zxc086
[StartTime] => -
[State] => Submitted
[Tag] => VideoProcess
)
)
)
```

精彩集锦任务接口

最近更新时间：2023-02-07 15:10:16

简介

本文档提供关于提交精彩集锦任务的 API 概览和 SDK 示例代码。

API	操作描述
提交精彩集锦任务	提交一个精彩集锦任务

提交精彩集锦任务

功能说明

用于提交一个精彩集锦任务。

方法原型

```
public Guzzle\Service\Resource\Model createMediaVideoMontageJobs(array $args = array());
```

请求示例

示例一： 使用模板

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

```
try {
$result = $cosClient->createMediaVideoMontageJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'VideoMontage',
'QueueId' => 'p81e648af2aeexxxxxxxxxxxxxxxxxx',
'Input' => array(
'Object' => 'video01.mp4'
),
'Operation' => array(
'TemplateId' => 't1fcc3770199e04737axxxxxxxxxxxxxx',
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'VideoMontage.mp4',
),
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
),
'CallBack' => '',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二： 自定义参数

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

```php
try {
$result = $cosClient->createMediaVideoMontageJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'VideoMontage',
'QueueId' => 'p81e648af2aeexxxxxxxxxxxxxxxxxx',
'Input' => array(
'Object' => 'video01.mp4'
),
'Operation' => array(
'VideoMontage' => array(
'Duration' => '',
'Container' => array(
'Format' => '',
),
'Video' => array(
'Codec' => '',
'Width' => '',
'Height' => '',
'Fps' => '',
'Bitrate' => '',
'Crf' => '',
),
'Audio' => array(
'Codec' => '',
'Samplerate' => '',
'Bitrate' => '',
'Channels' => '',
'Remove' => '',
),
),
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'VideoMontage.mp4',
),
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
),
'CallBack' => '',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
```

```
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	创建任务的 Tag：VideoMontage	String	是
Input	Request	待操作的媒体信息	Container	是
Operation	Request	操作规则	Container	是
QueueId	Request	任务所在的队列 ID	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型	String	否
CallBack	Request	任务回调地址，优先级高于队列的回调地址。设置为 no 时，表示队列的回调地址不产生回调	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	媒体文件名	String	是

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
VideoMontage	Request.Operation	指定精彩集锦模板参数	Container	否
TemplateId	Request.Operation	指定的模板 ID	String	否
Output	Request.Operation	结果输出地址	Container	是

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
UserData	Request.Operation	透传用户信息, 可打印的 ASCII 码, 长度不超过1024	String	否
JobLevel	Request.Operation	任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0	String	否

注意：

优先使用 TemplateId，无 TemplateId 时使用 VideoMontage。

Container 类型 VideoMontage 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
TimeInterval	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.TimeInterval	Container	否
Container	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.Container	Container	否
Video	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.Video	Container	否
Audio	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.Audio	Container	否
TransConfig	Request.Operation.Transcode	同创建转码模板 CreateMediaTemplate 接口中的 Request.TransConfig	Container	否
AudioMix	Request.Operation.Transcode	混音参数，详情见 AudioMix	Container 数组	否

Container 类型 Output 的具体数据描述如下：

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域	String	是
Bucket	Request.Operation.Output	存储结果的存储桶	String	是
Object	Request.Operation.Output	输出结果的文件名	String	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[Body] => GuzzleHttp\Psr7\Stream Object
(
[stream:GuzzleHttp\Psr7\Stream:private] => Resource id #88
[size:GuzzleHttp\Psr7\Stream:private] =>
[seekable:GuzzleHttp\Psr7\Stream:private] => 1
[readable:GuzzleHttp\Psr7\Stream:private] => 1
[writable:GuzzleHttp\Psr7\Stream:private] => 1
[uri:GuzzleHttp\Psr7\Stream:private] => php://temp
[customMetadata:GuzzleHttp\Psr7\Stream:private] => Array
(
)
)
[RequestId] => NjI2MjIxNDVfHAUOSDHAOIHDOIyMmZkOTQ=
[ContentType] => application/xml
[ContentLength] => 803
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-beijing.myqcloud.com/jobs
[Response] => Array
(
[JobsDetail] => Array
(
[Code] => Success
[CreationTime] => 2022-04-22T11:30:13+0800
[EndTime] => -
[Input] => Array
(
[BucketId] => examplebucket-1250000000
[Object] => video01.mp4
[Region] => ap-beijing
)
[JobId] => j85803a96zx7c98z7xc987z9c6eb232
[Message] =>
[Operation] => Array
(
```

```
[Output] => Array
(
[Bucket] => examplebucket-1250000000
[Object] => VideoMontage.mp4
[Region] => ap-beijing
)
[TemplateId] => t1fcc3770199e0zx7cz897c9zxc9b
[TemplateName] => video-montage-1
[UserData] => xxx
[JobLevel] => 0
)
[QueueId] => p81e648af2aezx7c9z7xc98zx7c98e086
[StartTime] => -
[State] => Submitted
[Tag] => VideoMontage
)
)
)
```

人声分离任务接口

最近更新时间：2023-02-07 15:10:16

简介

本文档提供关于提交人声分离任务的 API 概览和 SDK 示例代码。

API	操作描述
提交人声分离任务	提交一个人声分离任务

提交人声分离任务

功能说明

用于提交一个人声分离任务。

方法原型

```
public Guzzle\Service\Resource\Model createMediaVoiceSeparateJobs(array $args = a
rray());
```

请求示例

示例一： 使用模板

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，http
s://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，h
ttps://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，h
ttps://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

```
try {
$result = $cosClient->createMediaVoiceSeparateJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'VoiceSeparate',
'QueueId' => '',
'CallBack' => '',
'Input' => array(
'Object' => 'test.mp3'
),
'Operation' => array(
'TemplateId' => '',
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'VoiceSeparate01.mp3',
'AuObject' => 'VoiceSeparate02.mp3',
),
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

示例二： 自定义参数

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为 http
'credentials'=> array(
'secretId' => $secretId ,
```

```
'secretKey' => $secretKey)));
try {
$result = $cosClient->createMediaVoiceSeparateJobs(array(
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => 'VoiceSeparate',
'QueueId' => '',
'CallBack' => '',
'Input' => array(
'Object' => 'test.mp3'
),
'Operation' => array(
'Output' => array(
'Region' => $region,
'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Object' => 'VoiceSeparate01.mp3',
'AuObject' => 'VoiceSeparate02.mp3',
),
'VoiceSeparate' => array(
'AudioMode' => 'AudioAndBackground',
'AudioConfig' => array(
'Codec' => 'mp3',
'Samplerate' => '11025',
'Bitrate' => '256',
'Channels' => '2',
),
),
// 'UserData' => 'xxx', // 透传用户信息
// 'JobLevel' => '0', // 任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

节点名称（关键字）	父节点	描述	类型	是否必选
Tag	Request	创建任务的 Tag：VoiceSeparate	String	是
Input	Request	待操作的媒体信息	Container	是
Operation	Request	操作规则	Container	是
QueueId	Request	任务所在的队列 ID	String	是
CallBackFormat	Request	任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式	String	否
CallBackType	Request	任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型	String	否
CallBack	Request	任务回调地址，优先级高于队列的回调地址。设置为 no 时，表示队列的回调地址不产生回调	String	否
CallBackMqConfig	Request	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig	Container	否

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	媒体文件名	String	是

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
VoiceSeparate	Request.Operation	指定转码模板参数	Container	否
TemplateId	Request.Operation	指定的模板 ID	String	否
Output	Request.Operation	结果输出地址	Container	是
JobLevel	Request.Operation	任务优先级，级别限制：0、1、2。级别越大任务优先级越高，默认为0	String	否

注意：

版权所有：腾讯云计算（北京）有限责任公司

优先使用 TemplateId，无 TemplateId 时使用 VoiceSeparate。

Container 类型 VoiceSeparate 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
AudioMode	Request.Operation.VoiceSeparate	同创建人声分离模板 CreateMediaTemplate 接口中的 Request.AudioMode	Container	否
AudioConfig	Request.Operation.VoiceSeparate	同创建人声分离模板 CreateMediaTemplate 接口中的 Request.AudioConfig	Container	否

Container 类型 Output 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Region	Request.Operation.Output	存储桶的地域	String	是
Bucket	Request.Operation.Output	存储结果的存储桶	String	是
Object	Request.Operation.Output	背景音结果文件名，不能与 AuObject 同时为空	String	否
AuObject	Request.Operation.AuObject	人声结果文件名，不能与 Object 同时为空	String	否

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[Body] => GuzzleHttp\Psr7\Stream Object
(
[stream:GuzzleHttp\Psr7\Stream:private] => Resource id #88
[size:GuzzleHttp\Psr7\Stream:private] =>
[seekable:GuzzleHttp\Psr7\Stream:private] => 1
[readable:GuzzleHttp\Psr7\Stream:private] => 1
[writable:GuzzleHttp\Psr7\Stream:private] => 1
[uri:GuzzleHttp\Psr7\Stream:private] => php://temp
[customMetadata:GuzzleHttp\Psr7\Stream:private] => Array
(
```

版权所有：腾讯云计算（北京）有限责任公司

```
)
)
[RequestId] => NjI2MjIyOTHADOHDOADJHOjQ0OV8yMmU0OWM=
[ContentType] => application/xml
[ContentLength] => 857
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-beijing.myqcloud.com/jobs
[Response] => Array
(
[JobsDetail] => Array
(
[Code] => Success
[CreationTime] => 2022-04-22T11:35:47+0800
[EndTime] => -
[Input] => Array
(
[BucketId] => examplebucket-1250000000
[Object] => test.mp3
[Region] => ap-beijing
)
[JobId] => j4c70446zxc780z98xc09zxc6eb232
[Message] =>
[Operation] => Array
(
[Output] => Array
(
[AuObject] => VoiceSeparate02.mp3
[Bucket] => examplebucket-1250000000
[Object] => VoiceSeparate01.mp3
[Region] => ap-beijing
)
[TemplateId] => t1456ea89fczx8c0z8c09z8c0985
[TemplateName] => VoiceSeparate-1
[UserData] => xxx
[JobLevel] => 0
)
[QueueId] => p81e648af2azc709zx8c09z8xc0z7be086
[StartTime] => -
[State] => Submitted
[Tag] => VoiceSeparate
)
)
)
```

模板接口
搜索模板接口

最近更新时间：2022-08-30 17:00:09

简介

本文档提供关于搜索模板接口的 API 概览和 SDK 示例代码。

API	操作描述
搜索动图模板	用于搜索动图模板。
搜索截图模板	用于搜索截图模板。
搜索水印模板	用于搜索水印模板。
搜索转码模板	用于搜索转码模板。
搜索拼接模板	用于搜索拼接模板。
搜索极速高清转码模板	用于搜索极速高清转码模板。
搜索精彩集锦模板	用于搜索精彩集锦模板。
搜索人声分离模板	用于搜索人声分离转码模板。
搜索视频增强模板	用于搜索视频增强模板。
搜索图片处理模板	用于搜索图片处理模板。
搜索超分辨率模板	用于搜索超分辨率模板。

搜索模板

功能说明

用于搜索模板，支持 Animation、Snapshot、Watermark、Transcode、Concat、HighSpeedHd、VideoMontage、VoiceSeparate、VideoProcess、PicProcess、SuperResolution 多种模板类型。

方法原型

```
public Guzzle\Service\Resource\Model describeMediaTemplates(array $args = array
());
```

请求示例

```
<?php
require dirname(__FILE__) . '/../vendor/autoload.php';
$secretId = "SECRETID"; //替换为用户的 secretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$secretKey = "SECRETKEY"; //替换为用户的 secretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
$region = "ap-beijing"; //替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
array(
'region' => $region,
'schema' => 'https', //协议头部，默认为http
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
// 查询模板列表
$result = $cosClient->describeMediaTemplates(array(
'Bucket' => 'examplebucket-125000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.intl.cloud.tencent.com/cos5/bucket
'Tag' => '', // 模板 Tag：Animation、Snapshot、Watermark、Transcode、Concat、HighSpeedHd、VideoMontage、VoiceSeparate、VideoProcess、PicProcess
'Category' => 'Custom',
'Ids' => '',
'Name' => '',
'PageNumber' => '',
'PageSize' => '',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选

节点名称（关键字）	父节点	描述	类型	必选
Tag	无	模板 Tag	String	是
Category	无	Official，Custom，默认值：Custom	String	否
Ids	无	模板 ID，以 `,` 符号分割字符串	String	否
Name	无	模板名称前缀	String	否
PageNumber	无	第几页	Integer	否
PageSize	无	每页个数	Integer	否

返回结果示例

```
GuzzleHttp\Command\Result Object
(
[RequestId] => NjJhOTRlZDFfNzgwYzdkAOIUDOAIUDTA4ZWE=
[ContentType] => application/xml
[ContentLength] => 782
[TotalCount] => 1
[PageNumber] => 1
[PageSize] => 10
[TemplateList] => Array
(
[0] => Array
(
[TemplateId] => t1456ea89fcbdf45basd7fa98sd97a5
[Name] => VoiceSeparate-Name
[State] => Normal
[Tag] => VoiceSeparate
[CreateTime] => 2021-12-27T22:19:15+0800
[UpdateTime] => 2021-12-27T22:19:15+0800
[BucketId] => examplebucket-125000000
[Category] => Custom
[VoiceSeparate] => Array
(
[AudioMode] => AudioAndBackground
[AudioConfig] => Array
(
[Codec] => mp3
[Samplerate] => 44100
[Bitrate] => 128
[Channels] => 2
)
```

版权所有：腾讯云计算（北京）有限责任公司

```
)
)
)
[Bucket] => examplebucket-125000000
[Location] => examplebucket-125000000.ci.ap-guangzhou.myqcloud.com/template
)
```

版权所有：腾讯云计算（北京）有限责任公司

Python SDK 快速入门

最近更新时间：2023-06-14 14:20:44

下载与安装

相关资源

- 对象存储的 XML Python SDK 源码下载地址：XML Python SDK。
- SDK 快速下载地址：XML Python SDK。
- 演示示例 Demo 下载地址：XML Python Demo。
- SDK 文档中的所有示例代码请参见 SDK 代码示例。
- SDK 更新日志请参见 ChangeLog。
- SDK 常见问题请参见：Python SDK 常见问题。

> 说明：
> 如果您在使用 XML 版本 SDK 时遇到函数或方法不存在等错误，请先将 XML 版本 SDK 升级到最新版再重试。

环境依赖

对象存储的 XML Python SDK 目前可以支持 Python 2.7 以及 Python 3.4 及以上。

> 说明：
> 关于文章中出现的 SecretId、SecretKey、Bucket、Region 等名称的含义和获取方式请参见 COS 术语信息。

安装 SDK

安装 SDK 有三种安装方式：pip 安装、手动安装和离线安装。

- 使用 pip 安装（推荐）

```
pip install -U cos-python-sdk-v5
```

- 手动安装
  从 XML Python SDK 下载源码，通过 setup 手动安装，执行以下命令：

```
python setup.py install
```

- 离线安装

```
# 在有外网的机器下运行如下命令
mkdir cos-python-sdk-packages
pip download cos-python-sdk-v5 -d cos-python-sdk-packages
tar -czvf cos-python-sdk-packages.tar.gz cos-python-sdk-packages
# 将安装包拷贝到没有外网的机器后运行如下命令
# 请确保两台机器的 python 版本保持一致，否则会出现安装失败的情况
tar -xzvf cos-python-sdk-packages.tar.gz
pip install cos-python-sdk-v5 --no-index -f cos-python-sdk-packages
```

开始使用

下面为您介绍如何使用 COS Python SDK 完成一个基础操作，例如初始化客户端、创建存储桶、查询存储桶列表、上传对象、查询对象列表、下载对象和删除对象。

初始化

注意：

- 建议用户使用子账号密钥 + 环境变量的方式调用 SDK，提高 SDK 使用的安全性。为子账号授权时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

下面介绍几种初始化 Python SDK Client 的方式，您可以根据具体场景选择其中一种。

通过 COS 默认域名初始化（默认方式）

通过 COS 默认域名访问时，SDK 会以 **{bucket-appid}.cos.{region}.myqcloud.com** 的域名形式访问 COS。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
```

```
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性, 包括 secret_id, secret_key, region等。Appid 已在CosConfig中移除, 请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId, 建议使用子账号密钥, 授权遵循最小权限指引, 降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey, 建议使用子账号密钥, 授权遵循最小权限指引, 降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region, 已创建桶归属的region可以在控制台查看, https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token, 如果使用临时密钥需要填入, 临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS, 默认为 https, 可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)
```

注意：
正常情况下一个 region 只需要生成一个 CosS3Client 实例，然后循环上传或下载对象，不能每次访问都生成 CosS3Client 实例，否则 python 进程会占用过多的连接和线程。

说明：
关于临时密钥如何生成和使用，请参见 临时密钥生成及使用指引。

通过 COS 默认域名初始化（临时密钥方式）

通过 COS 默认域名访问时，SDK 会以 **{bucket-appid}.cos.{region}.myqcloud.com** 的域名形式访问 COS。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO, 需要定位时可以修改为DEBUG, 此时SDK会打印和服务端的通信信息
```

```
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id, secret_key, region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
tmp_secret_id = 'TmpSecretId' # 临时密钥的 SecretId，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
tmp_secret_key = 'TmpSecretKey' # 临时密钥的 SecretKey，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
token = 'TmpToken' # 临时密钥的 Token，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=tmp_secret_id, SecretKey=tmp_secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)
```

通过 COS 默认域名和代理初始化

当需要通过代理访问 COS 时配置，否则跳过。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id, secret_key, region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
```

```
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
proxies = {
'http': '127.0.0.1:80', # 替换为用户的 HTTP代理地址
'https': '127.0.0.1:443' # 替换为用户的 HTTPS代理地址
}

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Proxies=proxies)
client = CosS3Client(config)
```

通过 COS 加速域名初始化

通过 COS 全球加速域名访问时，SDK 会以 **{bucket-appid}.cos.accelerate.myqcloud.com** 的域名形式访问 COS，region 不会出现在访问域名中。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = None # 通过Endpoint初始化不需要配置region
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

endpoint = 'cos.accelerate.myqcloud.com' # 替换为用户的 endpoint 或者 cos全局加速域名，如果使用桶的全球加速域名，需要先开启桶的全球加速功能，请参见https://intl.cloud.tencent.com/document/product/436/33406
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Endpoint=endpoint, Scheme=scheme)
client = CosS3Client(config)
```

通过自定义域名初始化

通过自定义域名访问时，SDK 会使用配置的**用户域名**直接访问 COS，bucket 和 region 都不会出现在访问域名中。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = None # 通过自定义域名初始化不需要配置region
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

domain = 'user-define.example.com' # 用户自定义域名，需要先开启桶的自定义域名，具体请参见https://intl.cloud.tencent.com/document/product/436/31507
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Domain=domain, Scheme=scheme)
client = CosS3Client(config)
```

通过 CDN 默认域名初始化

通过 CDN 默认域名访问时，SDK 会以 **{bucket-appid}.file.myqcloud.com** 的域名形式访问 **CDN**，由 CDN 服务回源访问 COS。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
```

```
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id, secret_key, region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = None # 通过Endpoint初始化不需要配置region
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

endpoint = 'file.mycloud.com' # 替换为用户的 CDN 默认加速域名，需要开通 CDN 加速配置，参见https://intl.cloud.tencent.com/document/product/436/18670
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Endpoint=endpoint, Scheme=scheme)
client = CosS3Client(config)
```

通过 CDN 自定义域名初始化

通过 CDN 自定义域名访问时，SDK 会使用配置的**用户域名**直接访问 **CDN**，bucket 和 region 都不会出现在访问域名中，由 CDN 服务回源访问 COS。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id, secret_key, region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = None # 通过自定义域名初始化不需要配置region
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用
```

```
指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

domain = 'user-define.example-cdn.com' # 用户自定义 CDN 域名，需要开通 CDN 自定义域名加速，参见https://intl.cloud.tencent.com/document/product/436/18670
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Domain=domain, Scheme=scheme)
client = CosS3Client(config)
```

创建存储桶

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.create_bucket(
Bucket='examplebucket-1250000000'
)
```

查询存储桶列表

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.list_buckets(
)
```

上传对象

> 注意：
> 简单上传不支持超过5G的文件，推荐使用下方高级上传接口。参数说明可参见 对象操作 文档。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
```

```
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

#### 文件流简单上传（不支持超过5G的文件，推荐使用下方高级上传接口）
# 强烈建议您以二进制模式(binary mode)打开文件,否则可能会导致错误
with open('picture.jpg', 'rb') as fp:
response = client.put_object(
Bucket='examplebucket-1250000000',
Body=fp,
Key='picture.jpg',
StorageClass='STANDARD',
EnableMD5=False
)
print(response['ETag'])

#### 字节流简单上传
response = client.put_object(
Bucket='examplebucket-1250000000',
Body=b'bytes',
Key='picture.jpg',
EnableMD5=False
)
```

```
print(response['ETag'])


#### chunk 简单上传
import requests
stream = requests.get('https://intl.cloud.tencent.com/document/product/436/7778')

# 网络流将以 Transfer-Encoding:chunked 的方式传输到 COS
response = client.put_object(
Bucket='examplebucket-1250000000',
Body=stream,
Key='picture.jpg'
)
print(response['ETag'])

#### 高级上传接口（推荐）
# 根据文件大小自动选择简单上传或分块上传，分块上传具备断点续传功能。
response = client.upload_file(
Bucket='examplebucket-1250000000',
LocalFilePath='local.txt',
Key='picture.jpg',
PartSize=1,
MAXThread=10,
EnableMD5=False
)
print(response['ETag'])
```

查询对象列表

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/docum
```

```
ent/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.list_objects(
Bucket='examplebucket-1250000000',
Prefix='folder1'
)
```

单次调用 `list_objects` 接口一次只能查询1000个对象，如需要查询所有的对象，则需要循环调用。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token
```

```
=token, Scheme=scheme)
client = CosS3Client(config)

marker = ""
while True:
response = client.list_objects(
Bucket='examplebucket-1250000000',
Prefix='folder1',
Marker=marker
)
if 'Contents' in response:
print(response['Contents'])
if response['IsTruncated'] == 'false':
break
marker = response['NextMarker']
```

下载对象

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
```

```
client = CosS3Client(config)

#### 获取文件到本地
response = client.get_object(
Bucket='examplebucket-1250000000',
Key='picture.jpg',
)
response['Body'].get_stream_to_file('output.txt')

#### 获取文件流
response = client.get_object(
Bucket='examplebucket-1250000000',
Key='picture.jpg',
)
fp = response['Body'].get_raw_stream()
print(fp.read(2))

#### 设置 Response HTTP 头部
response = client.get_object(
Bucket='examplebucket-1250000000',
Key='picture.jpg',
ResponseContentType='text/html; charset=utf-8'
)
print(response['Content-Type'])
fp = response['Body'].get_raw_stream()
print(fp.read(2))

#### 指定下载范围
response = client.get_object(
Bucket='examplebucket-1250000000',
Key='picture.jpg',
Range='bytes=0-10'
)
fp = response['Body'].get_raw_stream()
print(fp.read())
```

删除对象

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
```

```
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id, secret_key, region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 删除object
## deleteObject
response = client.delete_object(
Bucket='examplebucket-1250000000',
Key='exampleobject'
)

# 删除多个 object
## deleteObjects
response = client.delete_objects(
Bucket='examplebucket-1250000000',
Delete={
'Object': [
{
'Key': 'exampleobject1',
},
{
'Key': 'exampleobject2',
},
],
'Quiet': 'true'|'false'
}
)
```

Python SDK 常见问题

最近更新时间：2023-01-30 16:12:04

COS Python SDK 升级后，无法执行“移动文件”操作，该如何处理？

您可以采用 PUT Object - Copy 和 DELETE Object 接口实现。建议您在执行删除操作前校验数据的一致性。详细介绍请参见 MD5 校验 文档。

COS Python SDK 如何获取下载文件的临时链接？

Python SDK 提供获取签名、获取请求预签名 URL 接口以及获取对象下载预签名 URL 接口。使用永久密钥或临时密钥获取预签名 URL 的调用方法相同，使用临时密钥时，需要在 header 或 query_string 中加上 x-cos-security-token。详情请参见 生成预签名 URL 文档。

使用 COS Python SDK 时出现异常，该如何处理？

COS XML Python SDK 操作成功会返回一个 dict 或者 None。若调用 SDK 接口请求 COS 服务失败，系统将抛出 CosClientError（客户端异常）或者 CosServiceError（服务端异常）。

- 服务端返回的错误：指服务端处理一些不符合要求的客户端请求所返回的错误。例如访问不存在的文件，没有文件的访问权限等。更多服务端返回的错误码详细信息，请参见 API 错误码。
- 客户端错误：主要指网络异常、文件读写 IO 异常、参数校验失败等。

版权所有：腾讯云计算（北京）有限责任公司

存储桶操作

最近更新时间：2023-01-30 11:50:32

简介

本文档提供关于存储桶基本操作的相关 API 概览以及 SDK 示例代码。

注意：

- 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

API	操作名	操作描述
GET Service	查询存储桶列表	查询指定账号下所有的存储桶列表
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

查询存储桶列表

功能说明

用于查询指定账号下所有存储桶列表（GET Service）。

方法原型

```
list_buckets()
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
```

```
import sys
import logging
# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = 'SecretId' # 替换为用户的 SecretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
secret_key = 'SecretKey' # 替换为用户的 SecretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)
response = client.list_buckets(
)
```

返回结果说明

查询存储桶列表，类型为 dict。

```
{
'Buckets': {
'Bucket': [
{
'Name': 'string',
'Location': 'string',
'CreationDate': 'string'
},
],
},
'Owner': {
'DisplayName': 'string',
'ID': 'string'
}
}
```

参数名称	参数描述	类型
Buckets	存储桶列表	Dict

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
Bucket	存储桶列表	List
Name	存储桶名称	String
Location	存储桶所在的地域名	String
CreationDate	存储桶创建的时间	String
Owner	存储桶所有者信息	Dict
DisplayName	存储桶所有者的名字信息	String
ID	存储桶所有者 ID	String

创建存储桶

功能说明

在指定账号下创建一个存储桶（PUT Bucket）。

方法原型

```
create_bucket(Bucket, BucketAZConfig=None, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging
# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = 'SecretId' # 替换为用户的 SecretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
secret_key = 'SecretKey' # 替换为用户的 SecretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用
```

```
指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token
=token, Scheme=scheme)
client = CosS3Client(config)
# 存储桶名称不支持大写字母，COS 后端会将用户传入的大写字母自动转换为小写字母用于创建存储桶
response = client.create_bucket(
Bucket='examplebucket-1250000000'
)
```

全部参数请求示例

```
response = client.create_bucket(
Bucket='examplebucket-1250000000',
BucketAZConfig='string',
ACL='private'|'public-read'|'public-read-write',
GrantFullControl='string',
GrantRead='string',
GrantWrite='string'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	待创建的存储桶名称，由 BucketName-APPID 构成. 存储桶名称不支持大写字母，COS 后端会将用户传入的大写字母自动转换为小写字母用于创建存储桶	String	是
BucketAZConfig	存储桶 AZ 配置，指定为 'MAZ' 以创建多 AZ 存储桶	String	否
ACL	设置存储桶的 ACL，例如 'private'，'public-read'，'public-read-write'	String	否
GrantFullControl	赋予指定账户对存储桶的读写权限，格式为 `id="OwnerUin"`	String	否
GrantRead	赋予指定账户对存储桶的读权限，格式为 `id="OwnerUin"`	String	否
GrantWrite	赋予指定账户对存储桶的写权限，格式为 `id="OwnerUin"`	String	否

返回结果说明

该方法返回值为 None。

版权所有：腾讯云计算（北京）有限责任公司

检索存储桶及其权限

功能说明

检索存储桶是否存在且是否有权限访问（HEAD Bucket）。

方法原型

```
head_bucket(Bucket)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging
# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = 'SecretId' # 替换为用户的 SecretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
secret_key = 'SecretKey' # 替换为用户的 SecretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)
response = client.head_bucket(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	待查询的存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

版权所有：腾讯云计算（北京）有限责任公司

该方法返回值为 None。

删除存储桶

功能说明

删除指定账号下的空存储桶（DELETE Bucket）。

方法原型

```
delete_bucket(Bucket)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging
# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = 'SecretId' # 替换为用户的 SecretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
secret_key = 'SecretKey' # 替换为用户的 SecretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)
response = client.delete_bucket(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名称	参数描述	类型	是否必填

参数名称	参数描述	类型	是否必填
Bucket	待删除的存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

该方法返回值为 None。

判断存储桶是否存在

功能说明

判断存储桶是否存在。

方法原型

```
bucket_exists(Bucket)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging
# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = 'SecretId' # 替换为用户的 SecretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
secret_key = 'SecretKey' # 替换为用户的 SecretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)
response = client.bucket_exists(
Bucket='examplebucket-1250000000'
```

```
)
print(response)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

True 表示存储桶存在，False 表示存储桶不存在。

对象操作
上传对象

最近更新时间：2023-06-14 15:05:28

简介

本文档提供关于对象的高级上传、简单上传、分块上传等操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object	简单上传对象	上传一个对象至存储桶
APPEND Object	追加上传对象	将对象以分块追加的方式上传至存储桶

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传任务
Upload Part	上传分块	分块上传文件
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

高级接口（推荐）

上传对象（断点续传）

功能说明

该高级接口根据用户文件的长度自动选择简单上传以及分块上传，对于小于等于20M的文件调用简单上传，对于大于20MB的文件调用分块上传，对于分块上传未完成的文件会自动进行断点续传。
对于分块上传的文件，可以通过 progress_callback 回调函数，获取上传进度。

方法原型

```
upload_file(Bucket, Key, LocalFilePath, PartSize=1, MAXThread=5, EnableMD5=False,
progress_callback=None, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos.cos_exception import CosClientError, CosServiceError
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 使用高级接口上传一次，不重试，此时没有使用断点续传的功能
response = client.upload_file(
Bucket='examplebucket-1250000000',
```

```
Key='exampleobject',
LocalFilePath='local.txt',
EnableMD5=False,
progress_callback=None
)

# 使用高级接口断点续传，失败重试时不会上传已成功的分块(这里重试10次)
for i in range(0, 10):
try:
response = client.upload_file(
Bucket='examplebucket-1250000000',
Key='exampleobject',
LocalFilePath='local.txt')
break
except CosClientError or CosServiceError as e:
print(e)
```

全部参数请求示例

```
def upload_percentage(consumed_bytes, total_bytes):
"""进度条回调函数，计算当前上传的百分比

:param consumed_bytes: 已经上传的数据量
:param total_bytes: 总数据量
"""
if total_bytes:
rate = int(100 * (float(consumed_bytes) / float(total_bytes)))
print('\r{0}% '.format(rate))
sys.stdout.flush()

response = client.upload_file(
Bucket='examplebucket-1250000000',
Key='exampleobject',
LocalFilePath='local.txt',
PartSize=1,
MAXThread=5,
progress_callback=upload_percentage,
EnableMD5=False|True,
ACL='private'|'public-read', # 请慎用此参数,否则会达到1000条ACL 上限
GrantFullControl='string',
GrantRead='string',
StorageClass='STANDARD'|'STANDARD_IA'|'ARCHIVE',
Expires='string',
CacheControl='string',
ContentType='string',
ContentDisposition='string',
```

版权所有：腾讯云计算（北京）有限责任公司

```
ContentEncoding='string',
ContentLanguage='string',
ContentLength='123',
ContentMD5='string',
Metadata={
'x-cos-meta-key1': 'value1',
'x-cos-meta-key2': 'value2'
},
TrafficLimit='1048576'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
LocalFilePath	本地文件的路径名	String	是
PartSize	分块上传的分块大小，默认为1MB	Int	否
MAXThread	分块上传的并发数量，默认为5个线程上传分块	Int	否
progress_callback	上传进度的回调函数，可以通过自定义此函数，来获取上传进度	Func	否
EnableMD5	是否需要 SDK 计算 Content-MD5，默认关闭，打开后会增加上传耗时	Bool	否
ACL	设置对象的 ACL，例如 private，public-read	String	否
GrantFullControl	赋予被授权者所有的权限，格式为 `id="OwnerUin"`，例如 `id="100000000001"`	String	否
GrantRead	赋予被授权者读的权限，格式为 `id="OwnerUin"`，例如 `id="100000000001"`	String	否

| StorageClass | 设置对象的存储类型，例如 STANDARD，STANDARD_IA，ARCHIVE。默认值为 STANDARD，更多存储类型，请参见 存储类型概述 | String | 否 |

| Expires | 设置 Expires | String| 否 |

| CacheControl | 缓存策略，设置 Cache-Control | String | 否 |

ContentType	内容类型，设置 Content-Type	String	否
ContentDisposition	文件名称，设置 Content-Disposition	String	否
ContentEncoding	编码格式，设置 Content-Encoding	String	否
ContentLanguage	语言类型，设置 Content-Language	String	否
ContentLength	设置传输长度	String	否
ContentMD5	设置上传对象的 MD5 值用于校验	String	否
Metadata	用户自定义的对象元数据	Dict	否
TrafficLimit	单链接限速的值，单位为bit/s，限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，高级接口限制的是单线程的速度	String	否

返回结果说明

上传对象的属性，类型为 dict：

```
{
'ETag': 'string'
'x-cos-hash-crc64ecma': 'string',
'x-cos-request-id': 'string',
'x-cos-storage-class': 'string',
'Location': 'string', # 触发分块上传时返回此字段
'Bucket': 'string', # 触发分块上传时返回此字段
'Key': 'string', # 触发分块上传时返回此字段
}
```

参数名称	参数描述	类型
ETag	分块上传时，该值不是对象内容的 MD5 校验值，仅能用于检查对象唯一性	String
x-cos-hash-crc64ecma	文件的 CRC64 校验码	String
x-cos-request-id	请求 ID	String
x-cos-storage-class	对象存储类型	String
Location	对象的访问 URL (触发分块上传时返回此字段)	String
Bucket	存储桶名 (触发分块上传时返回此字段)	String
Key	对象名 (触发分块上传时返回此字段)	String

批量上传（本地文件夹上传）

功能说明

该示例展示通过组合 SDK 的基本接口，完成批量上传本地文件夹到 COS。

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos import CosServiceError
from qcloud_cos import CosClientError
from qcloud_cos.cos_threadpool import SimpleThreadPool

import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig中 移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token) # 获取配置对象
client = CosS3Client(config)

uploadDir = '/root/logs'
bucket = 'examplebucket-125000000'
g = os.walk(uploadDir)
# 创建上传的线程池
pool = SimpleThreadPool()
for path, dir_list, file_list in g:
for file_name in file_list:
srcKey = os.path.join(path, file_name)
cosObjectKey = srcKey.strip('/')
# 判断 COS 上文件是否存在
```

```
exists = False
try:
response = client.head_object(Bucket=bucket, Key=cosObjectKey)
exists = True
except CosServiceError as e:
if e.get_status_code() == 404:
exists = False
else:
print("Error happened, reupload it.")
if not exists:
print("File %s not exists in cos, upload it", srcKey)
pool.add_task(client.upload_file, bucket, cosObjectKey, srcKey)


pool.wait_completion()
result = pool.get_result()
if not result['success_all']:
print("Not all files upload sucessed. you should retry")
```

简单操作

简单上传对象

功能说明

上传一个对象至存储桶（PUT Object）。

方法原型

```
put_object(Bucket, Body, Key, **kwargs)
```

请求示例1：简单文件上传

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id, secret_key, region等。Appid 已在 CosConfig 中移除，
```

```
请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 文件流 简单上传
file_name = 'test.txt'
with open('test.txt', 'rb') as fp:
response = client.put_object(
Bucket='examplebucket-1250000000', # Bucket 由 BucketName-APPID 组成
Body=fp,
Key=file_name,
StorageClass='STANDARD',
ContentType='text/html; charset=utf-8'
)
print(response['ETag'])

# 字节流 简单上传
response = client.put_object(
Bucket='examplebucket-1250000000',
Body=b'abcdefg',
Key=file_name
)
print(response['ETag'])

# 本地路径 简单上传
response = client.put_object_from_local_file(
Bucket='examplebucket-1250000000',
LocalFilePath='local.txt',
Key=file_name,
)
print(response['ETag'])
```

版权所有：腾讯云计算（北京）有限责任公司

```
# 设置 HTTP 头部 简单上传
response = client.put_object(
Bucket='examplebucket-1250000000',
Body=b'test',
Key=file_name,
ContentType='text/html; charset=utf-8'
)
print(response['ETag'])

# 设置自定义头部 简单上传
response = client.put_object(
Bucket='examplebucket-1250000000',
Body=b'test',
Key=file_name,
Metadata={
'x-cos-meta-key1': 'value1',
'x-cos-meta-key2': 'value2'
}
)
print(response['ETag'])

# 上传时限速
with open('test.bin', 'rb') as fp:
response = client.put_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Body=fp,
TrafficLimit='819200'
)
print(response['ETag'])
```

请求示例2：创建目录

COS 中的目录是'/'结尾的特殊对象名，调用 Put Object 接口即可。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移
```

```
除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 创建目录
dir_to_create='path/to/create/dir/'
response = client.put_object(
Bucket='examplebucket-1250000000', # Bucket 由 BucketName-APPID 组成
Key=dir_to_create,
Body=b'',
)
print(response)
```

请求示例3：上传对象到指定的目录

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权
```

```
# 遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

#上传由 '/' 分隔的对象名，自动创建包含文件的文件夹。想要在此文件夹中添加新文件时，只需要在上传文件至 COS 时，将 Key 填写为此目录前缀即可。
dir_name = 'path/to/dir/'
file_name = 'test.txt'
object_key = dir_name + file_name
with open('test.txt', 'rb') as fp:
response = client.put_object(
Bucket='examplebucket-1250000000', # Bucket 由 BucketName-APPID 组成
Body=fp,
Key=object_key,
StorageClass='STANDARD',
ContentType='text/html; charset=utf-8'
)
print(response['ETag'])
```

全部参数请求示例

```
response = client.put_object(
Bucket='examplebucket-1250000000',
Body=b'bytes'|file,
Key='exampleobject',
EnableMD5=False|True,
ACL='private'|'public-read', # 请慎用此参数,否则会达到1000条 ACL 上限
GrantFullControl='string',
GrantRead='string',
StorageClass='STANDARD'|'STANDARD_IA'|'ARCHIVE',
Expires='string',
CacheControl='string',
ContentType='string',
ContentDisposition='string',
ContentEncoding='string',
ContentLanguage='string',
```

版权所有：腾讯云计算（北京）有限责任公司

```
ContentLength='123',
ContentMD5='string',
Metadata={
'x-cos-meta-key1': 'value1',
'x-cos-meta-key2': 'value2'
},
TrafficLimit='1048576'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Body	上传对象的内容，可以为文件流或字节流	file/bytes	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
EnableMD5	是否需要 SDK 计算 Content-MD5，默认关闭，打开后将增加上传耗时	Bool	否
ACL	设置对象的 ACL，例如 'private'，'public-read'	String	否
GrantFullControl	赋予被授权者所有的权限，格式为 `id="OwnerUin"`，例如 `id="100000000001"`	String	否
GrantRead	赋予被授权者读的权限，格式为 `id="OwnerUin"`，例如 `id="100000000001"`	String	否

StorageClass	设置对象的存储类型，STANDARD，STANDARD_IA，ARCHIVE。默认值为 STANDARD。更多存储类型，请参见 存储类型概述	String	否
Expires	设置 Expires	String	否
CacheControl	缓存策略，设置 Cache-Control	String	否
ContentType	内容类型，设置 Content-Type	String	否
ContentDisposition	对象名称，设置 Content-Disposition	String	否
ContentEncoding	编码格式，设置 Content-Encoding	String	否
ContentLanguage	语言类型，设置 Content-Language	String	否
ContentLength	设置传输长度	String	否
ContentMD5	设置上传对象的 MD5 值用于校验	String	否

| Metadata | 用户自定义的对象元数据， 必须以 x-cos-meta 开头，否则会被忽略 | Dict | 否 |
| TrafficLimit | 单链接限速的值，单位为bit/s，限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s | String | 否 |

返回结果说明

上传对象的属性，类型为 dict：

```
{
'ETag': 'string',
'x-cos-version-id': 'string'
}
```

参数名称	参数描述	类型
ETag	上传对象的 MD5 值	String
x-cos-version-id	开启版本控制后，对象的版本号	String

追加上传对象

功能说明

将对象以分块追加的方式上传至存储桶（APPEND Object）。

方法原型

```
append_object(Bucket, Key, Position, Data, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，
请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循
最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/
```

```
product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.append_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Position=0,
Data=b'b'*1024*1024
)
```

全部参数请求示例

```
response = client.append_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Position=0,
Data=b'bytes'|file
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
Position	追加操作的起始点，单位为字节。首次追加则设置Position=0，后续追加则设置Position为当前Object的content-length	Int	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
Data	上传分块的内容，可以为本地文件流或输入流	file/bytes	是

返回结果说明

追加后对象的属性，包括下次追加的位置，类型为 dict。

```
{
'ETag': '"9a4802d5c99dafe1c04da0a8e7e166bf"',
'x-cos-next-append-position': '12',
'x-cos-request-id': 'NjEwN2Q0ZGZfMWNhZjU4NjRfMzM1M19hNzQzYjc2'
}
```

分块操作

关于分块上传的更多说明请参见 分块上传。分块上传对象可包括的操作如下：

- 分块上传对象：初始化分块上传，上传分块，完成所有分块上传。
- 分块续传：查询已上传的分块，上传分块，完成所有分块上传。
- 删除已上传分块。

查询分块上传

功能说明

查询指定存储桶正在进行中的分块上传信息（List Multipart Uploads）。

方法原型

```
list_multipart_uploads(Bucket, Prefix="", Delimiter="", KeyMarker="", UploadIdMar
ker="", MaxUploads=1000, EncodingType="", **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging
```

```
# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.list_multipart_uploads(
Bucket='examplebucket-1250000000',
Prefix='dir'
)
```

全部参数请求示例

```
response = client.list_multipart_uploads(
Bucket='examplebucket-1250000000',
Prefix='string',
Delimiter='string',
KeyMarker='string',
UploadIdMarker='string',
MaxUploads=100,
EncodingType='url'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
Prefix	默认为空，对分块上传的 key 进行筛选，匹配 prefix 为前缀的分块上传	String	否
Delimiter	默认为空，设置分隔符	String	否
KeyMarker	和 UploadIdMarker 一起使用，指明列出分块上传的起始位置	String	否
UploadIdMarker	和 KeyMarker 一起使用，指明列出分块上传的起始位置。如果未指定 KeyMarker，UploadIdMarker 将被忽略	String	否
MaxUploads	最多返回的分块上传的数量，默认为最大的1000	Int	否
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String	否

返回结果说明

获取分块上传的信息，类型为 dict：

```
{
'Bucket': 'examplebucket-1250000000',
'Prefix': 'string',
'Delimiter': 'string',
'KeyMarker': 'string',
'UploadIdMarker': 'string',
'NextKeyMarker': 'string',
'NextUploadIdMarker': 'string',
'MaxUploads': '1000',
'IsTruncated': 'true'|'false',,
'EncodingType': 'url',
'Upload':[
{
'UploadId': 'string',
'Key': 'string',
'Initiated': 'string',
'StorageClass': 'STANDARD',
'Owner': {
'DisplayName': 'string',
'ID': 'string'
},
'Initiator': {
'ID': 'string',
'DisplayName': 'string'
}
},
],
```

```
'CommonPrefixes':[
{
'Prefix': 'string'
},
],
}
```

参数名称	参数描述	类型
Bucket	存储桶名称，由 BucketName-APPID 构成	String
Prefix	默认为空，对分块上传的 key 进行筛选，匹配 prefix 为前缀的分块上传	String
Delimiter	默认为空，设置分隔符	String
KeyMarker	和 UploadIdMarker 一起使用，指明列出分块上传的 key 起始位置	String
UploadIdMarker	和 KeyMarker 一起使用，指明列出分块上传的 uploadId 起始位置。如果未指定 KeyMarker，UploadIdMarker 将被忽略	String
NextKeyMarker	当 IsTruncated 为 true 时，指明下一次列出分块上传的 key 的起始位置	String
NextUploadIdMarker	当 IsTruncated 为 true 时，指明下一次列出分块上传的 uploadId 的起始位置	String
MaxUploads	最多返回的分块上传的数量，默认为最大的1000	Int
IsTruncated	表示返回的分块上传是否被截断	String
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String
Upload	包含所有分块上传的 list，包括 'UploadId'，'StorageClass'，'Key'，'Owner'，'Initiator'，'Initiated' 等信息	List
CommonPrefixes	所有以 Prefix 开头，以 Delimiter 结尾的 Key 被归到同一类	List

初始化分块上传

功能说明

初始化分块上传，获取对应的 uploadId（Initiate Multipart Upload）。

方法原型

```
create_multipart_upload(Bucket, Key, **kwargs):
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.create_multipart_upload(
Bucket='examplebucket-1250000000',
Key='exampleobject',
StorageClass='STANDARD'
)
```

全部参数请求示例

```
response = client.create_multipart_upload(
Bucket='examplebucket-1250000000',
Key='multipart.txt',
StorageClass='STANDARD'|'STANDARD_IA'|'ARCHIVE',
Expires='string',
CacheControl='string',
ContentType='string',
```

版权所有：腾讯云计算（北京）有限责任公司

```
ContentDisposition='string',
ContentEncoding='string',
ContentLanguage='string',
Metadata={
'x-cos-meta-key1': 'value1',
'x-cos-meta-key2': 'value2'
},
ACL='private'|'public-read',
GrantFullControl='string',
GrantRead='string'
)
# 获取 UploadId 供后续接口使用
uploadid = response['UploadId']
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	Bucket 名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
StorageClass	设置对象的存储类型，STANDARD，STANDARD_IA，ARCHIVE。默认值为 STANDARD。更多存储类型，请参见 存储类型概述	String	否
Expires	设置 Expires	String	否
CacheControl	缓存策略，设置 Cache-Control	String	否
ContentType	内容类型，设置 Content-Type	String	否
ContentDisposition	文件名称，设置 Content-Disposition	String	否
ContentEncoding	编码格式，设置 Content-Encoding	String	否
ContentLanguage	语言类型，设置 Content-Language	String	否
Metadata	用户自定义的对象元数据	Dict	否
ACL	设置对象的 ACL，例如 'private'，'public-read'	String	否
GrantFullControl	赋予被授权者所有的权限，格式为 `id="OwnerUin"`，例如 `id="100000000001"`	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
GrantRead	赋予被授权者读的权限，格式为 `id="OwnerUin"`，例如 `id="100000000001"`	String	否

返回结果说明

获取分块上传的初始化信息，类型为 dict：

```
{
'UploadId': '150219101333cecfd6718d0caea1e2738401f93aa531a4be7a2afee0f8828416f3278e5570',
'Bucket': 'examplebucket-1250000000',
'Key': 'exampleobject'
}
```

参数名称	参数描述	类型
UploadId	标识分块上传的 ID	String
Bucket	存储桶名称，由 BucketName-APPID 组成	String
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String

上传分块

分块上传对象（Upload Part）。

方法原型

```
upload_part(Bucket, Key, Body, PartNumber, UploadId, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging
```

```
# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 注意，上传分块的块数最多10000块
response = client.upload_part(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Body=b'b'*1024*1024,
PartNumber=1,
UploadId='exampleUploadId'
)
```

全部参数请求示例

```
# 注意，上传分块的块数最多10000块
response = client.upload_part(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Body=b'bytes'|file,
PartNumber=1,
UploadId='string',
EnableMD5=False|True,
ContentLength='123',
ContentMD5='string',
TrafficLimit='1048576'
)
```

版权所有：腾讯云计算（北京）有限责任公司

参数说明

参数名称	参数描述	类型	是否必填
Bucket	Bucket 名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
Body	上传分块的内容，可以为本地文件流或输入流	file/bytes	是
PartNumber	标识上传分块的序号	Int	是
UploadId	标识分块上传的 ID	String	是
EnableMD5	是否需要 SDK 计算 Content-MD5，默认关闭，打开后会增加上传耗时	Bool	否
ContentLength	设置传输长度	String	否
ContentMD5	设置上传对象的 MD5 值用于校验	String	否
TrafficLimit	单链接限速的值，单位为bit/s，限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s	String	否

返回结果说明

上传分块的属性，类型为 dict：

```
{
'ETag': 'string'
}
```

参数名称	参数描述	类型
ETag	上传分块的 MD5 值。	String

复制分块

将其他对象复制为一个分块（Upload Part - Copy）。

方法原型

```
upload_part_copy(Bucket, Key, PartNumber, UploadId, CopySource, CopySourceRange=
'', **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，
请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循
最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/
product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权
遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/docum
ent/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，htt
ps://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/43
6/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用
指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token
=token, Scheme=scheme)
client = CosS3Client(config)

response = client.upload_part_copy(
Bucket='examplebucket-1250000000',
Key='exampleobject',
PartNumber=1,
UploadId='exampleUploadId',
CopySource={
'Bucket': 'sourcebucket-1250000000',
'Key': 'exampleobject',
'Region': 'ap-guangzhou'
```

```
}
)
```

全部参数请求示例

```
response = client.upload_part_copy(
Bucket='examplebucket-1250000000',
Key='exampleobject',
PartNumber=100,
UploadId='string',
CopySource={
'Bucket': 'sourcebucket-1250000000',
'Key': 'sourceObject',
'Region': 'COS_REGION', #替换为源存储桶的 Region
'VersionId': 'string'
},
CopySourceRange='string',
CopySourceIfMatch='string',
CopySourceIfModifiedSince='string',
CopySourceIfNoneMatch='string',
CopySourceIfUnmodifiedSince='string'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
PartNumber	标识上传分块的序号	Int	是
UploadId	标识分块上传的 ID	String	是
CopySource	描述拷贝源对象的路径，包含 Bucket、Key、Region、VersionId	Dict	是
CopySourceRange	描述拷贝源对象的范围，格式为 bytes=first-last。不指定时，默认拷贝整个源对象	String	否
CopySourceIfMatch	源对象的 Etag 与给定的值相同时才拷贝	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
CopySourceIfModifiedSince	源对象在给定的时间后被修改相才拷贝	String	否
CopySourceIfNoneMatch	源对象的 Etag 与给定的值不相同时才拷贝	String	否
CopySourceIfUnmodifiedSince	源对象在给定的时间后没有修改相才拷贝	String	否

返回结果说明

复制分块的属性，类型为 dict：

```
{
'ETag': 'string',
'LastModified': 'string',
'x-cos-copy-source-version-id': 'string',
}
```

参数名称	参数描述	类型
ETag	拷贝分块的 MD5 值	String
LastModified	拷贝分块的最后一次修改时间	String
x-cos-copy-source-version-id	源对象的版本号	String

查询已上传块

功能说明

查询特定分块上传操作中的已上传的块（List Parts）。

方法原型

```
list_parts(Bucket, Key, UploadId, MaxParts=1000, PartNumberMarker=0, EncodingType
='', **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging
```

```
# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.list_parts(
Bucket='examplebucket-1250000000',
Key='exampleobject',
UploadId='exampleUploadId'
)
```

全部参数请求示例

```
response = client.list_parts(
Bucket='examplebucket-1250000000',
Key='exampleobject',
UploadId=uploadid,
MaxParts=1000,
PartNumberMarker=100,
EncodingType='url'
)
```

参数说明

参数名称	参数描述	类型	是否必填

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
UploadId	标识分块上传的 ID	String	是
MaxParts	最多返回的分块的数量，默认为最大的1000	Int	否
PartNumberMarker	默认为0，从第一块列出分块，从 PartNumberMarker 下一个分块开始列出	Int	否
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String	否

返回结果说明

所有上传分块的信息，类型为 dict：

```
{
'Bucket': 'examplebucket-1250000000',
'Key': 'exampleobject',
'UploadId': '1502192444bdb382add546a35b2eeab81e06ed84086ca0bb75ea45ca7fa073fa9cf74ec4f2',
'EncodingType': None,
'MaxParts': '1000',
'IsTruncated': 'true',
'PartNumberMarker': '0',
'NextPartNumberMarker': '1000',
'StorageClass': 'Standard',
'Part': [
{
'LastModified': '2017-08-08T11:40:48.000Z',
'PartNumber': '1',
'ETag': '"8b8378787c0925f42ccb829f6cc2fb97"',
'Size': '10485760'
},
],
'Initiator': {
'DisplayName': '3333333333',
'ID': 'qcs::cam::uin/3333333333:uin/3333333333'
},
'Owner': {
```

版权所有：腾讯云计算（北京）有限责任公司

```
'DisplayName': '124564654654',
'ID': '124564654654'
}
}
```

参数名称	参数描述	类型
Bucket	存储桶名称，由 BucketName-APPID 构成	String
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String
UploadId	标识分块上传的 ID	String
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String
MaxParts	最多返回的分块的数量，默认为最大的1000	String
IsTruncated	表示返回的分块是否被截断	String
PartNumberMarker	默认为0，从第一块列出分块，从 PartNumberMarker 下一个分块开始列出	String
NextPartNumberMarker	指明下一次列出分块的起始位置	String
StorageClass	对象的存储类型，STANDARD，STANDARD_IA，ARCHIVE。默认值为 STANDARD，更多存储类型，请参见 存储类型概述	String
Part	上传分块的相关信息，包括 ETag，PartNumber，Size，LastModified	String
Initiator	分块上传的创建者，包括 DisplayName 和 ID	Dict
Owner	对象拥有者的信息，包括 DisplayName 和 ID	Dict

完成分块上传

功能说明

完成整个对象的分块上传（Complete Multipart Upload）。

方法原型

```
complete_multipart_upload(Bucket, Key, UploadId, MultipartUpload={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.complete_multipart_upload(
Bucket='examplebucket-1250000000',
Key='exampleobject',
UploadId='exampleUploadId',
MultipartUpload={
'Part': [
{
'ETag': 'string',
'PartNumber': 1
},
{
'ETag': 'string',
'PartNumber': 2
},
]
},
```

```
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	Bucket 名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
UploadId	标识分块上传的 ID	String	是
MultipartUpload	所有分块的 ETag 和 PartNumber 信息	Dict	是

返回结果说明

组装后的对象的相关信息，类型为 dict：

```
{
'ETag': '"3f866d0050f044750423e0a4104fa8cf-2"',
'Bucket': 'examplebucket-1250000000',
'Location': 'examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobject',
'Key': 'exampleobject'
}
```

参数名称	参数描述	类型
ETag	合并后对象的唯一标签值，该值不是对象内容的 MD5 校验值，仅能用于检查对象唯一性。如需校验对象内容，可以在上传过程中校验单个分块的 ETag 值。	String
Bucket	存储桶名称，由 BucketName-APPID 构成	String
Location	URL 地址	String
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块（Abort Multipart Upload）。

方法原型

```
abort_multipart_upload(Bucket, Key, UploadId, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.abort_multipart_upload(
Bucket='examplebucket-1250000000',
Key='exampleobject',
UploadId='exampleUploadId'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
UploadId	标识分块上传的 ID	String	是

返回结果说明

该方法返回值为 None。

下载对象

最近更新时间：2023-05-23 15:15:12

简介

本文档提供关于对象的简单操作、分块操作等其他操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地

高级接口（推荐）

下载对象（断点续传）

功能说明

该高级接口仅支持下载整个文件，根据用户文件的长度自动选择简单下载以及分块下载，对于小于等于20MB的文件使用简单下载，大于20MB的文件使用续传下载，对于分块下载未完成的文件会自动进行断点续传。

方法原型

```
download_file(Bucket, Key, DestFilePath, PartSize=20, MAXThread=5, EnableCRC=False, **Kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos.cos_exception import CosClientError, CosServiceError
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循
```

```
最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 使用高级接口下载一次，不重试，此时没有使用断点续传的功能
response = client.download_file(
Bucket='examplebucket-1250000000',
Key='exampleobject',
DestFilePath='local.txt'
)

# 使用高级接口断点续传，失败重试时不会下载已成功的分块(这里重试10次)
for i in range(0, 10):
try:
response = client.download_file(
Bucket='examplebucket-1250000000',
Key='exampleobject',
DestFilePath='local.txt')
break
except CosClientError or CosServiceError as e:
print(e)
```

全部参数请求示例

```
def upload_percentage(consumed_bytes, total_bytes):
"""进度条回调函数，计算当前上传的百分比

:param consumed_bytes: 已经上传的数据量
:param total_bytes: 总数据量
"""
if total_bytes:
rate = int(100 * (float(consumed_bytes) / float(total_bytes)))
print('\r{0}% '.format(rate))
```

```
sys.stdout.flush()

response = client.download_file(
Bucket='examplebucket-1250000000',
Key='exampleobject',
DestFilePath='local.txt',
PartSize=1,
MAXThread=5,
EnableCRC=True,
TrafficLimit='1048576'
IfMatch='"9a4802d5c99dafe1c04da0a8e7e166bf"',
IfModifiedSince='Wed, 28 Oct 2014 20:30:00 GMT',
IfNoneMatch='"9a4802d5c99dafe1c04da0a8e7e166bf"',
IfUnmodifiedSince='Wed, 28 Oct 2014 20:30:00 GMT',
ResponseCacheControl='string',
ResponseContentDisposition='string',
ResponseContentEncoding='string',
ResponseContentLanguage='string',
ResponseContentType='string',
ResponseExpires='string',
VersionId='string',
progress_callback=upload_percentage
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
DestFilePath	文件下载的本地目的路径名	String	是
PartSize	分块下载的分块大小，默认为20MB	Int	否
MAXThread	分块下载的并发数量，默认为5个线程下载分块	Int	否
EnableCRC	是否开启本地文件与远程文件的 crc 校验，默认为 False	Bool	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
TrafficLimit	单链接限速的值，单位为bit/s，限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，高级接口限制的是单线程的速度	String	否
IfMatch	ETag 与指定的内容一致时才返回	String	否
IfModifiedSince	在指定时间后被修改才返回，时间格式为 GMT	String	否
IfNoneMatch	ETag 与指定的内容不一致才返回	String	否
IfUnmodifiedSince	对象修改时间早于或等于指定时间才返回，时间格式为 GMT	String	否
ResponseCacheControl	设置响应头部 Cache-Control	String	否
ResponseContentDisposition	设置响应头部 Content-Disposition	String	否
ResponseContentEncoding	设置响应头部 Content-Encoding	String	否
ResponseContentLanguage	设置响应头部 Content-Language	String	否
ResponseContentType	设置响应头部 Content-Type	String	否
ResponseExpires	设置响应头部 Expires	String	否
VersionId	指定下载对象的版本	String	否
progress_callback	上传进度的回调函数，可以通过自定义此函数，来获取上传进度	Func	否

返回结果说明

None

批量下载（从 COS 下载目录）

功能说明

该示例展示通过组合 SDK 的基本接口，完成批量下载 COS 目录中的文件到本地磁盘。

请求示例

```
# -*- coding=utf-8
import os
import logging
import sys
```

```
import json
import os

from qcloud_cos import CosConfig, CosServiceError
from qcloud_cos import CosS3Client
from qcloud_cos.cos_threadpool import SimpleThreadPool

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme) # 获取配置对象
client = CosS3Client(config)

# 用户的 bucket 信息
test_bucket = 'examplebucket-1250000000'
start_prefix = 'data/'
# 对象存储依赖 分隔符 '/' 来模拟目录语义，
# 使用默认的空分隔符可以列出目录下面的所有子节点，实现类似本地目录递归的效果，
# 如果 delimiter 设置为 "/"，则需要在程序里递归处理子目录
delimiter = ''


# 列出当前目录子节点，返回所有子节点信息
def listCurrentDir(prefix):
file_infos = []
sub_dirs = []
marker = ""
count = 1
while True:
response = client.list_objects(test_bucket, prefix, delimiter, marker)
```

```
# 调试输出
# json_object = json.dumps(response, indent=4)
# print(count, " ====================================")
# print(json_object)
count += 1

if "CommonPrefixes" in response:
common_prefixes = response.get("CommonPrefixes")
sub_dirs.extend(common_prefixes)

if "Contents" in response:
contents = response.get("Contents")
file_infos.extend(contents)

if "NextMarker" in response.keys():
marker = response["NextMarker"]
else:
break

print("============================================================")

# 如果 delimiter 设置为 "/", 则需要进行递归处理子目录,
# sorted(sub_dirs, key=lambda sub_dir: sub_dir["Prefix"])
# for sub_dir in sub_dirs:
# print(sub_dir)
# sub_dir_files = listCurrentDir(sub_dir["Prefix"])
# file_infos.extend(sub_dir_files)

print("============================================================")

sorted(file_infos, key=lambda file_info: file_info["Key"])
for file in file_infos:
print(file)
return file_infos


# 下载文件到本地目录, 如果本地目录已经有同名文件则会被覆盖；
# 如果目录结构不存在, 则会创建和对象存储一样的目录结构
def downLoadFiles(file_infos):
localDir = "./download/"

pool = SimpleThreadPool()
for file in file_infos:
# 文件下载 获取文件到本地
file_cos_key = file["Key"]
localName = localDir + file_cos_key
```

```
# 如果本地目录结构不存在，递归创建
if not os.path.exists(os.path.dirname(localName)):
os.makedirs(os.path.dirname(localName))

# skip dir, no need to download it
if str(localName).endswith("/"):
continue

# 实际下载文件
# 使用线程池方式
pool.add_task(client.download_file, test_bucket, file_cos_key, localName)

# 简单下载方式
# response = client.get_object(
# Bucket=test_bucket,
# Key=file_cos_key,
# )
# response['Body'].get_stream_to_file(localName)

pool.wait_completion()
return None


# 功能封装，下载对象存储上面的一个目录到本地磁盘
def downLoadDirFromCos(prefix):
global file_infos

try:
file_infos = listCurrentDir(prefix)

except CosServiceError as e:
print(e.get_origin_msg())
print(e.get_digest_msg())
print(e.get_status_code())
print(e.get_error_code())
print(e.get_error_msg())
print(e.get_resource_location())
print(e.get_trace_id())
print(e.get_request_id())

downLoadFiles(file_infos)
return None


if __name__ == "__main__":
downLoadDirFromCos(start_prefix)
```

简单操作

下载对象

功能说明

下载一个对象到本地（GET Object）。

方法原型

```
get_object(Bucket, Key, **kwargs)
```

请求示例1：下载对象

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_object(
Bucket='examplebucket-1250000000',
```

```
Key='exampleobject'
)
response['Body'].get_stream_to_file('exampleobject')
```

请求示例2：下载对象部分内容

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Range='bytes=0-100'
)
response['Body'].get_stream_to_file('exampleobject')
```

全部参数请求示例

```
response = client.get_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Range='string',
IfMatch='"9a4802d5c99dafe1c04da0a8e7e166bf"',
IfModifiedSince='Wed, 28 Oct 2014 20:30:00 GMT',
IfNoneMatch='"9a4802d5c99dafe1c04da0a8e7e166bf"',
IfUnmodifiedSince='Wed, 28 Oct 2014 20:30:00 GMT',
ResponseCacheControl='string',
ResponseContentDisposition='string',
ResponseContentEncoding='string',
ResponseContentLanguage='string',
ResponseContentType='string',
ResponseExpires='string',
VersionId='string',
TrafficLimit='819200'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
Range	设置下载对象的范围，格式为 bytes=first-last	String	否
IfMatch	ETag 与指定的内容一致时才返回	String	否
IfModifiedSince	在指定时间后被修改才返回，时间格式为 GMT	String	否
IfNoneMatch	ETag 与指定的内容不一致才返回	String	否
IfUnmodifiedSince	对象修改时间早于或等于指定时间才返回，时间格式为 GMT	String	否
ResponseCacheControl	设置响应头部 Cache-Control	String	否
ResponseContentDisposition	设置响应头部 Content-Disposition	String	否
ResponseContentEncoding	设置响应头部 Content-Encoding	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
ResponseContentLanguage	设置响应头部 Content-Language	String	否
ResponseContentType	设置响应头部 Content-Type	String	否
ResponseExpires	设置响应头部 Expires	String	否
VersionId	指定下载对象的版本	String	否
TrafficLimit	单链接限速的值，单位为bit/s，限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，高级接口限制的是单线程的速度	String	否

返回结果说明

下载对象的 Body 和元信息，类型为 dict：

```
{
'Body': StreamBody(),
'ETag': '"9a4802d5c99dafe1c04da0a8e7e166bf"',
'Last-Modified': 'Wed, 28 Oct 2014 20:30:00 GMT',
'Accept-Ranges': 'bytes',
'Content-Range': 'bytes 0-16086/16087',
'Cache-Control': 'max-age=1000000',
'Content-Type': 'application/octet-stream',
'Content-Disposition': 'attachment; filename="filename.jpg"',
'Content-Encoding': 'gzip',
'Content-Language': 'zh-cn',
'Content-Length': '16807',
'Expires': 'Wed, 28 Oct 2019 20:30:00 GMT',
'x-cos-meta-test': 'test',
'x-cos-version-id': 'MTg0NDUxODMzMTMwMDM2Njc1ODA',
'x-cos-request-id': 'NTg3NzQ3ZmVfYmRjMzVfMzE5N182NzczMQ=='
}
```

参数名称	参数描述	类型
Body	下载对象的内容，get_raw_stream() 方法可以得到一个文件流，get_stream_to_file(local_file_path) 方法可以将对象内容下载到指定本地文件中	StreamBody
ETag	对象的 MD5 值	String
Last-Modified	对象最后修改时间	String
Accept-Ranges	范围单位， HTTP 标准头部	String

版权所有：腾讯云计算（北京）有限责任公司

Content-Range	内容范围， HTTP 标准头部	String
Cache-Control	缓存策略， HTTP 标准头部	String
Content-Type	内容类型，HTTP 标准头部	String
Content-Disposition	文件名称，HTTP 标准头部	String
Content-Encoding	编码格式，HTTP 标准头部	String
Content-Language	语言类型，HTTP 标准头部	String
Content-Length	对象大小	String
Expires	缓存过期时间， HTTP 标准头部	String
x-cos-meta-*	用户自定义的对象元数据， 必须以 x-cos-meta 开头，否则会被忽略	String
x-cos-version-id	开启版本控制后，对象的版本号	String

版权所有：腾讯云计算（北京）有限责任公司

复制和移动对象

最近更新时间：2023-05-22 15:28:17

简介

本文档提供关于复制、移动对象的相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Object - Copy	设置对象复制	复制对象到目标路径

高级接口（推荐）

复制对象

功能说明

该高级接口可支持小于5G的文件调用 copy_object，大于等于5G的文件调用分块上传的 upload_part_copy。

方法原型

```
copy(Bucket, Key, CopySource, CopyStatus='Copy', PartSize=10, MAXThread=5, **kwargs)
```

请求示例1：复制对象

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
```

```
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.copy(
Bucket='examplebucket-1250000000',
Key='exampleobject',
CopySource={
'Bucket': 'sourcebucket-1250000000',
'Key': 'sourceobject',
'Region': 'ap-guangzhou'
}
)
```

请求示例2：移动对象

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，htt
```

```
ps://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

bucket = 'examplebucket-1250000000'
srcKey = 'src_object_key' # 原始的对象路径
destKey = 'dest_object_key' # 目的对象路径

# COS本身没有移动对象的接口，所谓移动就是先拷贝老对象到新对象，再删除老对象。
response = client.copy(
Bucket=bucket,
Key=destKey,
CopySource={
'Bucket':bucket,
'Key':srcKey,
'Region':'ap-guangzhou',
})
client.delete_object(Bucket=bucket, Key=srcKey)
```

全部参数请求示例

```
response = client.copy(
Bucket='examplebucket-1250000000',
Key='exampleobject',
CopySource={
'Bucket': 'sourcebucket-1250000000',
'Key': 'exampleobject',
'Region': 'ap-guangzhou'
}
CopyStatus='Copy'|'Replaced',
PartSize=20,
MAXThread=10
)
```

参数说明

参数名称	参数描述	类型	是否必填

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
CopySource	描述拷贝源对象的路径，包含 Bucket、Key、Region、VersionId	Dict	是
CopyStatus	拷贝状态，可选值 Copy、Replaced	String	否
PartSize	分块下载的分块大小，默认为10MB	Int	否
MAXThread	分块下载的并发数量，默认为5个线程下载分块	Int	否

返回结果说明

若小于5G的文件，则为 copy_object 的返回结果，否则为 complete_multipart_upload 的返回结果，类型为 dict。

简单操作

复制对象

功能说明

复制文件到目标路径（PUT Object - Copy）。

方法原型

```
copy_object(Bucket, Key, CopySource, CopyStatus='Copy', **kwargs)
```

请求示例1：复制对象

源和目标是不同的对象，会生成一个新的目标对象，目标对象会复制源对象的元数据信息。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
```

```
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id, secret_key, region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.copy_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
CopySource={
'Bucket': 'sourcebucket-1250000000',
'Key': 'sourceobject',
'Region': 'ap-guangzhou'
}
)
```

请求示例2：移动对象

COS本身没有移动对象的接口，所谓移动就是先拷贝老对象到新对象，再删除老对象。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
```

```
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.copy_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
CopySource={
'Bucket': 'sourcebucket-1250000000',
'Key': 'sourceobject',
'Region': 'ap-guangzhou'
}
)
client.delete_object(Bucket='sourcebucket-1250000000', Key='sourceobject')
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
CopySource	描述拷贝源对象的路径，包含 Bucket、Key、Region、VersionId	Dict	是

参数名称	参数描述	类型	是否必填
CopyStatus	可选值为 Copy、Replaced，设置为 Copy 时，忽略设置的用户元数据信息直接复制，设置为 Replaced 时，按设置的元信息修改元数据，当目标路径和源路径一样时，必须设置为 Replaced	String	是
ACL	设置对象的 ACL，如 private，public-read	String	否
GrantFullControl	赋予指定账户对对象的所有权限，格式为 `id="OwnerUin"`	String	否
GrantRead	赋予指定账户对对象的读权限，格式为 `id="OwnerUin"`	String	否
StorageClass	设置对象的存储类型，STANDARD，STANDARD_IA，默认值 STANDARD	String	否
Expires	设置 Expires	String	否
CacheControl	缓存策略，设置 Cache-Control	String	否
ContentType	内容类型，设置 Content-Type	String	否
ContentDisposition	文件名称，设置 Content-Disposition	String	否
ContentEncoding	编码格式，设置 Content-Encoding	String	否
ContentLanguage	语言类型，设置 Content-Language	String	否
Metadata	用户自定义的对象元数据	Dict	否

返回结果说明

上传对象的属性，类型为 dict：

```
{
'ETag': 'string',
'LastModified': 'string',
'VersionId': 'string',
'x-cos-copy-source-version-id': 'string'
}
```

参数名称	参数描述	类型
ETag	拷贝对象的 MD5 值	String
LastModified	拷贝对象的最后一次修改时间	String
VersionId	开启版本控制后，目的对象的版本号	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
x-cos-copy-source-version-id	源对象的版本号	String

版权所有：腾讯云计算（北京）有限责任公司

列出对象

最近更新时间：2023-05-23 15:09:23

简介

本文档提供关于对象列举操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Object）	查询对象列表	查询存储桶下的部分或者全部对象
GET Bucket Object Versions	查询对象及其历史版本列表	查询存储桶下的部分或者全部对象及其历史版本信息

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

方法原型

```
list_objects(Bucket, Delimiter="", Marker="", MaxKeys=1000, Prefix="", EncodingType="", **kwargs)
```

请求示例1：列举指定存储桶中的所有对象

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循
```

```
最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.list_objects(Bucket='examplebucket-1250000000')
if 'Contents' in response:
for content in response['Contents']:
print(content['Key'])
# 注：如果桶内对象太多，则一次列举只返回1000个对象，需要分页列举，参见示例3
```

请求示例2：列举指定前缀的对象

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/43
```

```
6/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 列举前缀为 folder 的对象
response = client.list_objects(
Bucket='examplebucket-1250000000',
Prefix='folder'
)

# 列举 folder1 目录下的文件：COS中的目录是'/'结尾的前缀名
response = client.list_objects(
Bucket='examplebucket-1250000000',
Prefix='folder1/'
)
```

请求示例3：分页列举对象

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用
```

```
指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token
=token, Scheme=scheme)
client = CosS3Client(config)

# 分页列举桶内对象，每个分页10个对象
marker = ""
while True:
response = client.list_objects(
Bucket='examplebucket-1250000000', Prefix='folder1/', Marker=marker, MaxKeys=10)
if 'Contents' in response:
for content in response['Contents']:
print(content['Key'])

if response['IsTruncated'] == 'false':
break

marker = response["NextMarker"]
```

请求示例4：列出指定目录下的对象和子目录

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移
除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循
最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/
product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权
遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/docum
ent/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，htt
ps://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/43
6/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用
```

```
指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token
=token, Scheme=scheme)
client = CosS3Client(config)

# 列举 folder1 目录下的文件和子目录
response = client.list_objects(
Bucket='examplebucket-1250000000', Prefix='folder1/', Delimiter='/')
# 打印文件列表
if 'Contents' in response:
for content in response['Contents']:
print(content['Key'])
# 打印子目录
if 'CommonPrefixes' in response:
for folder in response['CommonPrefixes']:
print(folder['Prefix'])
```

全部参数请求示例

```
response = client.list_objects(
Bucket='examplebucket-1250000000',
Prefix='string',
Delimiter='/',
Marker='string',
MaxKeys=100,
EncodingType='url'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Prefix	默认为空，对对象的对象键进行筛选，匹配 prefix 为前缀的对象	String	否
Delimiter	默认为空，设置分隔符，例如设置 `/` 来模拟文件夹	String	否
Marker	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的起点位置	String	否
MaxKeys	最多返回的对象数量，默认为最大的1000	Int	否
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String	否

版权所有：腾讯云计算（北京）有限责任公司

返回结果说明

获取对象的元信息，类型为 dict：

```
{
'MaxKeys': '1000',
'Prefix': 'string',
'Delimiter': 'string',
'Marker': 'string',
'NextMarker': 'string',
'Name': 'examplebucket-1250000000',
'IsTruncated': 'false'|'true',
'EncodingType': 'url',
'Contents':[
{
'ETag': '"a5b2e1cfb08d10f6523f7e6fbf3643d5"',
'StorageClass': 'STANDARD',
'Key': 'exampleobject',
'Owner': {
'DisplayName': '1250000000',
'ID': '1250000000'
},
'LastModified': '2017-08-08T09:43:35.000Z',
'Size': '23'
},
],
'CommonPrefixes':[
{
'Prefix': 'string'
},
],
}
```

参数名称	参数描述	类型
MaxKeys	最多返回的对象数量，默认为最大的1000	String
Prefix	默认为空，匹配 Prefix 为前缀的对象	String
Delimiter	默认为空，设置分隔符，例如设置 `/` 来模拟文件夹	String
Marker	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的起点位置	String
NextMarker	当 IsTruncated 为 true 时，标记下一次返回对象的 list 的起点位置	String
Name	存储桶名称，由 BucketName-APPID 构成	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
IsTruncated	表示返回的对象是否被截断	String
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String
Contents	包含所有对象元数据的 list，包括 'ETag'，'StorageClass'，'Key'，'Owner'，'LastModified'，'Size' 等信息	List
CommonPrefixes	所有以 Prefix 开头，以 Delimiter 结尾的对象被归到同一类	List

查询对象及其历史版本列表

功能说明

查询存储桶下的部分或者全部对象及其历史版本信息。

方法原型

```
list_objects_versions(Bucket, Prefix="", Delimiter="", KeyMarker="", VersionIdMarker="", MaxKeys=1000, EncodingType="", **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
```

```
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.list_objects_versions(
Bucket='examplebucket-1250000000',
Prefix='string'
)
```

全部参数请求示例

```
response = client.list_objects_versions(
Bucket='examplebucket-1250000000',
Prefix='string',
Delimiter='/',
KeyMarker='string',
VersionIdMarker='string',
MaxKeys=100,
EncodingType='url'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Prefix	默认为空，对对象的对象键进行筛选，匹配 prefix 为前缀的对象	String	否
Delimiter	默认为空，设置分隔符，例如设置 `/` 来模拟文件夹	String	否
KeyMarker	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 Key 的起点位置	String	否
VersionIdMarker	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 VersionId 的起点位置	String	否
MaxKeys	最多返回的对象数量，默认为最大的1000	Int	否
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String	否

版权所有：腾讯云计算（北京）有限责任公司

返回结果说明

获取对象的元信息，类型为 dict：

```
{
'MaxKeys': '1000',
'Prefix': 'string',
'Delimiter': 'string',
'KeyMarker': 'string',
'VersionIdMarker': 'string',
'NextKeyMarker': 'string',
'NextVersionIdMarker': 'string',
'Name': 'examplebucket-1250000000',
'IsTruncated': 'false'|'true',
'EncodingType': 'url',
'Version':[
{
'ETag': '"a5b2e1cfb08d10f6523f7e6fbf3643d5"',
'StorageClass': 'STANDARD',
'Key': 'exampleobject',
'VersionId': 'string',
'IsLatest': 'true'|'false',
'Owner': {
'DisplayName': '1250000000',
'ID': '1250000000'
},
'LastModified': '2017-08-08T09:43:35.000Z',
'Size': '23'
},
],
'DeleteMarker': [
{
'Key': 'exampleobject',
'VersionId': 'string',
'IsLatest': 'true'|'false',
'Owner': {
'DisplayName': '1250000000',
'ID': '1250000000'
},
'LastModified': '2017-08-08T09:43:35.000Z'
},
],
'CommonPrefixes':[
{
'Prefix': 'string'
},
```

```
],
}
```

参数名称	参数描述
MaxKeys	最多返回的对象数量，默认为最大的1000
Prefix	默认为空，匹配 Prefix 为前缀的对象
Delimiter	默认为空，设置分隔符，例如设置 `/` 来模拟文件夹
KeyMarker	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 Key 的起点位置
VersionIdMarker	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 VersionId 的起点位置
NextKeyMarker	当 IsTruncated 为 true 时，标记下一次返回对象的 list 的 Key 的起点位置
NextVersionIdMarker	当 IsTruncated 为 true 时，标记下一次返回对象的 list 的 VersionId 的起点位置
Name	存储桶名称，由 BucketName-APPID 构成
IsTruncated	表示返回的对象是否被截断
EncodingType	默认不编码，规定返回值的编码方式，可选值：url
Version	包含所有多个版本对象元数据的 list，包括 'ETag'，'StorageClass'，'Key'，'VersionId'，'IsLatest'，'Owner'，'LastModified'，'Size' 等信息
DeleteMarker	包含所有delete marker 对象元数据的 list，包括 'Key'，'VersionId'，'IsLatest'，'Owner'，'LastModified' 等信息
CommonPrefixes	所有以 Prefix 开头，以 Delimiter 结尾的对象被归到同一类

版权所有：腾讯云计算（北京）有限责任公司

删除对象

最近更新时间：2023-06-14 11:31:01

简介

本文档提供关于删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除指定对象

删除单个对象

功能说明

删除指定的对象（DELETE Object）。

方法原型

```
delete_object(Bucket, Key, **kwargs)
```

请求示例1：删除单个对象

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权
```

```
遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.delete_object(
Bucket='examplebucket-1250000000',
Key='exampleobject'
)
```

请求示例2：删除目录

对象存储中，目录是特殊的路径以“/”结尾的 object。可直接调用 Delete Object 接口实现删除目录。注意，仅支持删除空目录，如果要删除非空目录，请参考“前缀批量删除”。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
```

```
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

to_delete_dir='path/to/delete/dir/'
response = client.delete_object(
Bucket='examplebucket-1250000000',
Key=to_delete_dir,
)
print(response)
```

请求示例3：前缀批量删除

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client

import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
```

```
client = CosS3Client(config)

# 删除指定前缀 (prefix)的文件
bucket = 'examplebucket-1250000000'
is_over = False
marker = ''
prefix = 'root/logs'
while not is_over:
response = client.list_objects(Bucket=bucket, Prefix=prefix, Marker=marker)
if 'Contents' in response:
for content in response['Contents']:
print("delete object: ", content['Key'])
client.delete_object(Bucket=bucket, Key=content['Key'])

if response['IsTruncated'] == 'false':
is_over = True
marker = response['Marker']
```

全部参数请求示例

```
response = client.delete_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
VersionId='string',
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
VersionId	开启版本控制后，指定对象的具体版本	String	否

返回结果说明

删除对象的信息，类型为 dict：

```
{
'x-cos-version-id': 'string',
```

```
'x-cos-delete-marker': 'true'|'false',
}
```

参数名称	参数描述	类型
x-cos-version-id	删除对象的版本号	String
x-cos-delete-marker	标识删除的对象是否为 delete marker	String

删除多个对象

功能说明

删除多个指定的对象（DELETE Multiple Objects）。

方法原型

```
delete_objects(Bucket, Delete={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用
```

```
指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token
=token, Scheme=scheme)
client = CosS3Client(config)

response = client.delete_objects(
Bucket='examplebucket-1250000000',
Delete={
'Object': [
{
'Key': 'exampleobject1'
},
{
'Key': 'exampleobject2'
}
]
}
)
```

全部参数请求示例

```
response = client.delete_objects(
Bucket='examplebucket-1250000000',
Delete={
'Object': [
{
'Key': 'exampleobject1',
'VersionId': 'string'
},
{
'Key': 'exampleobject2',
'VersionId': 'string'
},
],
'Quiet': 'true'|'false'
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	Bucket 名称，由 BucketName-APPID 构成	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
Delete	说明本次删除的返回结果方式和目标 Object	Dict	是
Object	说明每个将要删除的目标 Object 信息	List	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	否
VersionId	开启版本控制后，目的对象的版本号	String	否
Quiet	指明删除的返回结果方式，可选值为 true、false，默认为 false。设置为 true 只返回失败的错误信息，设置为 false 时返回成功和失败的所有信息	String	否

返回结果说明

批量删除对象的结果，类型为 dict：

```
{
'Deleted': [
{
'Key': 'string',
'VersionId': 'string',
'DeleteMarker': 'true'|'false',
'DeleteMarkerVersionId': 'string'
},
{
'Key': 'string',
},
],
'Error': [
{
'Key': 'string',
'VersionId': 'string',
'Code': 'string',
'Message': 'string'
},
]
}
```

参数名称	参数描述	类型
Deleted	删除成功的 Object 信息	List
Key	删除成功的 Object 的路径	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
VersionId	删除成功的 Object 的版本号	String
DeleteMarker	删除成功的 Object 是否为 delete marker	String
DeleteMarkerVersionId	删除成功的 Object 的 delete marker 的版本号	String
Error	删除失败的 Object 信息	List
Key	删除失败的 Object 的路径	String
VersionId	删除失败的 Object 的版本号	String
Code	删除失败的 Object 对应的错误码	String
Message	删除失败的 Object 对应的错误信息	String

批量删除对象（删除目录）

功能说明

对象存储中本身是没有目录的概念的，为了满足用户使用习惯，用户可通过分隔符/来模拟“目录”。

删除目录及其文件这一场景，实际在 COS 上相当于删除一批有着同样前缀的对象。目前 COS Python SDK 没有提供一个接口去实现这样的操作，但是可以通过组合查询对象列表加上批量删除对象的基本操作，达到删除目录及其文件的效果。

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos.cos_threadpool import SimpleThreadPool
import os
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循
```

```
最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

bucket = 'examplebucket-1250000000'
folder = 'folder/' # 要删除的目录，'/'结尾表示目录

def delete_cos_dir():
pool = SimpleThreadPool()
marker = ""
while True:
file_infos = []

# 列举一页100个对象
response = client.list_objects(Bucket=bucket, Prefix=folder, Marker=marker, MaxKeys=100)

if "Contents" in response:
contents = response.get("Contents")
file_infos.extend(contents)
pool.add_task(delete_files, file_infos)

# 列举完成，退出
if response['IsTruncated'] == 'false':
break

# 列举下一页
marker = response["NextMarker"]

pool.wait_completion()
return None

def delete_files(file_infos):
```

```
# 构造批量删除请求
delete_list = []
for file in file_infos:
delete_list.append({"Key": file['Key']})

response = client.delete_objects(Bucket=bucket, Delete={"Object": delete_list})
print(response)

if __name__ == "__main__":
delete_cos_dir()
```

判断对象是否存在

最近更新时间：2023-05-22 15:19:26

简介

本文档提供判断对象是否存在的 SDK 示例代码。

判断对象是否存在

功能说明

检查存储桶中是否存在某个对象。

方法原型

```
object_exists(Bucket, Key)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
```

版权所有：腾讯云计算（北京）有限责任公司

```
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.object_exists(
Bucket='examplebucket-1250000000',
Key='exampleobject')
print(response)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是

返回结果说明

True 表示对象存在，False 表示对象不存在。

查询对象元数据

最近更新时间：2023-05-22 15:24:02

简介

本文档提供关于查询对象元数据操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

查询对象元数据

功能说明

查询对象的元数据信息（HEAD Object）。

方法原型

```
head_object(Bucket, Key, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
```

```
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.head_object(
Bucket='examplebucket-1250000000',
Key='exampleobject'
)
print(response)
```

全部请求参数示例

```
response = client.head_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
VersionId='string',
IfModifiedSince='Wed, 28 Oct 2020 20:30:00 GMT',
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	Bucket 名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
VersionId	开启版本控制后，指定对象的具体版本	String	否
IfModifiedSince	在指定时间后被修改才返回，时间格式为 GMT	String	否

返回结果说明

获取对象的元信息，类型为 dict：

```
{
'ETag': '"9a4802d5c99dafe1c04da0a8e7e166bf"',
'Last-Modified': 'Wed, 28 Oct 2014 20:30:00 GMT',
'Cache-Control': 'max-age=1000000',
'Content-Type': 'application/octet-stream',
'Content-Disposition': 'attachment; filename="filename.jpg"',
'Content-Encoding': 'gzip',
'Content-Language': 'zh-cn',
'Content-Length': '16807',
'Expires': 'Wed, 28 Oct 2019 20:30:00 GMT',
'Connection': 'keep-alive',
'Accept-Ranges': 'bytes',
'Server': 'tencent-cos',
'Date': 'Wed, 28 Oct 2019 20:20:00 GMT',
'x-cos-meta-test': 'test',
'x-cos-version-id': 'MTg0NDUxODMzMTMwMDM2Njc1ODA',
'x-cos-hash-crc64ecma': '1220413902487941276',
'x-cos-request-id': 'NTg3NzQ3ZmVfYmRjMzVfMzE5N182NzczMQ=='
}
```

参数名称	参数描述	类型
ETag	分块上传时，该值不是对象内容的 MD5 校验值，仅能用于检查对象唯一性	String
Last-Modified	对象最后修改时间	String
Cache-Control	缓存策略， HTTP 标准头部	String
Content-Type	内容类型，HTTP 标准头部	String
Content-Disposition	文件名称，HTTP 标准头部	String
Content-Encoding	编码格式，HTTP 标准头部	String
Content-Language	语言类型，HTTP 标准头部	String
Content-Length	对象大小	String
Expires	缓存过期时间， HTTP 标准头部	String
Connection	连接方式，HTTP 标准头部	String
Accept-Ranges	范围请求标识，HTTP 标准头部	String
Server	服务端标识，HTTP 标准头部	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
Date	请求日期，HTTP 标准头部	String
x-cos-meta-*	用户自定义的对象元数据， 必须以 x-cos-meta 开头，否则会被忽略	String
x-cos-hash-crc64ecma	文件的 CRC64 校验码	String
x-cos-version-id	开启版本控制后，对象的版本号	String
x-cos-request-id	请求 ID	String

版权所有：腾讯云计算（北京）有限责任公司

修改对象元数据

最近更新时间：2023-05-23 15:12:32

简介

本文档提供关于修改对象元数据操作相关的 SDK 示例代码。

修改对象元数据

源和目标是同一个对象，且 CopyStatus 设置为 Replaced，即可以通过头域来修改对象的元数据。

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
```

```
client = CosS3Client(config)

response = client.copy_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
CopySource={
'Bucket': 'sourcebucket-1250000000',
'Key': 'exampleobject',
'Region': 'ap-guangzhou'
},
CopyStatus='Replaced',
ContentType='text/plain', # 修改对象 ContentType
Metadata={'x-cos-meta-key1': 'value1', 'x-cos-meta-key2': 'value2'} # 修改自定义元数据
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
CopySource	描述拷贝源对象的路径，包含 Bucket、Key、Region、VersionId	Dict	是
CopyStatus	可选值为 Copy、Replaced，设置为 Copy 时，忽略设置的用户元数据信息直接复制，设置为 Replaced 时，按设置的元信息修改元数据，当目标路径和源路径一样时，必须设置为 Replaced	String	是
ACL	设置对象的 ACL，如 private，public-read	String	否
GrantFullControl	赋予指定账户对对象的所有权限，格式为 `id="OwnerUin"`	String	否
GrantRead	赋予指定账户对对象的读权限，格式为 `id="OwnerUin"`	String	否
Expires	设置 Expires	String	否
CacheControl	缓存策略，设置 Cache-Control	String	否
ContentType	内容类型，设置 Content-Type	String	否
ContentDisposition	文件名称，设置 Content-Disposition	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
ContentEncoding	编码格式，设置 Content-Encoding	String	否
ContentLanguage	语言类型，设置 Content-Language	String	否
Metadata	用户自定义的对象元数据	Dict	否

修改对象存储类型

源和目标是同一个对象，且 CopyStatus 设置为 Replaced，即可以通过 StorageClass 参数来修改对象的存储类型。

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)
```

```
response = client.copy_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
CopySource={
'Bucket': 'examplebucket-1250000000',
'Key': 'exampleobject',
'Region': 'ap-guangzhou'
},
CopyStatus='Replaced',
StorageClass='STANDARD_IA' # 修改为低频存储
)
```

全部参数请求示例

```
response = client.copy_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
CopySource={
'Bucket': 'sourcebucket-1250000000',
'Key': 'exampleobject',
'Region': 'ap-guangzhou',
'VersionId': 'string'
},
CopyStatus='Copy'|'Replaced',
ACL='private'|'public-read',
GrantFullControl='string',
GrantRead='string',
StorageClass='STANDARD'|'STANDARD_IA',
Expires='string',
CacheControl='string',
ContentType='string',
ContentDisposition='string',
ContentEncoding='string',
ContentLanguage='string',
Metadata={
'x-cos-meta-key1': 'value1',
'x-cos-meta-key2': 'value2'
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
CopySource	描述拷贝源对象的路径，包含 Bucket、Key、Region、VersionId	Dict	是
CopyStatus	可选值为 Copy、Replaced，设置为 Copy 时，忽略设置的用户元数据信息直接复制，设置为 Replaced 时，按设置的元信息修改元数据，当目标路径和源路径一样时，必须设置为 Replaced	String	是
StorageClass	设置对象的存储类型，STANDARD，STANDARD_IA，默认值 STANDARD	String	否

对象访问 URL

最近更新时间：2023-05-23 15:35:36

简介

本文档提供获取对象访问 URL 的代码示例。

获取对象访问 URL

功能说明

获取对象访问 URL 用于匿名下载或分发。

方法原型

```
get_object_url(Bucket, Key)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging
import requests

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/43
```

版权所有：腾讯云计算（北京）有限责任公司

```
6/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 生成URL
url = client.get_object_url(
Bucket='examplebucket-1250000000',
Key='exampleobject'
)
print(url)

# 使用URL
response = requests.get(url)
print(response)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是

返回结果说明

该方法返回值为对象访问的 URL。

生成预签名 URL

最近更新时间：2023-06-16 15:18:23

简介

Python SDK 提供获取签名、预签名 URL 、下载预签名 URL 的接口，用于将请求分发使用。使用永久密钥或临时密钥获取预签名 URL 的调用方法相同，使用临时密钥时需要在 header 或 query_string 中加上 x-cos-security-token。关于使用预签名 url 上传的说明请参见 预签名授权上传， 使用预签名 url 下载的说明请参见 预签名授权下载。

说明：

- 在获取签名时强烈建议将敏感请求头部和请求参数算入签名，这样可以避免相关请求头部和请求参数被使用者篡改，杜绝权限越界的情况发生。同时 SDK 会默认将请求域名算入签名，如果分发后修改了请求域名会导致访问失败，此时可以在获取签名时传入参数忽略请求域名，详细方法参见下面的请求示例。
- 建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。

获取预签名 URL

功能说明

获取预签名链接用于分发。

方法原型

```
get_presigned_url(Bucket, Key, Method, Expired=300, Params={}, Headers={})
```

请求示例1：生成上传预签名 URL

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
```

```
import logging
import requests

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 生成上传 URL，未限制请求头部和请求参数
url = client.get_presigned_url(
Method='PUT',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(url)

# 生成上传 URL，同时限制存储类型和上传速度
url = client.get_presigned_url(
Method='PUT',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Headers={
'x-cos-storage-class':'STANDARD_IA',
'x-cos-traffic-limit':'819200' # 预签名 URL 本身是不包含请求头部的，但请求头部会算入签名，那么使用 URL 时就必须携带请求头部，并且请求头部的值必须是这里指定的值
},
Expired=300 # 300秒后过期，过期时间请根据自身场景定义
)
```

版权所有：腾讯云计算（北京）有限责任公司

```
print(url)

# 生成上传 URL，只能上传指定的文件内容
url = client.get_presigned_url(
Method='PUT',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Headers={'Content-MD5':'string'}, # 约定使用此 URL 上传对象的人必须携带 MD5 请求头部，并且请求头部的值必须是这里指定的值，这样就限定了文件的内容
Expired=300 # 300秒后过期，过期时间请根据自身场景定义
)
print(url)

# 生成上传 URL，只能用于上传 ACL
url = client.get_presigned_url(
Method='PUT',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Params={'acl':''}, # 指定了请求参数，则 URL 中会携带此请求参数，并且请求参数会算入签名，不允许使用者修改请求参数的值
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(url)

# 生成上传 URL，请求域名不算入签名，签名后使用者需要修改请求域名时使用
url = client.get_presigned_url(
Method='PUT',
Bucket='examplebucket-1250000000',
Key='exampleobject',
SignHost=False, # 请求域名不算入签名，允许使用者修改请求域名，有一定安全风险
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(url)

# 使用上传 URL
response = requests.put(url=url, data=b'123')
print(response)
```

请求示例2：生成下载预签名 URL

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging
```

```
import requests

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 生成下载 URL，未限制请求头部和请求参数
url = client.get_presigned_url(
Method='GET',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(url)

# 生成下载 URL，同时指定响应的 content-disposition 头部，让文件在浏览器另存为，而不是显示
url = client.get_presigned_url(
Method='GET',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Params={
'response-content-disposition':'attachment; filename=example.xlsx' # 下载时保存为指定的文件
# 除了 response-content-disposition，还支持 response-cache-control、response-content-encoding、response-content-language、
# response-content-type、response-expires 等请求参数，详见下载对象 API，https://cloud.tencent.com/document/product/436/7753
},
```

```
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(url)

# 生成下载 URL，同时限制下载速度
url = client.get_presigned_url(
Method='GET',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Headers={'x-cos-traffic-limit':'819200'}, # 预签名URL本身是不包含请求头部的，但请求头部会算入签名，那么使用 URL 时就必须携带请求头部，并且请求头部的值必须是这里指定的值
Expired=300 # 300秒后过期，过期时间请根据自身场景定义
)
print(url)

# 生成下载 URL，只能用于下载 ACL
url = client.get_presigned_url(
Method='GET',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Params={'acl':''}, # 指定了请求参数，则 URL 中会携带此请求参数，并且请求参数会算入签名，不允许使用者修改请求参数的值
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(url)

# 生成下载 URL，请求域名不算入签名，签名后使用者需要修改请求域名时使用
url = client.get_presigned_url(
Method='GET',
Bucket='examplebucket-1250000000',
Key='exampleobject',
SignHost=False, # 请求域名不算入签名，允许使用者修改请求域名，有一定安全风险
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(url)

# 使用下载URL
response = requests.get(url)
print(response)
```

请求示例3：使用临时密钥生成下载预签名 URL

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
```

```
import os
import logging
import requests

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 生成下载 URL
url = client.get_presigned_url(
Method='GET',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Headers={'x-cos-traffic-limit':'819200'}, # 预签名 URL 本身是不包含请求头部的，但请求头部会算入签名，那么使用 URL 时就必须携带请求头部，并且请求头部的值必须是这里指定的值
Params={
'x-cos-security-token': 'string' # 使用临时密钥需要填入 Token 到请求参数
},
Expired=120, # 120秒后过期，过期时间请根据自身场景定义
SignHost=False # 请求域名不算入签名，签名后使用者需要修改请求域名时使用，有一定安全风险
)
print(url)

# 使用下载 URL
response = requests.get(url)
print(response)
```

全部参数请求示例

```
response = client.get_presigned_url(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Method='PUT'|'POST'|'GET'|'DELETE'|'HEAD',
Expired=300,
Headers={
'header1': 'string',
'header2': 'string',
},
Params={
'param1': 'string',
'param2': 'string'
},
SignHost=True|False
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg **(用户无需进行URL编码)**	String	是
Method	对应操作的 Method, 可选值为 'PUT'，'POST'，'GET'，'DELETE'，'HEAD'	String	是
Expired	签名过期时间，单位为秒	Int	否
Params	预签名 URL 中的请求参数。指定了请求参数，则 URL 中会携带此请求参数，并且请求参数会算入签名，不允许使用者修改请求参数的值。可以携带的 Params 和具体的操作相关，例如下载对象可以携带和签入的 Params 参见 GET Object 中的请求参数描述	Dict	否
Headers	预签名 URL 中要签入的请求头部。预签名 URL 本身是不包含请求头部的，但请求头部会算入签名，那么使用 URL 时就必须携带请求头部，并且请求头部的值必须是这里指定的值。可以签入的 Headers 和具体的操作相关，例如上传对象可以签入的 Headers 参见 PUT Object 中的请求头描述	Dict	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
SignHost	请求域名是否算入签名，默认值 True，签名后使用者需要修改请求域名时设置为 False	Bool	否

返回结果说明

该方法返回值为预签名的 URL。

获取预签名下载 URL

功能说明

获取预签名下载链接用于直接下载。

方法原型

```
get_presigned_download_url(Bucket, Key, Expired=300, Params={}, Headers={})
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging
import requests

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
```

```
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 生成下载 URL，未限制请求头部和请求参数
url = client.get_presigned_download_url(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(url)

# 生成下载 URL，同时指定响应的 content-disposition 头部，让文件在浏览器另存为，而不是显示
url = client.get_presigned_download_url(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Params={
'response-content-disposition':'attachment; filename=example.xlsx' # 下载时保存为指定的文件
# 除了 response-content-disposition，还支持 response-cache-control、response-content-encoding、response-content-language、
# response-content-type、response-expires 等请求参数，详见下载对象 API，https://cloud.tencent.com/document/product/436/7753
},
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(url)

# 生成下载 URL，同时限制下载速度
url = client.get_presigned_download_url(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Headers={'x-cos-traffic-limit':'819200'}, # 预签名 URL 本身是不包含请求头部的，但请求头部会算入签名，那么使用 URL 时就必须携带请求头部，并且请求头部的值必须是这里指定的值
Expired=300 # 300秒后过期，过期时间请根据自身场景定义
)
print(url)

# 生成下载 URL，只能用于下载 ACL
url = client.get_presigned_download_url(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Params={'acl':''}, # 指定了请求参数，则 URL 中会携带此请求参数，并且请求参数会算入签名，不允
```

```
许使用者修改请求参数的值
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(url)

# 生成下载 URL，请求域名不算入签名，签名后使用者需要修改请求域名时使用
url = client.get_presigned_download_url(
Bucket='examplebucket-1250000000',
Key='exampleobject',
SignHost=False, # 请求域名不算入签名，允许使用者修改请求域名，有一定安全风险
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(url)

# 生成下载 URL，使用临时密钥签名
url = client.get_presigned_download_url(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Params={
'x-cos-security-token': 'string' # 使用永久密钥不需要填入 token，如果使用临时密钥需要填入
}
)
print(url)

# 使用下载 URL
response = requests.get(url)
print(response)
```

全部参数请求示例

```
response = client.get_presigned_download_url(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Expired=300,
Headers={
'header1': 'string'
},
Params={
'param1': 'string',
'param2': 'string'
},
SignHost=True|False
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg **(用户无需进行URL编码)**	String	是
Expired	签名过期时间，单位为秒	Int	否
Params	预签名URL中的请求参数。指定了请求参数，则URL中会携带此请求参数，并且请求参数会算入签名，不允许使用者修改请求参数的值。可以携带的Params和具体的操作相关，例如下载对象可以携带和签入的Params参见 GET Object 中的请求参数描述	Dict	否
Headers	预签名URL中要签入的请求头部。预签名URL本身是不包含请求头部的，但请求头部会算入签名，那么使用URL时就必须携带请求头部，并且请求头部的值必须是这里指定的值。可以签入的Headers和具体的操作相关，例如上传对象可以签入的Headers参见 PUT Object 中的请求头描述	Dict	否
SignHost	请求域名是否算入签名，默认值True，签名后使用者需要修改请求域名时设置为False	Bool	否

返回结果说明

该方法返回值为预签名的下载 URL。

获取签名

功能说明

获取指定操作的签名，常用于移动端的签名分发。

方法原型

```
get_auth(Method, Bucket, Key, Expired=300, Headers={}, Params={})
```

请求示例1：生成上传签名

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
```

```
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

# 生成上传签名，未限制请求头部和请求参数
response = client.get_auth(
Method='PUT',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(response)

# 生成上传签名，同时限制存储类型和上传速度
response = client.get_auth(
Method='PUT',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Headers={
'x-cos-storage-class':'STANDARD_IA',
'x-cos-traffic-limit':'819200' # 约定使用此签名的人必须携带链接限速请求头部，并且请求头部的值必须是这里指定的值
},
Expired=300 # 300秒后过期，过期时间请根据自身场景定义
```

```
)
print(response)

# 生成上传签名，只能上传指定的文件内容
response = client.get_auth(
Method='PUT',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Headers={'Content-MD5':'string'}, # 约定使用此签名的人必须携带 MD5请求头部，并且请求头部的值必须是这里指定的值，这样就限定了文件的内容
Expired=300 # 300秒后过期，过期时间请根据自身场景定义
)
print(response)

# 生成上传签名，只能用于上传 ACL
response = client.get_auth(
Method='PUT',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Params={'acl':''}, # 约定使用此签名的人携带此 ACL 请求参数，这样请求就只能用于上传对象ACL
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(response)

# 生成上传签名，请求域名不算入签名，签名后使用者需要修改请求域名时使用
response = client.get_auth(
Method='PUT',
Bucket='examplebucket-1250000000',
Key='exampleobject',
SignHost=False, # 请求域名不算入签名，允许使用者修改请求域名，有一定安全风险
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(response)
```

请求示例2：生成下载签名

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
```

```
# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)


# 生成下载签名，未限制请求头部和请求参数
response = client.get_auth(
Method='GET',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(response)

# 生成下载签名，同时限制下载速度
response = client.get_auth(
Method='GET',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Headers={'x-cos-traffic-limit':'819200'}, # 约定使用此签名的人必须携带链接限速请求头部，并且请求头部的值必须是这里指定的值
Expired=300 # 300秒后过期，过期时间请根据自身场景定义
)
print(response)

# 生成下载签名，只能用于下载 ACL
response = client.get_auth(
Method='GET',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Params={'acl':''}, # 约定使用此签名的人携带此ACL请求参数，这样请求就只能用于下载对象ACL
```

```
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(response)

# 生成下载签名，请求域名不算入签名，签名后使用者需要修改请求域名时使用
response = client.get_auth(
Method='GET',
Bucket='examplebucket-1250000000',
Key='exampleobject',
SignHost=False, # 请求域名不算入签名，允许使用者修改请求域名，有一定安全风险
Expired=120 # 120秒后过期，过期时间请根据自身场景定义
)
print(response)

# 生成下载签名，使用临时密钥签名
response = client.get_auth(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Params={
'x-cos-security-token': 'string' # 使用永久密钥不需要填入 token，如果使用临时密钥需要填入
}
)
print(response)
```

全部参数请求示例

```
response = client.get_auth(
Method='PUT'|'POST'|'GET'|'DELETE'|'HEAD',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Expired=300,
Headers={
'header1': 'string',
'header2': 'string'
},
Params={
'param1': 'string',
'param2': 'string'
},
SignHost=True|False
)
```

参数说明

参数名称	参数描述	类型	是否必填

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
Method	对应操作的 Method, 可选值为 'PUT'，'POST'，'GET'，'DELETE'，'HEAD'	String	是
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	Bucket 操作填入根路径 `/`，object 操作填入文件的路径 (**用户无需进行URL编码**)	String	是
Expired	签名过期时间，单位为秒	Int	否
Params	签名中要签入的请求参数。使用此签名时必须携带这里指定的请求参数，并且参数的值必须是这里指定的值。可以签入的 Params 和具体的操作相关，例如下载对象可以携带和签入的 Params 参见 GET Object 中的请求参数描述	Dict	否
Headers	签名中要签入的请求头部。使用此签名时必须携带这里指定的请求头部，并且头部的值必须是这里指定的值。可以签入的 Headers 和具体的操作相关，例如上传对象可以签入的 Headers 参见 PUT Object 中的请求头描述	Dict	否
SignHost	请求域名是否算入签名，默认值 True，签名后使用者需要修改请求域名时设置为False	Bool	否

返回结果说明

该方法返回值为对应操作的签名值。

版权所有：腾讯云计算（北京）有限责任公司

恢复归档对象

最近更新时间：2023-05-22 15:25:44

简介

本文档提供关于恢复归档对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
POST Object restore	恢复归档对象	将归档类型的对象取回访问

恢复归档对象

功能说明

将归档类型的对象取回访问（POST Object restore）。

方法原型

```
restore_object(Bucket, Key, RestoreRequest={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
```

```
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.restore_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
RestoreRequest={
'Days': 100,
'CASJobParameters': {
'Tier': 'Standard'
}
}
)
```

全部参数请求示例

```
response = client.restore_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
RestoreRequest={
'Days': 100,
'CASJobParameters': {
'Tier': 'Expedited'|'Standard'|'Bulk'
}
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

参数名称	参数描述	类型	是否必填
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
RestoreRequest	描述取回的临时对象的规则	Dict	是
Days	描述临时对象的过期时间	Int	是
CASJobParameters	描述恢复类型的配置信息	Dict	否
Tier	描述恢复对象的模式，若需恢复归档存储类型的数据，可选值为 Expedited、Standard、Bulk，分别对应快速取回、标准取回以及批量取回这三种模式。若需恢复深度归档存储类型的数据，则可选值为 Standard、Bulk	String	否

返回结果说明

该方法返回值为 None。

版权所有：腾讯云计算（北京）有限责任公司

检索对象内容

最近更新时间：2023-05-22 15:21:49

简介

本文档提供关于检索对象内容相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
SELECT Object content	检索对象内容	从指定对象中检索内容

检索对象内容

功能说明

从指定对象中检索内容（SELECT Object content）。

方法原型

```
select_object_content(Bucket, Key, Expression, ExpressionType, InputSerialization, OutputSerialization, RequestProgress=None, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/docum
```

```
ent/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.select_object_content(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Expression='Select * from COSObject',
ExpressionType='SQL',
InputSerialization={
'CompressionType': 'NONE',
'JSON': {
'Type': 'LINES'
}
},
OutputSerialization={
'CSV': {
'RecordDelimiter': '\n'
}
}
)

# 获取封装在响应结果中的 EventStream 实例
event_stream = response['Payload']

# 一次性获取全部检索结果
# 注意，因为 EventStream 对检索结果的获取是流式的，所以当您再次调用 get_select_result() 方法时将返回空集
result = event_stream.get_select_result()
print(result)
```

全部参数请求示例

```
response = client.select_object_content(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Expression='Select * from COSObject',
```

```
ExpressionType='SQL',
InputSerialization={
'CompressionType': 'GZIP',
'JSON': {
'Type': 'LINES'
}
},
OutputSerialization={
'CSV': {
'RecordDelimiter': '\n'
}
},
RequestProgress={
'Enabled': 'FALSE'
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
Expression	SQL 表达式，代表您需要发起的检索操作	String	是
ExpressionType	表达式类型，该项为扩展项，目前只支持 SQL 表达式，仅支持 SQL 参数	String	是
InputSerialization	描述待检索对象的格式，详情参见 请求示例	Dict	是
OutputSerialization	描述检索结果的输出格式，详情参见 请求示例	Dict	是
RequestProgress	是否需要返回查询进度 QueryProgress 信息，如果选中 COS Select 将周期性返回查询进度	Dict	否

返回结果说明

对象的检索结果，类型为 dict。

```
{
'Payload': EventStream()
```

```
}
```

返回结果中仅存在一条 Key 为 `'Payload'`，Value 为 `EventStream` 实例的 Key-Value 对，对象的检索结果封装在 `EventStream` 实例里，您可以调用其 `next_event()`、`get_select_result()` 以及 `get_select_result_to_file()` 方法获取检索结果。

服务端加密
对象加密

最近更新时间：2023-05-23 15:28:48

简介

本文档提供在上传和下载对象时使用服务端加密的 SDK 示例代码。服务端加密说明请参见 服务端加密概述。

使用 SSE-COS 服务端加密保护数据

由腾讯云对象存储（Cloud Object Storage，COS）托管主密钥和管理数据。COS 会帮助您在数据写入数据中心时自动加密，并在您取用该数据时自动解密。目前支持使用 COS 主密钥对数据进行 AES-256 加密。

请求示例：上传和下载 SSE-COS 加密对象

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 服务端加密必须使用 https
```

```
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token
=token, Scheme=scheme) # 获取配置对象
client = CosS3Client(config)

# 上传 SSE-COS 加密对象，指定加密类型为 AES256即为 SSE-COS 加密
response = client.put_object(
Bucket='examplebucket-125000000',
Key='sdk-sse-cos',
Body='123',
ServerSideEncryption='AES256'
)
print(response['x-cos-server-side-encryption']) # 服务端返回'x-cos-server-side-enc
ryption': 'AES256' 表示是 SSE-COS 对象

# 下载 SSE-COS 加密对象，不需要携带加密头域，由服务端自动解密
response = client.get_object(
Bucket='examplebucket-125000000',
Key='sdk-sse-cos',
)
print(response['x-cos-server-side-encryption']) # 服务端返回'x-cos-server-side-enc
ryption': 'AES256' 表示是 SSE-COS 对象
```

参数说明

参数名	参数描述	类型	是否必填
ServerSideEncryption	服务段加密类型，AES256 表示使用 SSE-COS 加密，cos/kms 表示使用 KMS 加密	String	否

使用 SSE-KMS 服务端加密保护数据

由腾讯云密钥管理系统 KMS 托管密钥的服务端加密方式，可选择使用默认密钥或自建密钥。关于密钥信息，可参见创建 KMS 密钥，关于 SSE-KMS 的更多信息，请参见 服务端加密概述：SSE-KMS。

请求示例：上传和下载 SSE-KMS 加密对象

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos.cos_comm import to_byte

import sys
import os
```

```
import logging
import base64

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 服务端加密必须使用 https

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme) # 获取配置对象
client = CosS3Client(config)

# 上传 SSE-KMS 加密对象，指定加密类型为 cos/kms 即为 SSE-KMS 加密
response = client.put_object(
Bucket='examplebucket-125000000',
Key='sdk-sse-kms',
Body='123',
ServerSideEncryption='cos/kms', # SSE-KMS 加密固定填写 cos/kms
SSEKMSKeyId='kms-key-id', # 填写 KMS 的用户主密钥 CMK，如不填写则使用 COS 默认创建的 CMK
SSEKMSContext=base64.standard_b64encode(to_bytes("{\"test\":\"test\"}")) # 加密上下文，值为 JSON 格式加密上下文键值对的 Base64 编码，可选
)
print(response['x-cos-server-side-encryption']) # 服务端返回'x-cos-server-side-encryption': 'cos/kms' 表示是 SSE-KMS 对象

# 下载 SSE-KMS 加密对象，不需要携带加密头域，由服务端自动解密
response = client.get_object(
Bucket='examplebucket-125000000',
Key='sdk-sse-kms',
)
print(response['x-cos-server-side-encryption']) # 服务端返回'x-cos-server-side-encryption': 'cos/kms' 表示是 SSE-KMS 对象
```

```
# 高级接口上传 SSE-KMS 加密对象，指定加密类型为 cos/kms 即为 SSE-KMS 加密
response = client.upload_file(
Bucket='examplebucket-125000000',
Key='sdk-sse-kms',
LocalFilePath="sdk-sse-kms.local",
ServerSideEncryption='cos/kms') # SSE-KMS 加密固定填写 cos/kms

# 高级接口下载 SSE-KMS 加密对象，不需要携带加密头域，由服务端自动解密
response = client.download_file(
Bucket='examplebucket-125000000',
Key='sdk-sse-kms',
DestFilePath='sdk-sse-kms.local')
```

参数说明

参数名	参数描述	类型	是否必填
ServerSideEncryption	服务段加密配置，AES256 表示使用 SSE-COS 加密，cos/kms 表示使用 KMS 加密	String	否
SSEKMSKeyId	当 ServerSideEncryption='cos/kms'时，可以填写 KMS 的用户主密钥 CMK，如不填写则使用 COS 默认创建的 CMK	String	否
SSEKMSContext	当ServerSideEncryption='cos/kms'时，可以填加密上下文，值为 JSON 格式加密上下文键值对的 Base64 编码	String	否

使用 SSE-C 服务端加密保护数据

加密密钥由用户自己提供，用户在上传对象时，COS 将使用用户提供的加密密钥对用户的数据进行 AES-256 加密。

注意：

- 该加密所运行的服务需要使用 HTTPS 请求。
- customerKey：用户提供的密钥，传入一个32字节的字符串，支持数字、字母、字符的组合，不支持中文。
- 如果上传的源文件调用了该方法，那么在使用 GET（下载）、HEAD（查询）时对源对象操作的时候也要调用该方法。

请求示例：上传、下载、拷贝、取回 SSE-C 加密对象

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos.cos_comm import get_md5, to_byte

import sys
import os
import logging
import base64

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 服务端加密必须使用 https

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme) # 获取配置对象
client = CosS3Client(config)
bucket = 'examplebucket-125000000'
file_name = 'sdk-sse-c'

# 编码和哈希加密密钥
ssec_secret = '00000000000000000000000000000001' # 原始密钥需要是32位
ssec_key = base64.standard_b64encode(to_bytes(ssec_secret)) # 密钥必须 Base64编码
ssec_key_md5 = get_md5(ssec_secret)

# 上传 SSE-C 加密对象
response = client.put_object(
Bucket=bucket, Key=file_name, Body="00000",
SSECustomerAlgorithm='AES256', SSECustomerKey=ssec_key, SSECustomerKeyMD5=ssec_key_md5)
print(response['x-cos-server-side-encryption-customer-algorithm']) # 服务端返回'x-
```

```
cos-server-side-encryption-customer-algorithm': 'AES256' 表示是 SSE-C 对象

# 下载 SSE-C 加密对象
response = client.get_object(
Bucket=bucket, Key=file_name,
SSECustomerAlgorithm='AES256', SSECustomerKey=ssec_key, SSECustomerKeyMD5=ssec_ke
y_md5)
print(response['x-cos-server-side-encryption-customer-algorithm']) # 服务端返回'x-
cos-server-side-encryption-customer-algorithm': 'AES256' 表示是 SSE-C 对象

# HEAD SSE-C 加密对象
response = client.head_object(
Bucket=bucket, Key=file_name,
SSECustomerAlgorithm='AES256', SSECustomerKey=ssec_key, SSECustomerKeyMD5=ssec_ke
y_md5)
print(response['x-cos-server-side-encryption-customer-algorithm']) # 服务端返回'x-
cos-server-side-encryption-customer-algorithm': 'AES256' 表示是 SSE-C 对象

# 高级接口上传 SSE-C 加密对象
response = client.upload_file(
Bucket=bucket, Key=file_name, LocalFilePath="sdk-sse-c.local",
SSECustomerAlgorithm='AES256', SSECustomerKey=ssec_key, SSECustomerKeyMD5=ssec_ke
y_md5)

# 高级接口下载 SSE-C 加密对象
response = client.download_file(
Bucket=bucket, Key=file_name, DestFilePath='sdk-sse-c.local',
SSECustomerAlgorithm='AES256', SSECustomerKey=ssec_key, SSECustomerKeyMD5=ssec_ke
y_md5)

# 拷贝 SSE-C 加密对象，源和目标的加密类型和密钥可以不一样。例子中的源对象是 SSE-C 类型，目标对
象也是 SSE-C 类型。
dest_ssec_secret = '00000000000000000000000000000002' # 密钥需要是32位
dest_ssec_key = base64.standard_b64encode(to_bytes(dest_ssec_secret))
dest_ssec_key_md5 = get_md5(dest_ssec_secret)
copy_source = {'Bucket': bucket, 'Key': file_name, 'Region': region}
response = client.copy_object(
Bucket=bucket, Key='sdk-sse-c-copy', CopySource=copy_source,
SSECustomerAlgorithm='AES256', SSECustomerKey=dest_ssec_key, SSECustomerKeyMD5=de
st_ssec_key_md5,
CopySourceSSECustomerAlgorithm='AES256', CopySourceSSECustomerKey=ssec_key, CopyS
ourceSSECustomerKeyMD5=ssec_key_md5
)

# 高级接口拷贝 SSE-C 加密对象
response = client.copy(
Bucket=bucket, Key='sdk-sse-c-copy', CopySource=copy_source, MAXThread=2,
```

版权所有：腾讯云计算（北京）有限责任公司

```
SSECustomerAlgorithm='AES256', SSECustomerKey=dest_ssec_key, SSECustomerKeyMD5=de
st_ssec_key_md5,
CopySourceSSECustomerAlgorithm='AES256', CopySourceSSECustomerKey=ssec_key, CopyS
ourceSSECustomerKeyMD5=ssec_key_md5)

# 取回 SSE-C 加密对象
response = client.restore_object(
Bucket=bucket, Key=file_name, RestoreRequest={
'Days': 1,
'CASJobParameters': {
'Tier': 'Expedited'
}
},
SSECustomerAlgorithm='AES256', SSECustomerKey=ssec_key, SSECustomerKeyMD5=ssec_ke
y_md5)
```

参数说明

参数名	参数描述	类型	是否必填
SSECustomerAlgorithm	SSE-C 加密算法，目前只支持 AES256	String	否
SSECustomerKey	用户的加密密钥，密钥需要是32位字符串，支持数字、字母、字符的组合，不支持中文，并且使用 Base64 编码	String	SSECustomerAlgorithm 存在时必填
SSECustomerKeyMD5	用户的加密密钥 MD5 值，用于服务端校验密钥完整性	String	SSECustomerAlgorithm 存在时必填
CopySourceSSECustomerAlgorithm	在拷贝接口中，如果源对象使用了 SSE-C 加密，则需要通过此参数指定源对象的加密算法，目前只支持 AES256	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
CopySourceSSECustomerKey	在拷贝接口中，如果源对象使用了 SSE-C 加密，则需要通过此参数指定源对象的加密密钥，密钥需要是32位字符串，并且使用 Base64 编码	String	CopySourceSSECustomerAlgorithm 存在时必填
CopySourceSSECustomerKeyMD5	在拷贝接口中，如果源对象使用了 SSE-C 加密，需要通过此参数指定源对象的加密密钥 MD5 值，用于服务端校验密钥完整性	String	CopySourceSSECustomerAlgorithm 存在时必填

版权所有：腾讯云计算（北京）有限责任公司

存储桶加密

最近更新时间：2023-05-23 15:26:30

简介

本文档提供存储桶加密的 API 概览和 SDK 示例代码。

API	操作名	操作描述
PUT Bucket encryption	设置存储桶加密	设置指定存储桶下的默认加密配置
GET Bucket encryption	查询存储桶加密	查询指定存储桶下的默认加密配置
DELETE Bucket encryption	删除存储桶加密	删除指定存储桶下的默认加密配置

设置存储桶加密

功能说明

用于设置指定存储桶下的默认服务端加密配置。如需执行此接口，必须拥有 PutBucketEncryption 权限。默认情况下，Bucket 的持有者直接拥有权限使用该 API 接口，Bucket 持有者也可以将权限授予其他用户。

方法原型

```
put_bucket_encryption(Bucket, ServerSideEncryptionConfiguration={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循
```

```
最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

config_dict = {
'Rule': [
{
'ApplySideEncryptionConfiguration': {
'SSEAlgorithm': 'AES256',
}
},
]
}
client.put_bucket_encryption(Bucket='examplebucket-1250000000', ServerSideEncryptionConfiguration=config_dict)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
ServerSideEncryptionConfiguration	服务端加密配置	Dict	是

ServerSideEncryptionConfiguration 说明：

参数名	参数描述	类型	是否必填
Rule	服务端加密规则列表，目前只支持一个规则	List	是
ApplySideEncryptionConfiguration	具体的服务加密配置描述	Dict	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
SSEAlgorithm	服务端加密算法，目前桶加密只支持 SSE-COS 类型，使用 AES256 加密算法	String	是

返回结果说明

该方法返回值为 None。

查询存储桶加密

功能说明

用于查询指定存储桶下的默认服务端加密配置。如需执行此接口，必须拥有 GetBucketEncryption 权限。默认情况下，Bucket 的持有者直接拥有权限使用该 API 接口，Bucket 持有者也可以将权限授予其他用户。

方法原型

```
get_bucket_encryption(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
```

```
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_bucket_encryption(Bucket='examplebucket-1250000000')
sse_algorithm = response['Rule'][0]['ApplyServerSideEncryptionByDefault']['SSEAlgorithm']
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

参数名称	参数描述	类型	是否必填
ServerSideEncryptionConfiguration	服务端加密配置	Dict	是

ServerSideEncryptionConfiguration 说明：

参数名	参数描述	类型	是否必填
Rule	服务端加密规则列表，目前只支持一个规则	List	是
ApplySideEncryptionConfiguration	具体的服务加密配置描述	Dict	是
SSEAlgorithm	服务端加密算法，目前桶加密只支持 SSE-COS 类型，使用 AES256 加密算法	String	是

删除存储桶加密

功能说明

用于删除指定存储桶下的默认加密配置。如需执行此接口，必须拥有 DeleteBucketEncryption 权限。默认情况下，Bucket 的持有者直接拥有权限使用该 API 接口，Bucket 持有者也可以将权限授予其他用户。

方法原型

版权所有：腾讯云计算（北京）有限责任公司

```
delete_bucket_encryption(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 ''https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.delete_bucket_encryption(Bucket='examplebucket-1250000000')
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

该方法返回值为 None。

客户端加密

最近更新时间：2023-05-22 15:17:33

简介

本文档提供关于客户端加密的 SDK 示例代码。

功能说明

Python 支持客户端加密，将文件加密后再进行上传，并在下载时进行解密。客户端加密支持对称 AES 和非对称 RSA 加密。
这里的对称和非对称只是用来加密每次生成的随机密钥，对文件数据的加密始终使用 AES256 对称加密。
客户端加密适用于存储敏感数据的客户，客户端加密会牺牲部分上传速度，SDK 内部对于分块上传会使用串行的方式进行上传。

上传加密流程

1. 每次上传一个文件对象前，我们随机生成一个对称加密密钥，随机生成的密钥通过用户的提供的对称或非对称密钥进行加密，将加密后的结果 base64 编码存储在对象的元数据中。
2. 进行文件对象的上传，上传时在内存使用 AES256 算法加密。

下载解密流程

1. 获取文件元数据中加密必要的信息，Base64 解码后使用用户密钥进行解密，得到当时加密数据的密钥。
2. 使用密钥对下载输入流进行使用 AES256 解密，得到解密后的文件输入流。

请求示例1：使用对称 AES256 加密每次生成的随机密钥示例。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos.cos_encryption_client import CosEncryptionClient
from qcloud_cos.crypto import AESProvider
import sys
import os
import logging
```

```
# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048

conf = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token)

# 方式一：通过密钥值初始化加密客户端
# 注意：按照 AES 算法的要求，aes_key_value 需为 base64编码后的结果
aes_provider = AESProvider(aes_key='aes_key_value')

# 方式二：通过密钥路径初始化加密客户端
aes_key_pair = AESProvider(aes_key_path='aes_key_path')

client_for_aes = CosEncryptionClient(conf, aes_provider)

# 上传对象，兼容非加密客户端的 put_object 的所有功能，具体使用可参考 put_object
response = client_for_aes.put_object(
Bucket='examplebucket-1250000000',
Body=b'bytes'|file,
Key='exampleobject',
EnableMD5=False)

# 下载对象，兼容非加密客户端的 get_object 的所有功能，具体使用可参考 get_object
response = client_for_aes.get_object(
Bucket='examplebucket-1250000000',
Key='exampleobject')

# 分块上传，兼容非加密客户端的分块上传，除了最后一个part，每个 part 的大小必须为16字节的整数倍
response = client_for_aes.create_multipart_upload(
Bucket='examplebucket-1250000000',
Key='exampleobject_upload')
```

```
uploadid = response['UploadId']
client_for_aes.upload_part(
Bucket='examplebucket-1250000000',
Key='exampleobject_upload',
Body=b'bytes'|file,
PartNumber=1,
UploadId=uploadid)
response = client_for_aes.list_parts(
Bucket='examplebucket-1250000000',
Key='exampleobject_upload',
UploadId=uploadid)
client_for_aes.complete_multipart_upload(
Bucket='examplebucket-1250000000',
Key='exampleobject_upload',
UploadId=uploadid,
MultipartUpload={'Part':response['Part']})

# 断点续传方式上传对象，`partsize`大小必须为16字节的整数倍
response = client_for_aes.upload_file(
Bucket='test04-123456789',
LocalFilePath='local.txt',
Key=file_name,
PartSize=10,
MAXThread=10
)
```

请求示例2：使用非对称 RSA 加密每次生成的随机密钥示例。

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos.cos_encryption_client import CosEncryptionClient
from qcloud_cos.crypto import RSAProvider
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
```

```
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048

conf = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token)

# 方式一：通过密钥值初始化加密客户端
rsa_key_pair = RSAProvider.get_rsa_key_pair('public_key_value', 'private_key_value')

# 方式二：通过密钥路径初始化加密客户端
rsa_key_pair = RSAProvider.get_rsa_key_pair_path('public_key_path', 'private_key_path')

rsa_provider = RSAProvider(key_pair_info=rsa_key_pair)
client_for_rsa = CosEncryptionClient(conf, rsa_provider)

# 上传对象，兼容非加密客户端的 put_object 的所有功能，具体使用可参考 put_object
response = client_for_rsa.put_object(
Bucket='examplebucket-1250000000',
Body=b'bytes'|file,
Key='exampleobject',
EnableMD5=False)

# 下载对象，兼容非加密客户端的 get_object 的所有功能，具体使用可参考 get_object
response = client_for_rsa.get_object(
Bucket='examplebucket-1250000000',
Key='exampleobject')

# 分块上传，兼容非加密客户端的分块上传，除了最后一个 part，每个 part 的大小必须为16字节的整数倍
response = client_for_rsa.create_multipart_upload(
Bucket='examplebucket-1250000000',
Key='exampleobject_upload')
uploadid = response['UploadId']
client_for_rsa.upload_part(
Bucket='examplebucket-1250000000',
Key='exampleobject_upload',
Body=b'bytes'|file,
PartNumber=1,
```

版权所有：腾讯云计算（北京）有限责任公司

```
UploadId=uploadid)
response = client_for_rsa.list_parts(
Bucket='examplebucket-1250000000',
Key='exampleobject_upload',
UploadId=uploadid)
client_for_rsa.complete_multipart_upload(
Bucket='examplebucket-1250000000',
Key='exampleobject_upload',
UploadId=uploadid,
MultipartUpload={'Part':response['Part']})

# 断点续传方式上传对象，`partsize`大小必须为16字节的整数倍
response = client_for_rsa.upload_file(
Bucket='test04-123456789',
LocalFilePath='local.txt',
Key=file_name,
PartSize=10,
MAXThread=10
)
```

单链接限速

最近更新时间：2023-05-22 15:30:23

简介

本文档提供关于上传下载时对链接进行限速的使用说明。

单链接限速

功能说明

限速值设置范围为**819200 - 838860800**，单位默认为 bit/s，即100KB/s - 100MB/s，如果超出该范围将返回400错误。

示例代码1：上传时对单链接限速

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket

# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填
```

```
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token
=token, Scheme=scheme)
client = CosS3Client(config)

with open('test.bin', 'rb') as fp:
response = client.put_object(
Bucket='examplebucket-1250000000', # Bucket 由 BucketName-APPID 组成
Key='exampleobject',
Body=fp,
TrafficLimit='819200'
)
print(response['ETag'])
```

示例代码2：下载时对单链接限速

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请
在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循
最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/
product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权
遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/docum
ent/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，htt
ps://console.intl.cloud.tencent.com/cos5/bucket

# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/622
4
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用
指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token
=token, Scheme=scheme)
client = CosS3Client(config)
```

```
response = client.get_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
TrafficLimit='819200'
)
response['Body'].get_stream_to_file('exampleobject')
```

参数说明

参数名称	参数描述	类型	是否必填
TrafficLimit	单链接限速的值，单位为bit/s，限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s	String	否

版权所有：腾讯云计算（北京）有限责任公司

异地容灾
版本控制

最近更新时间：2023-03-30 14:43:28

简介

本文档提供关于版本控制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

设置版本控制

功能说明

设置指定存储桶的版本控制功能（PUT Bucket versioning）。

方法原型

```
put_bucket_versioning(Bucket, Status, **kwargs)
```

请求示例

开启版本控制

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，
请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循
```

```
最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_bucket_versioning(
Bucket='examplebucket-1250000000',
Status='Enabled'
)
```

暂停版本控制

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用
```

```
指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token
=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_bucket_versioning(
Bucket='examplebucket-1250000000',
Status='Suspended'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Status	设置存储桶版本控制的状态，可选值为 'Enabled'， 'Suspended'	String	是

返回结果说明

该方法返回值为 None。

查询版本控制

功能说明

查询指定存储桶的版本控制信息（GET Bucket versioning）。

方法原型

```
get_bucket_versioning(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
```

```
# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_bucket_versioning(
Bucket='examplebucket-1250000000',
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

Bucket 版本控制配置，类型为 dict。

```
{
'Status': 'Enabled'|'Suspended'
}
```

参数名称	参数描述	类型
Status	存储桶版本控制的状态，可选值为 'Enabled'，Suspended'	String

存储桶复制

最近更新时间：2023-05-22 11:34:13

简介

本文档提供关于存储桶复制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket replication	设置存储桶复制	设置存储桶的存储桶复制规则
GET Bucket replication	查询存储桶复制	查询存储桶的存储桶复制规则
DELETE Bucket replication	删除存储桶复制	删除存储桶的存储桶复制规则

设置存储桶复制

功能说明

设置指定存储桶的存储桶复制规则（PUT Bucket replication）。

方法原型

```
put_bucket_replication(Bucket, ReplicationConfiguration={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/
```

```
product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_bucket_replication(
Bucket='examplebucket-1250000000',
ReplicationConfiguration={
'Role': 'qcs::cam::uin/100000000001:uin/100000000001',
'Rule': [
{
'ID': 'string',
'Status': 'Enabled',
'Destination': {
'Bucket': 'qcs::cos:ap-shanghai::destinationbucket-1250000000',
'StorageClass': 'STANDARD'
}
}
]
}
)
```

全部参数请求示例

```
response = client.put_bucket_replication(
Bucket='examplebucket-1250000000',
ReplicationConfiguration={
'Role': 'qcs::cam::uin/100000000001:uin/100000000001',
'Rule': [
{
'ID': 'string',
'Status': 'Enabled'|'Disabled',
'Prefix': 'string',
'Destination': {
'Bucket': 'qcs::cos:ap-beijing::destinationbucket-1250000000',
```

```
'StorageClass': 'STANDARD'|'STANDARD_IA'
}
},
{
'ID': 'string',
'Status': 'Enabled'|'Disabled',
'Prefix': 'string',
'Destination': {
'Bucket': 'qcs::cos:ap-beijing::destinationbucket2-1250000000',
'StorageClass': 'STANDARD'|'STANDARD_IA'
}
},
]
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	源存储桶名称，由 BucketName-APPID 构成	String	是
Role	发起者身份标示, 格式为 `qcs::cam::uin/<owneruin>:uin/<subuin>`	String	否
Rule	设置对应的规则，包括 ID，Status，Prefix，Destination	List	是
ID	设置规则的 ID	String	否
Status	设置 Rule 是否启用，可选值为 Enabled 或者 Disabled	String	是
Prefix	设置 Rule 的前缀匹配规则，为空时表示作用存储桶中的所有对象	String	是
Destination	描述目的资源，包括 Bucket 和StorageClass	Dict	是
Bucket	设置存储桶复制的目标存储桶，格式为 `qcs::cos:[region]::[BucketName-APPID]`	String	是
StorageClass	设置目的文件的存储类型，可选值为 `STANDARD` ， `STANDARD_IA`	String	否

返回结果说明

该方法返回值为 None。

查询存储桶复制

功能说明

查询指定存储桶的存储桶复制规则（GET Bucket replication）。

方法原型

```
get_bucket_replication(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_bucket_replication(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

Bucket 存储桶复制配置，类型为 dict。

```
{
'Role': 'qcs::cam::uin/100000000001:uin/100000000001',
'Rule': [
{
'ID': 'string',
'Status': 'Enabled'|'Disabled',
'Prefix': 'string',
'Destination': {
'Bucket': 'qcs::cos:ap-beijing::destinationbucket-1250000000',
'StorageClass': 'STANDARD'|'STANDARD_IA'
}
},
{
'ID': 'string',
'Status': 'Enabled'|'Disabled',
'Prefix': 'string',
'Destination': {
'Bucket': 'qcs::cos:ap-beijing::destinationbucket2-1250000000',
'StorageClass': 'STANDARD'|'STANDARD_IA'
}
},
]
}
```

参数名称	参数描述	类型
Role	发起者身份标示, 格式为 `qcs::cam::uin/<owneruin>:uin/<subuin>`	String
Rule	存储桶复制对应的规则，包括 ID，Status，Prefix，Destination	List
ID	存储桶复制规则的 ID	String
Status	存储桶复制 Rule 是否启用，可选值为 Enabled 或者 Disabled	String
Prefix	存储桶复制 Rule 的前缀匹配规则，为空时表示作用存储桶中的所有对象	String
Destination	描述目的资源，包括 Bucket 和 StorageClass	Dict

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
Bucket	存储桶复制的目标存储桶，格式为 `qcs::cos:[region]::[BucketName-APPID]`	String
StorageClass	目的文件的存储类型，可选值为 `STANDARD` ， `STANDARD_IA`	String

删除存储桶复制

功能说明

删除指定存储桶的存储桶复制规则（DELETE Bucket replication）。

方法原型

```
delete_bucket_replication(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填
```

版权所有：腾讯云计算（北京）有限责任公司

```
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token
=token, Scheme=scheme)
client = CosS3Client(config)

response = client.delete_bucket_replication(
Bucket='examplebucket-1250000000',
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

该方法返回值为 None。

数据管理
生命周期

最近更新时间：2023-05-22 15:09:45

简介

本文档提供关于生命周期的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket lifecycle	设置生命周期	设置存储桶生命周期管理的配置
GET Bucket lifecycle	查询生命周期	查询存储桶生命周期管理的配置
DELETE Bucket lifecycle	删除生命周期	删除存储桶生命周期管理的配置

设置生命周期

功能说明

设置指定存储桶的生命周期配置信息（PUT Bucket lifecycle）。

方法原型

```
put_bucket_lifecycle(Bucket, LifecycleConfiguration={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，
请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
```

```
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_bucket_lifecycle(
Bucket='examplebucket-1250000000',
LifecycleConfiguration={
'Rule': [
{
'ID': 'string', # 设置规则的 ID，例如Rule-1
'Filter': {
'Prefix': '', # 配置前缀为空，桶内所有对象都会执行此规则
},
'Status': 'Enabled', # Enabled 表示启用规则
'Expiration': {
'Days': 200 # 设置对象的当前版本200天后过期删除
},
'Transition': [
{
'Days': 100, # 设置对象的当前版本100天后沉降
'StorageClass': 'Standard_IA' # 沉降为低频存储
},
],
'AbortIncompleteMultipartUpload': {
'DaysAfterInitiation': 7 # 设置7天后回收未合并的分块
}
}
]
}
)
```

全部参数请求示例

```
from qcloud_cos import get_date
response = client.put_bucket_lifecycle(
Bucket='examplebucket-1250000000',
LifecycleConfiguration={
'Rule': [
{
'ID': 'string', # 设置规则的 ID，例如Rule-1
'Filter': {
'Prefix': 'string', # 配置前缀非空，只有符合前缀的对象才会执行此规则
'Tag': [ # 配置标签过滤规则，只有标签匹配的对象才会执行此规则，和前缀二选一
{
'Key': 'string',
'Value': 'string'
}
]
},
'Status': 'Enabled'|'Disabled', # Enabled 表示启用规则，Disabled 表示不启用，二选一
'Expiration': {
'Days': 100, # 设置对象的当前版本100天后过期删除
'Date': get_date(2021, 4, 20) # 设置对象的当前版本在2021年4月20日之后过期删除，和Days二选一
},
'Transition': [
{
'Days': 60, # 设置对象的当前版本60天后沉降
'Date': get_date(2021, 4, 20), # 设置对象的当前版本在2021年4月20日之后沉降，和Days二选一
'StorageClass': 'Archive' # 沉降为归档存储
},
],
'NoncurrentVersionExpiration': {
'NoncurrentDays': 100 # 设置对象的历史版本100天后过期删除
},
'NoncurrentVersionTransition': [
{
'NoncurrentDays': 60, # 设置对象的历史版本60天后沉降
'StorageClass': 'Standard_IA' # 沉降为低频存储
},
],
'AbortIncompleteMultipartUpload': {
'DaysAfterInitiation': 100 # 设置100天后回收未合并的分块
}
}
]
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Rule	设置对应的规则，包括 ID，Filter，Status，Expiration，Transition，NoncurrentVersionExpiration，NoncurrentVersionTransition，AbortIncompleteMultipartUpload	List	是
ID	设置规则的 ID	String	否
Filter	用于描述规则影响的 Object 集合，如需设置 Bucket 中的所有 objects，请设置 Prefix 为空''	Dict	是
Status	设置 Rule 是否启用，可选值为 Enabled 或者 Disabled	Dict	是
Expiration	设置 Object 过期规则，可以指定天数 Days 或者指定日期 Date，Date 的格式必须是 GMT ISO 8601，建议使用 get_date 方法来指定具体的日期	Dict	否
Transition	设置 Object 转换存储类型规则，可以指定天数 Days 或者指定日期 Date，Date 的格式必须是 GMT ISO 8601，建议使用 get_date 方法来指定具体的日期。StorageClass 可选 Standard_IA，Archive，Deep_Archive，可以同时设置多条此类规则	List	否
NoncurrentVersionExpiration	设置非当前版本 Object 过期规则，可以指定天数 NoncurrentDays	Dict	否
NoncurrentVersionTransition	设置非当前版本 Object 转换存储类型规则，可以指定天数 NoncurrentDays，StorageClass 可选 Standard_IA，可以同时设置多条此类规则	List	否
AbortIncompleteMultipartUpload	指明分块上传开始后多少天内必须完成上传	Dict	否

返回结果说明

该方法返回值为 None。

查询生命周期

版权所有：腾讯云计算（北京）有限责任公司

功能说明

查询指定存储桶的生命周期（GET Bucket lifecycle）。

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_bucket_lifecycle(
Bucket='examplebucket-1250000000',
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

版权所有：腾讯云计算（北京）有限责任公司

返回结果说明

Bucket 生命周期配置，类型为 dict。

```
{
'Rule': [
{
'ID': 'string',
'Filter': {
'Prefix': 'string',
'Tag': [
{
'Key': 'string',
'Value': 'string'
}
]
},
'Status': 'string',
'Expiration': {
'Days': 100,
'Date': 'string'
},
'Transition': [
{
'Days': 100,
'Date': 'string',
'StorageClass': 'STANDARD_IA'|'Archive'
},
],
'NoncurrentVersionExpiration': {
'NoncurrentDays': 100
},
'NoncurrentVersionTransition': [
{
'NoncurrentDays': 100,
'StorageClass': 'STANDARD_IA'
},
],
'AbortIncompleteMultipartUpload': {
'DaysAfterInitiation': 100
}
}
]
}
```

参数名称	参数描述	类型

参数名称	参数描述	类型
Rule	对应的规则，包括 ID，Filter，Status，Expiration，Transition，NoncurrentVersionExpiration，NoncurrentVersionTransition，AbortIncompleteMultipartUpload	List
ID	规则的 ID	String
Filter	必用于描述规则影响的 Object 集合	Dict
Status	Rule 是否启用，可选值为 Enabled 或者 Disabled	Dict
Expiration	Object 过期规则，可以指定天数 Days 或者指定日期 Date	Dict
Transition	Object 转换存储类型规则，可以指定天数 Days 或者指定日期 Date，StorageClass 可选 STANDARD_IA，Archive	List
NoncurrentVersionExpiration	非当前版本 Object 过期规则，可以指定天数 NoncurrentDays	Dict
NoncurrentVersionTransition	非当前版本 Object 转换存储类型规则，可以指定天数 NoncurrentDays，StorageClass 可选 STANDARD_IA	List
AbortIncompleteMultipartUpload	分块上传开始后多少天内必须完成上传	Dict

删除生命周期

功能说明

删除指定存储桶的生命周期（DELETE Bucket lifecycle）。

方法原型

```
delete_bucket_lifecycle(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
```

```
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.delete_bucket_lifecycle(
Bucket='examplebucket-1250000000',
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

该方法返回值为 None。

日志管理

最近更新时间：2023-05-23 15:51:17

简介

本文档提供关于日志管理的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket logging	设置日志管理	为源存储桶开启日志记录
GET Bucket logging	查询日志管理	查询源存储桶的日志配置信息

设置日志管理

功能说明

PUT Bucket logging 用于为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中。

方法原型

```
put_bucket_logging(Bucket, BucketLoggingStatus={}, **kwargs):
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权
```

```
# 遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_bucket_logging(
Bucket='examplebucket-1250000000',
BucketLoggingStatus={
'LoggingEnabled': {
'TargetBucket': 'logging-bucket-1250000000',
'TargetPrefix': 'string'
}
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	开启日志功能的源存储桶名称，命名格式为 BucketName-APPID ，详情请参见 命名规范	String	是
TargetBucket	存放日志的目标存储桶名称，命名格式为 BucketName-APPID ，详情请参见 命名规范	String	是
TargetPrefix	日志存放在目标存储桶的指定路径前缀	String	是

返回结果说明

该方法返回值为 None。

查询日志管理

功能说明

GET Bucket logging 用于查询指定存储桶的日志配置信息。

方法原型

```
get_bucket_logging(Bucket, **kwargs):
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_bucket_logging(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名称	参数描述	类型	是否必填

参数名称	参数描述	类型	是否必填
Bucket	存放日志的目标存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String	是

返回结果说明

Bucket 日志管理配置，类型为 dict。

```
{
'LoggingEnabled': {
'TargetBucket': 'logging-bucket-1250000000',
'TargetPrefix': 'string'
}
}
```

参数名称	参数描述	类型
TargetBucket	存放日志的目标存储桶名称，命名格式为 BucketName-APPID ，详情请参见 命名规范	String
TargetPrefix	日志存放在目标存储桶的指定路径前缀	String

存储桶标签

最近更新时间：2023-05-23 15:53:07

简介

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket tagging	设置存储桶标签	为已存在的存储桶设置标签
GET Bucket tagging	查询存储桶标签	查询指定存储桶下已有的存储桶标签
DELETE Bucket tagging	删除存储桶标签	删除指定的存储桶标签

设置存储桶标签

功能说明

PUT Bucket tagging 用于为已存在的存储桶设置标签。

方法原型

```
put_bucket_tagging(Bucket, Tagging={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/
```

```
product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675

region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_bucket_tagging(
Bucket='examplebucket-1250000000',
Tagging={
'TagSet': {
'Tag': [
{
'Key': 'string',
'Value': 'string'
},
]
}
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	设置标签的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String	是
Tag	标签的集合	List	是
Key	标签的 Key，长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String	是
Value	标签的 Value，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String	是

返回结果说明

版权所有：腾讯云计算（北京）有限责任公司

该方法返回值为 None。

查询存储桶标签

功能说明

GET Bucket tagging 用于查询指定存储桶下已有的存储桶标签。

方法原型

```
get_bucket_tagging(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675

region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)
```

```
response = client.get_bucket_tagging(
Bucket='examplebucket-1250000000'
)
```

参数说明 |

参数名称	参数描述	类型	是否必填
Bucket	查询标签的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String	是

返回结果说明

Bucket 标签管理列表，类型为 dict。

```
{
'TagSet': {
'Tag': [
{
'Key': 'string',
'Value': 'string'
},
]
}
}
```

参数名称	参数描述	类型
Tag	标签的集合	List
Key	标签的 Key，长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String
Value	标签的 Value，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String

删除存储桶标签

功能说明

DELETE Bucket tagging 用于删除指定存储桶下已有的存储桶标签。

方法原型

```
delete_bucket_tagging(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675

region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.delete_bucket_tagging(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名称	参数描述	类型	是否必填

参数名称	参数描述	类型	是否必填
Bucket	被删除标签的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String	是

返回结果说明

该方法返回值为 None。

对象标签

最近更新时间：2023-04-06 14:18:21

简介

本文档提供关于对象标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Object tagging	设置对象标签	为已存在的对象设置标签
GET Object tagging	查询对象标签	查询指定对象下已有的对象标签
DELETE Object tagging	删除对象标签	删除指定对象下已有的对象标签

设置对象标签

功能说明

为已存在的对象设置标签（PUT Object tagging）。

方法原型

```
put_object_tagging(self, Bucket, Key, Tagging={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
```

```
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_object_tagging(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Tagging={
'TagSet': {
'Tag': [
{
'Key': "string",
'Value': 'string'
}
]
}
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	Bucket 名称，由 BucketName-APPID 构成，详情请参见 命名规范	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.tencentcos.cn/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
Tag	标签的集合	List	是
Key	标签的 Key，长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
Value	标签的 Value，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String	是

返回结果说明

该方法返回值为 None。

查询对象标签

功能说明

查询指定对象下已有的对象标签（GET Object tagging）。

方法原型

```
get_object_tagging(self, Bucket, Key, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
```

```
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token
=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_object_tagging(
Bucket='examplebucket-1250000000',
Key='exampleobject'
)
print(response)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	Bucket 名称，由 BucketName-APPID 构成，详情请参见 命名规范	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.tencentcos.cn/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是

返回结果说明

Object 标签管理列表，类型为 dict。

```
{
'TagSet': {
'Tag': [
{
'Key': 'string',
'Value': 'string'
}
]
}
}
```

删除对象标签

功能说明

删除制定对象下已有的对象标签（DELETE Object tagging）。

方法原型

```
delete_object_tagging(self, Bucket, Key, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.delete_object_tagging(
Bucket='examplebucket-1250000000',
Key='exampleobject'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	Bucket 名称，由 BucketName-APPID 构成，详情请参见 命名规范	String	是

参数名称	参数描述	类型	是否必填
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.tencentcos.cn/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是

返回结果说明

该方法返回值为 None。

静态网站

最近更新时间：2023-05-23 15:48:58

简介

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket website	设置静态网站	设置存储桶的静态网站配置
GET Bucket website	查询静态网站配置	查询存储桶的静态网站配置
DELETE Bucket website	删除静态网站配置	删除存储桶的静态网站配置

设置静态网站

功能说明

PUT Bucket website 用于为存储桶配置静态网站。

方法原型

```
put_bucket_website(Bucket, WebsiteConfiguration={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/
```

```
product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_bucket_website(
Bucket='bucket',
WebsiteConfiguration={
'IndexDocument': {
'Suffix': 'string'
},
'ErrorDocument': {
'Key': 'string'
},
'RedirectAllRequestsTo': {
'Protocol': 'http'|'https'
},
'RoutingRules': [
{
'Condition': {
'HttpErrorCodeReturnedEquals': 'string',
'KeyPrefixEquals': 'string'
},
'Redirect': {
'HttpRedirectCode': 'string',
'Protocol': 'http'|'https',
'ReplaceKeyPrefixWith': 'string',
'ReplaceKeyWith': 'string'
}
}
]
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	设置静态网站的存储桶，格式为 BucketName-APPID，详情请参见 命名规范	String	是
IndexDocument	设置静态网站首页的配置	Dict	是
Suffix	静态网站首页地址的后缀配置	String	是
ErrorDocument	设置静态网站错误页面配置	Dict	否
Key	错误页面地址	String	否
RedirectAllRequestsTo	设置全站重定向配置	Dict	否
Protocol	全站重定向协议，可选值为 http、https	String	否
RoutingRules	设置静态网站路由规则	List	否
Condition	路由的条件,包括错误码重定向和前缀重定向	Dict	否
HttpErrorCodeReturnedEquals	错误码重定向的条件	String	否
KeyPrefixEquals	前缀重定向的条件	String	否
Redirect	重定向的具体规则	Dict	否
HttpRedirectCode	重定向时的错误码	String	否
Protocol	重定向的协议，可选值为 http、https	String	否
ReplaceKeyPrefixWith	重定向时替换前缀为指定的 key	String	否
ReplaceKeyWith	重定向时替换整个路径为指定的 key	String	否

返回结果说明

该方法返回值为 None。

查询静态网站配置

功能说明

GET Bucket website 用于查询与存储桶关联的静态网站配置信息。

方法原型

版权所有：腾讯云计算（北京）有限责任公司

```
get_bucket_website(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_bucket_website(
Bucket='examplebucket-1250000000'
)
```

参数说明 |

参数名称	参数描述	类型	是否必填
Bucket	查询静态网站配置的存储桶，格式为 BucketName-APPID ，详情请参见命名规范	String	是

返回结果说明

Bucket 静态网站配置，类型为 dict。

```
{
'IndexDocument': {
'Suffix': 'string'
},
'ErrorDocument': {
'Key': 'string'
},
'RedirectAllRequestsTo': {
'Protocol': 'http'|'https'
},
'RoutingRules': [
{
'Condition': {
'HttpErrorCodeReturnedEquals': 'string',
'KeyPrefixEquals': 'string'
},
'Redirect': {
'HttpRedirectCode': 'string',
'Protocol': 'http'|'https',
'ReplaceKeyPrefixWith': 'string',
'ReplaceKeyWith': 'string'
}
}
]
}
```

参数名称	参数描述	类型
IndexDocument	设置静态网站首页的配置	Dict
Suffix	静态网站首页地址的后缀配置	String
ErrorDocument	设置静态网站错误页面配置	Dict
Key	错误页面地址	String
RedirectAllRequestsTo	设置全站重定向配置	Dict
Protocol	全站重定向协议，可选值为 http、https	String
RoutingRules	设置静态网站路由规则	List
Condition	路由的条件，包括错误码重定向和前缀重定向	Dict

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型
HttpErrorCodeReturnedEquals	错误码重定向的条件	String
KeyPrefixEquals	前缀重定向的条件	String
Redirect	重定向的具体规则	Dict
HttpRedirectCode	重定向时的错误码	String
Protocol	重定向的协议，可选值为 http、https	String
ReplaceKeyPrefixWith	重定向时替换前缀为指定的 key	String
ReplaceKeyWith	重定向时替换整个路径为指定的 key	String

删除静态网站配置

功能说明

DELETE Bucket website 用于删除存储桶中的静态网站配置。

方法原型

```
delete_bucket_website(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/docum
```

```
ent/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.delete_bucket_website(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	被删除静态网站配置的存储桶，格式为 BucketName-APPID ，详情请参见命名规范	String	是

返回结果说明

该方法返回值为 None。

清单

最近更新时间：2023-05-23 15:41:08

简介

本文档提供关于清单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket inventory	设置清单任务	设置存储桶的清单任务
GET Bucket inventory	查询清单任务	查询存储桶的清单任务
DELETE Bucket inventory	删除清单任务	删除存储桶的清单任务

设置清单任务

功能说明

PUT Bucket inventory 用于在存储桶中创建清单任务。

方法原型

```
put_bucket_inventory(Bucket, Id, InventoryConfiguration={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/
```

```
product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_bucket_inventory(
Bucket='examplebucket-1250000000',
Id='string',
InventoryConfiguration={
'Destination': {
'COSBucketDestination': {
'AccountId': '100000000001',
'Bucket': 'qcs::cos:ap-guangzhou::examplebucket-1250000000',
'Format': 'CSV',
'Prefix': 'string',
'Encryption': {
'SSECOS': {}
}
}
},
'IsEnabled': 'true'|'false',
'Filter': {
'Prefix': 'string'
},
'IncludedObjectVersions':'All'|'Current',
'OptionalFields': {
'Field': [
'Size',
'LastModifiedDate',
'ETag',
'StorageClass',
'IsMultipartUploaded',
'ReplicationStatus'
]
},
'Schedule': {
```

```
'Frequency': 'Daily'|'Weekly'
}
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	设置清单任务的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String	是
Id	清单任务的名称，合法字符：a-z，A-Z，0-9，-，_，.	String	是
Destination	清单任务投递的目的信息	Dict	是
COSBucketDestination	清单任务结果投递的目的存储桶信息	Dict	是
AccountId	目的存储桶的账号信息	String	否
Bucket	目的存储桶的名称	String	是
Format	清单结果文件的格式，可选值 CSV	String	是
Prefix	目的存储桶投递路径的前缀	String	否
Encryption	目的存储桶投递文件的加密信息，可选值 SSECOS	Dict	否
IsEnabled	清单任务是否启用的标识，可选值 true、flase	String	是
Filter	清单分析对象的过滤条件	Dict	否
Prefix	清单分析对象的过滤前缀	String	否
IncludedObjectVersions	包含多版本的情况，可选值 All、Current	String	是
OptionalFields	清单可选的字段	Dict	否
Field	字段名，包括 Size、LastModifiedDate、ETag、StorageClass、IsMultipartUploaded、ReplicationStatus	List	否
Schedule	清单任务执行的计划	Dict	是
Frequency	清单任务执行的频率，可选值 Daily、Weekly	String	是

返回结果说明

该方法返回值为 None。

版权所有：腾讯云计算（北京）有限责任公司

错误码说明

该请求可能会发生的一些常见的特殊错误如下：

错误码	描述	状态码
InvalidArgument	不合法的参数值	HTTP 400 Bad Request
TooManyConfigurations	清单数量已经达到1000条的上限	HTTP 400 Bad Request
AccessDenied	未授权的访问。您可能不具备访问该存储桶的权限	HTTP 403 Forbidden

查询清单任务

功能说明

GET Bucket inventory 用于查询存储桶中用户的清单任务信息。

方法原型

```
get_bucket_inventory(Bucket, Id, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/43
```

```
6/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_bucket_inventory(
Bucket='examplebucket-1250000000',
Id='string'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	查询清单任务的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String	是
Id	清单任务的名称，合法字符：a-z，A-Z，0-9，-，_，.	String	是

返回结果说明

Bucket 清单任务配置，类型为 dict。

```
{
'Id': 'string',
'Destination': {
'COSBucketDestination': {
'AccountId': '100000000001',
'Bucket': 'qcs::cos:ap-guangzhou::examplebucket-1250000000',
'Format': 'CSV',
'Prefix': 'string',
'Encryption': {
'SSECOS': {}
}
}
},
'IsEnabled': 'true'|'false',
'Filter': {
'Prefix': 'string'
},
'IncludedObjectVersions':'All'|'Current',
'OptionalFields': {
```

```
'Field': [
'Size',
'LastModifiedDate',
'ETag',
'StorageClass',
'IsMultipartUploaded',
'ReplicationStatus'
]
},
'Schedule': {
'Frequency': 'Daily'|'Weekly'
}
}
```

参数名称	参数描述	类型
Id	清单任务的名称，合法字符：a-z，A-Z，0-9，-，_，.	String
Destination	清单任务投递的目的信息	Dict
COSBucketDestination	清单任务结果投递的目的存储桶信息	Dict
AccountId	目的存储桶的账号信息	String
Bucket	目标存储桶的名称	String
Format	清单结果文件的格式，可选值 CSV	String
Prefix	目的存储桶投递路径的前缀	String
Encryption	目的存储桶投递文件的加密信息，可选值 SSECOS	Dict
IsEnabled	清单任务是否启用的标识，可选值 true、flase	String
Filter	清单分析对象的过滤条件	Dict
Prefix	清单分析对象的过滤前缀	String
IncludedObjectVersions	包含多版本的情况，可选值 All、Current	String
OptionalFields	清单可选的字段	Dict
Field	字段名，包括 Size、LastModifiedDate、ETag、StorageClass、IsMultipartUploaded、ReplicationStatus	List
Schedule	清单任务执行的计划	Dict
Frequency	清单任务执行的频率，可选值 Daily、Weekly	String

版权所有：腾讯云计算（北京）有限责任公司

删除清单任务

功能说明

DELETE Bucket inventory 用于删除存储桶中指定的清单任务。

方法原型

```
delete_bucket_inventory(Bucket, Id, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.delete_bucket_inventory(
Bucket='examplebucket-1250000000',
Id='string'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	被删除清单任务的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String	是
Id	清单任务的名称，合法字符：a-z，A-Z，0-9，-，_，.	String	是

返回结果说明

该方法返回值为 None。

访问管理
跨域访问

最近更新时间：2023-05-22 15:11:54

简介

本文档提供关于跨域访问的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域名访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域名访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域名访问配置信息

设置跨域配置

功能说明

设置指定存储桶的跨域名访问配置信息（PUT Bucket cors）。

方法原型

```
put_bucket_cors(Bucket, CORSConfiguration={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
```

```
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_bucket_cors(
Bucket='examplebucket-1250000000',
CORSConfiguration={
'CORSRule': [
{
'ID': 'string',
'MaxAgeSeconds': 100,
'AllowedOrigin': [
'string',
],
'AllowedMethod': [
'string',
],
'AllowedHeader': [
'string',
],
'ExposeHeader': [
'string',
]
}
]
},
)
```

参数说明

参数名称	参数描述	类型	是否必填

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
CORSRule	设置对应的跨域规则，包括 ID，MaxAgeSeconds，AllowedOrigin，AllowedMethod，AllowedHeader，ExposeHeader	List	是
ID	设置规则的 ID	String	否
MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	Int	否
AllowedOrigin	设置允许的访问来源，如 `"http://cloud.tencent.com"`，支持通配符 `*`	Dict	是
AllowedMethod	设置允许的方法，如 GET，PUT，HEAD，POST，DELETE	Dict	是
AllowedHeader	设置请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 `*`	Dict	否
ExposeHeader	设置浏览器可以接收到的来自服务器端的自定义头部信息	Dict	否

返回结果说明

该方法返回值为 None。

查询跨域配置

功能说明

查询指定存储桶的跨域名访问配置信息（GET Bucket cors）。

方法原型

```
get_bucket_cors(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
```

```
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_bucket_cors(
Bucket='examplebucket-1250000000',
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

存储桶跨域配置，类型为 dict。

```
{
'CORSRule': [
{
'ID': 'string',
'MaxAgeSeconds': 100,
'AllowedOrigin': [
'string',
],
'AllowedMethod': [
'string',
```

```
],
'AllowedHeader': [
'string',
],
'ExposeHeader': [
'string',
],
}
]
}
```

参数名称	参数描述	类型
CORSRule	跨域规则，包括 ID，MaxAgeSeconds，AllowedOrigin，AllowedMethod，AllowedHeader，ExposeHeader	List
ID	规则的 ID	String
MaxAgeSeconds	OPTIONS 请求得到结果的有效期	String
AllowedOrigin	允许的访问来源，如 `"http://cloud.tencent.com"`，支持通配符 `*`	Dict
AllowedMethod	允许的方法，如 GET，PUT，HEAD，POST，DELETE	Dict
AllowedHeader	请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 `*`	Dict
ExposeHeader	浏览器可以接收到的来自服务器端的自定义头部信息	Dict

删除跨域配置

功能说明

删除指定存储桶的跨域名访问配置（DELETE Bucket cors）。

方法原型

```
delete_bucket_cors(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
```

```
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.delete_bucket_cors(
Bucket='examplebucket-1250000000',
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

该方法返回值为 None。

存储桶策略

最近更新时间：2023-05-22 15:15:23

简介

本文档提供关于存储桶策略的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket policy	设置存储桶策略	设置指定存储桶的权限策略
GET Bucket policy	查询存储桶策略	查询指定存储桶的权限策略
DELETE Bucket policy	删除存储桶策略	删除指定存储桶的权限策略

设置存储桶策略

功能说明

设置指定存储桶的权限策略（PUT Bucket policy）。

方法原型

```
put_bucket_policy(Bucket, Policy, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/
```

```
product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_bucket_policy(
Bucket='examplebucket-1250000000',
Policy={
"Statement": [
{
"Principal": {
"qcs": [
"qcs::cam::uin/100000000001:uin/100000000011"
]
},
"Effect": "allow",
"Action": [
"name/cos:GetBucket"
],
"Resource": [
"qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
],
"condition": {
"ip_equal": {
"qcs:ip": "10.121.2.10/24"
}
}
}
],
"version": "2.0"
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Statement	描述一条或多条权限的详细信息	List	是
Principal	描述策略授权的实体，详情请参见 访问策略语言概述	Dict	是
Version	策略语法版本，默认为2.0	String	是
Effect	有 allow（允许）和 deny（显式拒绝）两种情况	String	是
Action	此处是指 COS API，根据需求指定一个或者一序列操作的组合或所有操作(*)，例如 action 为 name/cos:GetService，**请注意区分英文大小写**	List	是
Resource	授权操作的具体数据，可以是任意资源、指定路径前缀的资源、指定绝对路径的资源或它们的组合	List	是
Condition	约束条件，可以不填，具体说明请参见 condition 说明	List	否

返回结果说明

该方法返回值为 None。

查询存储桶策略

功能说明

查询指定存储桶的权限策略（GET Bucket policy）。

方法原型

```
get_bucket_policy(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging
import json

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
```

```
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_bucket_policy(
Bucket='examplebucket-1250000000',
)
policy = json.loads(response['Policy'])
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

Bucket 存储桶策略规则，类型为 dict，该 dict 包含一条 key-value 对，其中 key 是字符串 `'Policy'`，value 是表示具体存储桶策略规则的 JSON 字符串，可以使用 `json.loads()` 将其转换为 dict：

```
{
"Statement": [
{
"Principal": {
"qcs": [
"qcs::cam::uin/100000000001:uin/100000000011"
]
},
```

```
"Effect": "allow",
"Action": [
"name/cos:GetBucket"
],
"Resource": [
"qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
],
"condition": {
"ip_equal": {
"qcs:ip": "10.121.2.10/24"
}
}
}
],
"version": "2.0"
}
```

参数名称	参数描述	类型
Statement	描述一条或多条权限的详细信息	List
Principal	描述策略授权的实体，详情请参见 访问策略语言概述	Dict
Version	策略语法版本，默认为2.0	String
Effect	有 allow（允许）和 deny（显式拒绝）两种情况	String
Action	此处是指 COS API，根据需求指定一个或者一序列操作的组合或所有操作(*)，例如 action 为 name/cos:GetService，**请注意区分英文大小写**	List
Resource	授权操作的具体数据，可以是任意资源、指定路径前缀的资源、指定绝对路径的资源或它们的组合	List
Condition	约束条件，可以不填，具体说明请参见 condition 说明	List

删除存储桶策略

功能说明

删除指定存储桶的存储桶策略（DELETE Bucket policy）。

方法原型

```
delete_bucket_policy(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.delete_bucket_policy(
Bucket='examplebucket-1250000000',
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

该方法返回值为 None。

自定义域名

最近更新时间：2023-04-07 14:26:19

简介

本文档提供关于自定义域名的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket domain	设置自定义域名	设置存储桶的自定义域名信息
GET Bucket domain	查询自定义域名	查询存储桶的自定义域名信息
DELETE Bucket domain	删除自定义域名	删除存储桶的自定义域名

设置自定义域名

功能说明

PUT Bucket domain 用于为存储桶配置自定义域名。

方法原型

```
put_bucket_domain(Bucket, DomainConfiguration={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
```

```
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_bucket_domain(
Bucket='bucket',
DomainConfiguration={
'DomainRule': [
{
'Name': 'example.com',
'Type': 'REST'|'WEBSITE'|'ACCELERATE',
'Status': 'ENABLED'|'DISABLED',
'ForcedReplacement': 'CNAME'|'TXT'
},
]
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	设置自定义域名的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String	是
DomainRule	自定义域名的集合	List	是
Name	自定义域名名称	String	是
Type	绑定的源站类型，可选值有 REST、WEBSITE	String	是
Status	域名上线状态，可选值有 ENABLED、DISABLED	String	是
ForcedReplacement	强制覆盖已存在的配置，可选值有 CNAME、TXT	String	否

版权所有：腾讯云计算（北京）有限责任公司

返回结果说明

该方法返回值为 None。

返回错误码说明

该请求可能会发生的一些常见的特殊错误如下：

状态码	说明
HTTP 409 Conflict	该域名记录已存在，且请求中没有设置强制覆盖。或者该域名记录不存在，且请求中设置了强制覆盖
HTTP 451 Unavailable For Legal Reasons	该域名是中国境内域名，并且没有备案

查询自定义域名

功能说明

GET Bucket domain 用于查询存储桶的自定义域名信息。

方法原型

```
get_bucket_domain(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
```

```
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_bucket_domain(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	查询自定义域名的存储桶，格式为 BucketName-APPID ，详情请参见 命名规范	String	是

返回结果说明

Bucket 自定义域名配置，类型为 dict。

```
{
'x-cos-domain-txt-verification': 'string',
'DomainRule': [
{
'Name': 'example.com',
'Type': 'REST'|'WEBSITE'|'ACCELERATE',
'Status': 'ENABLED'|'DISABLED',
'ForcedReplacement': 'CNAME'|'TXT'
},
]
}
```

参数名称	参数描述	类型
x-cos-domain-txt-verification	域名校验信息，该字段是一个 MD5 校验值，原串格式为：`cos[Region][BucketName-APPID][BucketCreateTime]` ，其中 Region 为存储桶所在地域，BucketCreateTime 为存储桶 GMT 创建时间	String

参数名称	参数描述	类型
DomainRule	自定义域名的集合	List
Name	自定义域名名称	String
Type	绑定的源站类型，可选值有 REST、WEBSITE	String
Status	域名上线状态，可选值有 ENABLED、DISABLED	String
ForcedReplacement	强制覆盖已存在的配置，可选值有 CNAME、TXT	String

删除自定义域名

功能说明

DELETE Bucket domain 用于删除指定存储桶下已有的自定义域名配置。

方法原型

```
delete_bucket_domain(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/43
```

```
6/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.delete_bucket_domain(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	被删除自定义域名的存储桶，格式为 BucketName-APPID ，详情请参见命名规范	String	是

返回结果说明

该方法返回值为 None。

访问控制

最近更新时间：2023-05-23 15:38:02

简介

本文档提供关于存储桶、对象的访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

存储桶 ACL

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶的访问权限控制列表
GET Bucket acl	查询存储桶 ACL	查询指定存储桶的访问控制列表

对象 ACL

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表
GET Object acl	查询对象 ACL	查询对象的访问控制列表

存储桶 ACL

设置存储桶 ACL

功能说明

设置指定存储桶的访问权限控制列表（PUT Bucket acl）。AccessControlPolicy 参数与其它权限参数是互斥的，无法同时指定。

方法原型

```
put_bucket_acl(Bucket, AccessControlPolicy={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
```

```
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_bucket_acl(
Bucket='examplebucket-1250000000',
ACL='public-read'
)
```

全部参数请求示例

```
response = client.put_bucket_acl(
Bucket='examplebucket-1250000000',
ACL='private'|'public-read'|'public-read-write',
GrantFullControl='id="100000000002"',
GrantRead='id="100000000003",id="100000000004"',
GrantWrite='id="100000000005"',
AccessControlPolicy={
'AccessControlList': {
'Grant': [
{
'Grantee': {
```

```
'DisplayName': 'qcs::cam::uin/100000000002:uin/100000000002',
'Type': 'CanonicalUser'|'Group',
'ID': 'qcs::cam::uin/100000000002:uin/100000000002', # Type为CanonicalUser时必须填写ID
'URI': 'http://cam.qcloud.com/groups/global/AllUsers' # Type为Group时必须填写URI
},
'Permission': 'FULL_CONTROL'|'WRITE'|'READ'
},
]
},
'Owner': {
'DisplayName': 'qcs::cam::uin/100000000001:uin/100000000001',
'ID': 'qcs::cam::uin/100000000001:uin/100000000001' # 必须是桶 Owner 的 ID
}
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
ACL	设置存储桶的 ACL，例如 'private'，'public-read'，'public-read-write'，详情请参见 ACL 概述	String	否
GrantFullControl	赋予指定账户对存储桶的读写权限，格式为 `id="OwnerUin"` ，可使用半角逗号（,）分隔多组被授权者，如 id="100000000001",id="100000000002"	String	否
GrantRead	赋予指定账户对存储桶的读权限，格式为 `id="OwnerUin"` ，可使用半角逗号（,）分隔多组被授权者，如 id="100000000001",id="100000000002"	String	否
GrantWrite	赋予指定账户对存储桶的写权限，格式为 `id="OwnerUin"` ，可使用半角逗号（,）分隔多组被授权者，如 id="100000000001",id="100000000002"	String	否
AccessControlPolicy	赋予指定账户对存储桶的访问权限，与其它权限参数是互斥的，无法同时指定	Dict	否

AccessControlPolicy 参数说明：

参数名称	参数描述	类型	是否必填
Owner	存储桶拥有者的信息，包括 DisplayName 和 ID	Dict	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
AccessControlList	存储桶权限被授予者的信息，包括 Grant 列表	Dict	是

Owner 参数说明：

参数名称	参数描述	类型	是否必填
ID	Type 为 CanonicalUser 时，对应权限授予者的 ID，格式为 qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]，如 qcs::cam::uin/100000000001:uin/100000000001	String	是
DisplayName	权限被授予者的名称，可以不填，或者和ID的值保持一致	String	否

AccessControlList 参数说明：

参数名称	参数描述	类型	是否必填
Grant	存储桶权限被授予者的信息，包括 Grantee 和 Permission	List	是
Grantee	权限被授予者的信息，包括 DisplayName，Type，ID 和 URI	Dict	是
Type	权限被授予者的类型，类型为 CanonicalUser 或者 Group	String	是
ID	Type 为 CanonicalUser 时，对应权限授予者的 ID，格式为 qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]，如 qcs::cam::uin/100000000001:uin/100000000001	String	当 Grantee 的 Type 指定为 CanonicalUser 时必选
URI	Type 为 Group 时，填入预设用户组的 URI，例如 http://cam.qcloud.com/groups/global/AllUsers， 详情请参见 ACL 概述	String	当 Grantee 的 Type 指定为 Group 时必选
Permission	指明授予被授权者的存储桶权限，可选值有 FULL_CONTROL，WRITE，READ，分别对应读写权限、写权限、读权限	String	是

返回结果说明

该方法返回值为 None。

查询存储桶 ACL

功能说明

查询指定存储桶的访问权限控制列表（GET Bucket acl）。

版权所有：腾讯云计算（北京）有限责任公司

方法原型

```
get_bucket_acl(Bucket, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_bucket_acl(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

版权所有：腾讯云计算（北京）有限责任公司

返回结果说明

Bucket ACL 信息，类型为 dict。

```
{
'Owner': {
'DisplayName': 'qcs::cam::uin/100000000001:uin/100000000001',
'ID': 'qcs::cam::uin/100000000001:uin/100000000001'
},
'AccessControlList': {
'Grant': [
{
'Grantee': {
'DisplayName': 'qcs::cam::uin/100000000002:uin/100000000002',
'Type': 'CanonicalUser'|'Group',
'ID': 'qcs::cam::uin/100000000002:uin/100000000002',
'URI': 'http://cam.qcloud.com/groups/global/AllUsers'
},
'Permission': 'FULL_CONTROL'|'WRITE'|'READ'
},
]
}
}
```

参数名称	参数描述	类型
Owner	存储桶拥有者的信息，包括 DisplayName 和 ID，详细说明见PUT Bucket acl	Dict
AccessControlList	存储桶权限被授予者的信息，包括 Grant 列表，详细说明见PUT Bucket acl	Dict

对象 ACL

设置对象 ACL

功能说明

设置存储桶中某个对象的访问控制列表（PUT Object acl）。AccessControlPolicy 参数与其它权限参数是互斥的，无法同时指定。

方法原型

```
put_object_acl(Bucket, Key, AccessControlPolicy={}, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.put_object_acl(
Bucket='examplebucket-1250000000',
Key='exampleobject',
ACL='public-read'
)
```

全部参数请求示例

```
response = client.put_object_acl(
Bucket='examplebucket-1250000000',
Key='exampleobject',
ACL='default'|'private'|'public-read',
GrantFullControl='id="100000000003"',
GrantRead='id="100000000003",id="100000000004"',
AccessControlPolicy={
```

版权所有：腾讯云计算（北京）有限责任公司

```
'AccessControlList': {
'Grant': [
{
'Grantee': {
'DisplayName': 'qcs::cam::uin/100000000002:uin/100000000002',
'Type': 'CanonicalUser'|'Group',
'ID': 'qcs::cam::uin/100000000002:uin/100000000002', # Type为CanonicalUser时必须填写ID
'URI': 'http://cam.qcloud.com/groups/global/AllUsers' # Type为Group时必须填写URI
},
'Permission': 'FULL_CONTROL'|'READ'
},
]
},
'Owner': {
'DisplayName': 'qcs::cam::uin/100000000001:uin/100000000001',
'ID': 'qcs::cam::uin/100000000001:uin/100000000001'
}
}
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
ACL	设置对象的 ACL，例如 'default'，'private'，'public-read'，详情请参见 ACL 概述	String	否
GrantFullControl	赋予指定账户对对象的所有权限，格式为 `id="OwnerUin"` ，可使用半角逗号（,）分隔多组被授权者，例如id="100000000001",id="100000000002"	String	否
GrantRead	赋予指定账户对对象的读权限，格式为 `id="OwnerUin"` ，可使用半角逗号（,）分隔多组被授权者，例如 id="100000000001",id="100000000002"	String	否
AccessControlPolicy	赋予指定账户对对象的访问权限，与其它权限参数是互斥的，无法同时指定	Dict	否

版权所有：腾讯云计算（北京）有限责任公司

AccessControlPolicy 参数说明：

参数名称	参数描述	类型	是否必填
Owner	存储桶拥有者的信息，包括 DisplayName 和 ID	Dict	是
AccessControlList	存储桶权限被授予者的信息，包括 Grant 列表	Dict	是

Owner 参数说明：

参数名称	参数描述	类型	是否必填
ID	Type 为 CanonicalUser 时，对应权限授予者的 ID，格式为 qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]，如 qcs::cam::uin/100000000001:uin/100000000001	String	是
DisplayName	权限被授予者的名称，可以不填，或者和ID的值保持一致	String	否

AccessControlList 参数说明：

参数名称	参数描述	类型	是否必填
Grant	存储桶权限被授予者的信息，包括 Grantee 和 Permission	List	是
Grantee	权限被授予者的信息，包括 DisplayName，Type，ID 和 URI	Dict	是
Type	权限被授予者的类型，类型为 CanonicalUser 或者 Group	String	是
ID	Type 为 CanonicalUser 时，对应权限授予者的 ID，格式为 qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]，如 qcs::cam::uin/100000000001:uin/100000000001	String	当 Grantee 的 Type 指定为 CanonicalUser 时必选
URI	Type 为 Group 时，填入预设用户组的 URI，例如 http://cam.qcloud.com/groups/global/AllUsers， 详情请参见 ACL 概述	String	当 Grantee 的 Type 指定为 Group 时必选
Permission	指明授予被授权者的权限信息，可选值有 FULL_CONTROL，READ，分别对应所有权限、读权限	String	是

返回结果说明

该方法返回值为 None。

查询对象 ACL

功能说明

查询对象的访问控制列表（GET Object acl）。

方法原型

```
get_object_acl(Bucket, Key, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# 1. 设置用户属性，包括 secret_id，secret_key，region 等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

response = client.get_object_acl(
Bucket='examplebucket-1250000000',
Key='exampleobject'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是

返回结果说明

Object ACL 信息，类型为 dict：

```
{
'Owner': {
'DisplayName': 'qcs::cam::uin/100000000001:uin/100000000001',
'ID': 'qcs::cam::uin/100000000001:uin/100000000001'
},
'AccessControlList': {
'Grant': [
{
'Grantee': {
'DisplayName': 'qcs::cam::uin/100000000002:uin/100000000002',
'Type': 'CanonicalUser'|'Group',
'ID': 'qcs::cam::uin/100000000002:uin/100000000002',
'URI': 'http://cam.qcloud.com/groups/global/AllUsers'
},
'Permission': 'FULL_CONTROL'|'READ'
},
]
}
}
```

参数名称	参数描述	类型
Owner	对象拥有者的信息，包括 DisplayName 和 ID，详细说明见PUT Object acl	Dict
AccessControlList	对象权限被授予者的信息，包括 Grant 列表，详细说明见PUT Object acl	Dict

版权所有：腾讯云计算（北京）有限责任公司

防盗链

最近更新时间：2023-05-23 15:31:43

简介

本文档提供关于存储桶 Referer 白名单或者黑名单的 API 概览以及 SDK 示例代码。

> 注意：
> 需要 COS PYTHON SDK v5.1.9.7 及以上版本。

API	操作名	操作描述
PUT Bucket referer	设置存储桶 Referer	设置存储桶 Referer 白名单或者黑名单
GET Bucket referer	查询存储桶 Referer	查询存储桶 Referer 白名单或者黑名单

设置存储桶 Referer

功能说明

设置指定存储桶的 Referer 白名单或者黑名单（PUT Bucket referer）。

方法原型

```
put_bucket_referer(Bucket, RefererConfiguration, **kwargs)
```

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import os
import logging

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
```

```
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在 CosConfig 中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = os.environ['COS_SECRET_ID'] # 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
secret_key = os.environ['COS_SECRET_KEY'] # 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://intl.cloud.tencent.com/document/product/598/32675
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的 region 可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS 支持的所有 region 列表参见 https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入 token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见 https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)

referer_config = {
'Status': 'Enabled',
'RefererType': 'White-List',
'EmptyReferConfiguration': 'Allow',
'DomainList': {
'Domain': [
'*.qq.com',
'*.qcloud.com'
]
}
}
response = client.put_bucket_referer(
Bucket='examplebucket-1250000000',
RefererConfiguration=referer_config
)
```

参数说明

参数名	参数描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String
RefererConfiguration	存储桶 Referer 配置	Dict

RefererConfiguration 说明：

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Status	是否开启防盗链，枚举值：Enabled、Disabled	String	是
RefererType	防盗链类型，枚举值：Black-List、White-List	String	是
DomainList	生效域名的列表	Dict	是
Domain	生效域名，支持带端口和 IP、支持通配符*, 支持多条	List	是
EmptyReferConfiguration	是否允许空 Refer 访问，枚举值： Allow、Deny	String	否

返回结果说明

该方法返回值为 None。

查询存储桶 Referer

功能说明

查询指定存储桶 Referer 白名单或者黑名单（GET Bucket referer）。

方法原型

```
get_bucket_referer(Bucket, **kwargs)
```

请求示例

```
response = client.get_bucket_referer(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名	参数描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String

返回结果说明

返回存储桶的 Referer 配置，参见 RefererConfiguration 说明。

删除存储桶 Referer

功能说明

删除指定存储桶 Referer 白名单或者黑名单（DELETE Bucket referer）。

方法原型

```
delete_bucket_referer(Bucket, **kwargs)
```

请求示例

```
response = client.delete_bucket_referer(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名	参数描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID，详情请参见 命名规范	String

返回结果说明

该方法返回值为 None。

内容识别
图片二维码

最近更新时间：2022-10-09 16:25:23

简介

本文档提供关于图片二维码相关 API 概览以及 SDK 示例代码。

API	操作描述
二维码识别	二维码识别功能可识别图片中有效二维码的位置及内容，输出图像中二维码包含的文本信息（每个二维码对应的 URL 或文本），并可对识别出的二维码添加马赛克

二维码识别

二维码识别功能可识别图片中有效二维码的位置及内容，输出图像中二维码包含的文本信息（每个二维码对应的 URL 或文本），并可对识别出的二维码添加马赛克。

上传时识别二维码

功能说明

图片上传时识别二维码的请求包与 COS 简单上传文件接口一致，只需在请求包头部增加图片处理参数 Pic-Operations。

示例代码

```
# 先创建 cos client
example_object = 'example_object.jpg'
with open(example_object, 'rb') as fp:
opts = '{"is_pic_info":1,"rules":[{"fileid":"format.jpg","rule":"QRcode/cover/
0"}]}'
response,data = client.ci_put_object_from_local_file_and_get_qrcode(
Bucket='example-bucket-123456789',
LocalFilePath=example_object,
Key='example_key',
EnableMD5=False,
PicOperations=opts
)
```

版权所有：腾讯云计算（北京）有限责任公司

```
# 查看响应信息，可根据需要读指定数据
print(response,data)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
LocalFilePath	图片路径	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
EnableMD5	是否需要 SDK 计算 Content-MD5，默认关闭，打开后将增加上传耗时	Bool	否
PicOperations	万象图片处理参数，请参见二维码识别	String	是

云上数据识别二维码

功能说明

对云上图片进行二维码识别，返回二维码识别结果。

示例代码

```
# 先创建 cos client
response,data = client.ci_get_object_qrcode(
Bucket='example_bucket-123456789',
Key='example_object',
Cover=0
)
# 查看响应信息，可根据需要读指定数据
print(response,data)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
Cover	万象图片处理二维码覆盖开关，请参见二维码识别	Int	是

设置访问域名（CDN/全球加速）

最近更新时间：2022-04-18 14:30:06

简介

本文档提供关于如何使用 PYTHON SDK 设置 访问域名。

通过 COS 默认域名访问

通过 COS 默认域名访问时，SDK 会以 **{bucket-appid}.cos.{region}.myqcloud.com** 的域名形式访问 COS。

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging
# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = 'SecretId' # 替换为用户的 SecretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
secret_key = 'SecretKey' # 替换为用户的 SecretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
region = 'ap-beijing' # 替换为用户的 region，已创建桶归属的region可以在控制台查看，https://console.intl.cloud.tencent.com/cos5/bucket
# COS支持的所有region列表参见https://intl.cloud.tencent.com/document/product/436/6224
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Scheme=scheme)
client = CosS3Client(config)
```

通过 COS 全球加速域名访问

通过 COS 全球加速域名访问时，SDK 会以 **{bucket-appid}.cos.accelerate.myqcloud.com** 的域名形式访问 COS，region 不会出现在访问域名中。

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging
# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = 'SecretId' # 替换为用户的 SecretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
secret_key = 'SecretKey' # 替换为用户的 SecretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
region = None # 通过Endpoint初始化不需要配置region
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填
endpoint = 'cos.accelerate.myqcloud.com' # 替换为用户的 endpoint 或者 cos全局加速域名，如果使用桶的全球加速域名，需要先开启桶的全球加速功能，请参见https://intl.cloud.tencent.com/document/product/436/33406
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Endpoint=endpoint, Scheme=scheme)
client = CosS3Client(config)
```

通过 COS 自定义域名访问

通过自定义域名访问时，SDK 会使用配置的**用户域名**直接访问 COS，bucket 和 region 都不会出现在访问域名中。

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging
# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = 'SecretId' # 替换为用户的 SecretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
secret_key = 'SecretKey' # 替换为用户的 SecretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
region = None # 通过自定义域名初始化不需要配置region
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
```

```
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填
domain = 'user-define.example.com' # 用户自定义域名，需要先开启桶的自定义域名，具体请参见 https://intl.cloud.tencent.com/document/product/436/31507
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Domain=domain, Scheme=scheme)
client = CosS3Client(config)
```

通过 CDN 默认域名访问

通过 CDN 默认域名访问时，SDK 会以 **{bucket-appid}.file.mycloud.com** 的域名形式访问 **CDN**，由 CDN 服务回源访问 COS。

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging
# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = 'SecretId' # 替换为用户的 SecretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
secret_key = 'SecretKey' # 替换为用户的 SecretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
region = None # 通过Endpoint初始化不需要配置region
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填
endpoint = 'file.mycloud.com' # 替换为用户的 CDN 默认加速域名，需要开通 CDN 加速配置，参见https://intl.cloud.tencent.com/document/product/436/18670
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Endpoint=endpoint, Scheme=scheme)
client = CosS3Client(config)
```

通过 CDN 自定义域名访问

通过 CDN 自定义域名访问时，SDK 会使用配置的**用户域名**直接访问 **CDN**，bucket 和 region 都不会出现在访问域名中，由 CDN 服务回源访问 COS。

请求示例

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging
# 正常情况日志级别使用INFO，需要定位时可以修改为DEBUG，此时SDK会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
# 1. 设置用户属性，包括 secret_id，secret_key，region等。Appid 已在CosConfig中移除，请在参数 Bucket 中带上 Appid。Bucket 由 BucketName-Appid 组成
secret_id = 'SecretId' # 替换为用户的 SecretId，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
secret_key = 'SecretKey' # 替换为用户的 SecretKey，请登录访问管理控制台进行查看和管理，https://console.intl.cloud.tencent.com/cam/capi
region = None # 通过自定义域名初始化不需要配置region
token = None # 如果使用永久密钥不需要填入token，如果使用临时密钥需要填入，临时密钥生成和使用指引参见https://intl.cloud.tencent.com/document/product/436/14048
scheme = 'https' # 指定使用 http/https 协议来访问 COS，默认为 https，可不填
domain = 'user-define.example-cdn.com' # 用户自定义 CDN 域名，需要开通 CDN 自定义域名加速，参见https://intl.cloud.tencent.com/document/product/436/18670
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token, Domain=domain, Scheme=scheme)
client = CosS3Client(config)
```

版权所有：腾讯云计算（北京）有限责任公司

异常处理

最近更新时间：2022-01-05 14:31:57

简介

COS XML Python SDK 操作成功会返回一个 dict 或者 None。若调用 SDK 接口请求 COS 服务失败，系统将抛出 CosClientError（客户端异常）或者 CosServiceError （服务端异常）。

- CosClientError 是由于客户端无法和 COS 服务端正常进行交互所引起。如客户端无法连接到服务端，无法解析服务端返回的数据，读取本地文件发生 IO 异常等。
- CosServiceError 是客户端和 COS 服务端交互正常，但操作 COS 资源失败。如客户端访问一个不存在的存储桶，删除一个不存在的对象，没有权限进行某个操作等。

客户端异常

CosClientError 一般指如 timeout 引起的客户端错误，用户捕获后可以选择重试或其它操作。

服务端异常

CosServiceError 提供服务端返回的具体信息，包含了服务端返回的状态码、requestid 和出错明细等。捕获异常后，建议对整个异常进行打印，异常包含了必须的排查因素。以下是异常成员变量的描述以及异常捕获示例。

成员	描述	类型
request_id	请求 ID，用于唯一标识一个请求，对于排查问题十分重要	string
status_code	response 的 status 状态码，更多详情请参见 COS 错误码	string
error_code	请求失败时 body 返回的 Error Code，更多详情请参见 COS 错误码	string
error_msg	请求失败时 body 返回的 Error Message，更多详情请参见 COS 错误码	string

异常捕获示例

```
from qcloud_cos import CosServiceError
except CosServiceError as e:
e.get_origin_msg() # 获取原始错误信息，格式为XML
e.get_digest_msg() # 获取处理过的错误信息，格式为dict
```

版权所有：腾讯云计算（北京）有限责任公司

```
e.get_status_code() # 获取 http 错误码（如4XX, 5XX)
e.get_error_code() # 获取 COS 定义的错误码
e.get_error_msg() # 获取 COS 错误码的具体描述
e.get_trace_id() # 获取请求的 trace_id
e.get_request_id() # 获取请求的 request_id
e.get_resource_location() # 获取 URL 地址
```

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。

   COS 自助诊断工具

   点击自助诊断

   输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**。
4. 稍侯片刻，便能看到相应的智能诊断结果。

版权所有：腾讯云计算（北京）有限责任公司

图片处理
图片持久化处理

最近更新时间：2022-06-09 16:05:02

简介

本文档提供关于图片持久化处理相关的 API 概览以及 SDK 示例代码。

API	操作描述
图片持久化处理	对象存储（Cloud Object Storage，COS）提供的上传时处理功能可以帮助使用者在上传时实现图片处理。此外能够对已存储在 COS 的图片进行相应处理操作，并将结果存入到 COS

上传时处理

功能说明

数据万象提供的上传时处理功能可以帮助使用者在上传时实现图片处理。您只需要在请求包头部中加入 Pic-Operations 项并设置好相应参数，就可在图片上传时实现相应的图片处理，并可将原图和处理结果存入到 COS。目前支持20M以内文件处理。

方法原型

```
ci_put_object(self, Bucket, Body, Key, EnableMD5=False, **kwargs)
```

请求示例

```
with open('local.jpg', 'rb') as fp:
response, data = client.ci_put_object(
Bucket='examplebucket-1250000000',
Body=fp,
Key=ci_file_name,
# pic operation json struct
PicOperations='{"is_pic_info":1,"rules":[{"fileid":"format.png","rule":"imageView
2/format/png"}]}')
```

全部参数请求示例

```
response = client.ci_put_object(
Bucket='examplebucket-1250000000',
Body=b'bytes'|file,
Key='exampleobject',
EnableMD5=False|True,
ACL='private'|'public-read', # 请慎用此参数,否则会达到1000条 ACL 上限
GrantFullControl='string',
GrantRead='string',
StorageClass='STANDARD'|'STANDARD_IA'|'ARCHIVE',
Expires='string',
CacheControl='string',
ContentType='string',
ContentDisposition='string',
ContentEncoding='string',
ContentLanguage='string',
ContentLength='123',
ContentMD5='string',
Metadata={
'x-cos-meta-key1': 'value1',
'x-cos-meta-key2': 'value2'
},
TrafficLimit='1048576'
PicOperations='{"is_pic_info":1,"rules":[{"fileid":"format.png","rule":"imageView2/format/png"}]}'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Body	上传对象的内容，可以为文件流或字节流	file/bytes	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
PicOperations	万象图片处理参数，请参见 图片持久化处理	String	是
EnableMD5	是否需要 SDK 计算 Content-MD5，默认关闭，打开后将增加上传耗时	Bool	否
ACL	设置对象的 ACL，例如 'private'，'public-read'	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
GrantFullControl	赋予被授权者所有的权限，格式为 `id="OwnerUin"`	String	否
GrantRead	赋予被授权者读的权限，格式为 `id="OwnerUin"`	String	否
StorageClass	设置对象的存储类型，STANDARD，STANDARD_IA，ARCHIVE。默认值为 STANDARD。更多存储类型，请参见 存储类型概述	String	否
Expires	设置 Expires	String	否
CacheControl	缓存策略，设置 Cache-Control	String	否
ContentType	内容类型，设置 Content-Type	String	否
ContentDisposition	对象名称，设置 Content-Disposition	String	否
ContentEncoding	编码格式，设置 Content-Encoding	String	否
ContentLanguage	语言类型，设置 Content-Language	String	否
ContentLength	设置传输长度	String	否
ContentMD5	设置上传对象的 MD5 值用于校验	String	否
Metadata	用户自定义的对象元数据， 必须以 x-cos-meta 开头，否则会被忽略	Dict	否
TrafficLimit	单链接限速的值，单位为bit/s，限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s	String	否

PicOperations 为 json 格式的字符串，具体参数如下：

参数名称	描述	类型	是否必选
is_pic_info	是否返回原图信息，0不返回原图信息，1返回原图信息，默认为0	Int	否
rules	处理规则，一条规则对应一个处理结果（目前支持五条规则），不填则不进行图片处理	Array	否

rules（json 数组）中每一项具体参数如下：

参数名称	描述	类型	是否必选

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型	是否必选
bucket	存储结果的目标存储桶名称，格式为 BucketName-APPID，如果不指定的话默认保存到当前存储桶	String	否
fileid	处理结果的文件路径名称，如以 `/` 开头，则存入指定文件夹中，否则，存入原图文件存储的同目录	String	是
rule	处理参数，参见数据万象图片处理 API。若按指定样式处理，则以 `style/` 开头，后加样式名，如样式名为 `test` ，则 rule 字段为 `style/test`	String	是

返回结果说明

获取原图片和处理信息，类型为 dict：

```
{
'OriginalInfo': {
'Key': 'local.jpg',
'Location': 'examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/local.jpg',
'ETag': '"aff1b996bcc63a0f0259df0de2fa989f38c5ce7e"',
'ImageInfo': {
'Format': 'JPEG',
'Width': '300',
'Height': '168',
'Quality': '74',
'Ave': '0x1a3451',
'Orientation': '0'
}
},
'ProcessResults': {
'Object': {
'Key': 'format.png',
'Location': 'examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/format.png',
'Format': 'png',
'Width': '300',
'Height': '168',
'Size': '77063',
'Quality': '74',
'ETag': '"a07dc5bcfa238e7d23d2d5884da4ac328aaaa9c6"'
}
}
}
```

响应包体具体数据内容如下：

参数名称	类型	描述
UploadResult	Container	原图信息

UploadResult 节点内容：

参数名称	类型	描述
OriginalInfo	Container	原图信息
ProcessResults	Container	图片处理结果

OriginalInfo 节点内容：

节点名称	类型	描述
Key	String	原图文件名
Location	String	图片路径
ImageInfo	Container	原图图片信息
ETag	String	原图 ETag 信息（若处理结果图覆盖原图则为结果图 ETag 信息）

ImageInfo 节点内容：

节点名称	类型	描述
Format	String	格式
Width	Int	图片宽度
Height	Int	图片高度
Quality	Int	图片质量
Ave	String	图片主色调
Orientation	Int	图片旋转角度

ProcessResults 节点内容：

节点名称	类型	描述

版权所有：腾讯云计算（北京）有限责任公司

节点名称	类型	描述
Object	Container	每一个图片处理结果

Object 节点内容：

节点名称	类型	描述
Key	String	文件名
Location	String	图片路径
Format	String	图片格式
Width	Int	图片宽度
Height	Int	图片高度
Size	Int	图片大小
Quality	Int	图片质量
ETag	String	处理结果图 ETag 信息

云上数据处理

功能说明

图片持久化处理 API 能够对已存储在 COS 的图片进行相应处理操作，并将结果存入到 COS。

方法原型

```
ci_image_process(self, Bucket, Key, **kwargs)
```

请求示例

```
response, data = client.ci_image_process(
Bucket='examplebucket-1250000000',
Key=ci_file_name,
# pic operation json struct
PicOperations='{"is_pic_info":1,"rules":[{"fileid":"format.png","rule":"imageView
2/format/png"}]}')
```

全部参数请求示例

```
response, data = client.ci_image_process(
Bucket='examplebucket-1250000000',
Key=ci_file_name,
# pic operation json struct
PicOperations='{"is_pic_info":1,"rules":[{"fileid":"format.png","rule":"imageView
2/format/png"}]}')
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
PicOperations	万象图片处理参数，请参见 图片持久化处理	String	是

PicOperations 为 json 格式的字符串，具体参数如下：

参数名称	描述	类型	是否必选
is_pic_info	是否返回原图信息，0不返回原图信息，1返回原图信息，默认为0	Int	否
rules	处理规则，一条规则对应一个处理结果（目前支持五条规则），不填则不进行图片处理	Array	否

rules（json 数组）中每一项具体参数如下：

参数名称	描述	类型	是否必选
bucket	存储结果的目标存储桶名称，格式为 BucketName-APPID，如果不指定的话默认保存到当前存储桶	String	否
fileid	处理结果的文件路径名称，如以 `/` 开头，则存入指定文件夹中，否则，存入原图文件存储的同目录	String	是
rule	处理参数，可参见图片处理 API。若按指定样式处理，则以 `style/` 开头，后加样式名，如样式名为 `test`，则 rule 字段为 `style/test`	String	是

版权所有：腾讯云计算（北京）有限责任公司

返回结果说明

获取对象的元信息，类型为 dict：

```
{
'OriginalInfo': {
'Key': 'local.jpg',
'Location': 'examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/local.jpg',
'ETag': '"aff1b996bcc63a0f0259df0de2fa989f38c5ce7e"',
'ImageInfo': {
'Format': 'JPEG',
'Width': '300',
'Height': '168',
'Quality': '74',
'Ave': '0x1a3451',
'Orientation': '0'
}
},
'ProcessResults': {
'Object': {
'Key': 'format.png',
'Location': 'examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/format.png',
'Format': 'png',
'Width': '300',
'Height': '168',
'Size': '77063',
'Quality': '74',
'ETag': '"a07dc5bcfa238e7d23d2d5884da4ac328aaaa9c6"'
}
}
}
```

响应包体具体数据内容如下：

参数名称	类型	描述
UploadResult	Container	原图信息

UploadResult 节点内容：

参数名称	类型	描述
OriginalInfo	Container	原图信息
ProcessResults	Container	图片处理结果

版权所有：腾讯云计算（北京）有限责任公司

OriginalInfo 节点内容：

节点名称	类型	描述
Key	String	原图文件名
Location	String	图片路径
ImageInfo	Container	原图图片信息
ETag	String	原图 ETag 信息（若处理结果图覆盖原图则为结果图 ETag 信息）

ImageInfo 节点内容：

节点名称	类型	描述
Format	String	格式
Width	Int	图片宽度
Height	Int	图片高度
Quality	Int	图片质量
Ave	String	图片主色调
Orientation	Int	图片旋转角度

ProcessResults 节点内容：

节点名称	类型	描述
Object	Container	每一个图片处理结果

Object 节点内容：

节点名称	类型	描述
Key	String	文件名
Location	String	图片路径
Format	String	图片格式
Width	Int	图片宽度

版权所有：腾讯云计算（北京）有限责任公司

节点名称	类型	描述
Height	Int	图片高度
Size	Int	图片大小
Quality	Int	图片质量
ETag	String	处理结果图 ETag 信息

版权所有：腾讯云计算（北京）有限责任公司

图片高级压缩

最近更新时间：2022-10-10 14:31:53

简介

本文档提供关于图片高级压缩的 API 概览以及 SDK 示例代码。

API	操作描述
图片高级压缩	对指定存储桶下的图片进行压缩

图片高级压缩

功能说明

通过数据万象 imageMogr2 接口提供图片高级压缩功能。

方法原型

```
ci_download_compress_image(self, Bucket, Key, DestImagePath, CompressType, **kwargs)
```

请求示例

```
# TPG 压缩
response = client.ci_download_compress_image(
Bucket='examplebucket-1250000000',
Key='sample.png',
DestImagePath='sample.tpg',
CompressType='tpg'
)
# HEIF 压缩
response = client.ci_download_compress_image(
Bucket='examplebucket-1250000000',
Key='sample.png',
DestImagePath='sample.heif',
CompressType='heif'
)
```

参数说明

版权所有：腾讯云计算（北京）有限责任公司

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，格式为 BucketName-APPID ，详情请参见 命名规范	String	是
Key	对象Key，长度不超过128字节, 支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String	是
DestImagePath	压缩图片保存至本地的目的路径	String	是
CompressType	压缩格式，支持 TPG 或 HEIF	String	是

返回结果说明

该方法返回值为请求响应头部。

盲水印

最近更新时间：2022-06-21 15:02:10

简介

本文档提供关于盲水印的 API 概览以及 SDK 示例代码。

API	操作描述
盲水印	对本地图片添加或提取盲水印并上传至存储桶

盲水印

功能说明

盲水印功能基于腾讯云数据万象，是一种全新的水印模式。

方法原型

```
def ci_put_object_from_local_file(self, Bucket, LocalFilePath, Key, EnableMD5=False, **kwargs)
```

请求示例

```
# 添加盲水印
watermark_url = 'http://{bucket}.cos.{region}.myqcloud.com/watermark.png'.format(bucket='examplebucket-1250000000', region=region)
watermark_url_base64 = bytes.decode(base64.b64encode(str.encode(watermark_url)))
print(watermark_url_base64)
response, data = client.ci_put_object_from_local_file(
Bucket='examplebucket-1250000000',
LocalFilePath='sample.png',
Key="sample.png",
# pic operation json struct
PicOperations='{"is_pic_info":1,"rules":[{"fileid": "format.png","rule": "watermark/3/type/1/image/' +
watermark_url_base64 + '" }]}'
)
# 提取盲水印
sample_url = 'http://{bucket}.cos.{region}.myqcloud.com/sample.png'.format(bucket='examplebucket-1250000000', region=region)
```

```
sample_url_base64 = bytes.decode(base64.b64encode(str.encode(sample_url)))
response, data = client.ci_put_object_from_local_file(
Bucket='examplebucket-1250000000',
LocalFilePath='format.png',
Key="format.png",
# pic operation json struct
PicOperations='{"is_pic_info":1,"rules":[{"fileid": "watermark.png","rule": "wate
rmark/4/type/1/image/' +
sample_url_base64 + '" }]}'
)
```

参数说明

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称，格式为 BucketName-APPID ，详情请参见 命名规范	String	是
LocalFilePath	需要处理的本地图片路径	String	是
Key	对象键，长度不超过128字节, 支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String	是
EnableMD5	开启对象上传的 MD5 校验	Bool	否

返回结果说明

该方法返回值为请求响应头部和内容。

React Native SDK 快速入门

最近更新时间：2023-03-30 14:22:42

相关资源

- SDK 源码下载
- 示例 Demo
- SDK 更新日志
- NPM 地址

准备工作

1. 您需要一个纯 React Native 项目或 React Native 原生混合项目，这个应用可以是您现有的工程，也可以是您新建的一个空的工程。
2. React Native 版本要求：0.69.7 及以上

第一步：SDK 介绍

react-native-cos-sdk 目前兼容支持 iOS、Android，是通过 React Native 桥接原生 Android 和 iOS 的 COS SDK 实现。

第二步：集成 SDK

1. 运行此命令：

使用 npm:

```
npm install --save react-native-cos-sdk
```

或者使用 yarn:

```
yarn add react-native-cos-sdk
```

2. 在您的代码中，您可以使用 import 进行导入，然后开始使用：

```
import Cos from 'react-native-cos-sdk';
```

第三步：开始使用

注意：

- 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

1. 初始化密钥

实现获取临时密钥

调用 Cos 的 initWithSessionCredentialCallback 方法，实现请求临时密钥并返回结果的过程。

```
import Cos from 'react-native-cos-sdk';

Cos.initWithSessionCredentialCallback(async () => {
// 首先从您的临时密钥服务器获取包含了密钥信息的响应，例如：
// 临时密钥服务器 url
let stsUrl = "http://stsservice.com/sts";
const response = await fetch(stsUrl);
// 然后解析响应，获取临时密钥信息
const responseJson = await response.json();
const credentials = responseJson.credentials;
const startTime = responseJson.startTime;
const expiredTime = responseJson.expiredTime;
const sessionCredentials = {
tmpSecretId: credentials.tmpSecretId,
tmpSecretKey: credentials.tmpSecretKey,
startTime: startTime,
expiredTime: expiredTime,
sessionToken: credentials.sessionToken
};
console.log(sessionCredentials);
// 最后返回临时密钥信息对象
return sessionCredentials;
})
```

使用永久密钥进行本地调试

您可以使用腾讯云的永久密钥来进行开发阶段的本地调试。**由于该方式存在泄漏密钥的风险，请务必在上线前替换为临时密钥的方式。**

```
import Cos from 'react-native-cos-sdk';

let SECRET_ID = "SECRETID"; //永久密钥 secretId
let SECRET_KEY = "SECRETKEY"; //永久密钥 secretKey

Cos.initWithPlainSecret(
SECRET_ID,
SECRET_KEY
)
```

2. 注册 COS 服务

```
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
let region = "COS_REGION";
// 创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
let serviceConfig = {
region: region,
isDebuggable: true,
isHttps: true,
};
// 注册默认 COS Service
let cosService = await Cos.registerDefaultService(serviceConfig);
// 获取默认 COS Service
let cosService1 = Cos.getDefaultService();

// 创建 TransferConfig 对象，根据需要修改默认的配置参数
// TransferConfig 可以设置智能分块阈值 默认对大于或等于2M的文件自动进行分块上传，可以通过如下代码修改分块阈值
let transferConfig = {
forceSimpleUpload: false,
enableVerification: true,
divisionForUpload: 2097152, // 设置大于等于 2M 的文件进行分块上传
sliceSizeForUpload: 1048576, //设置默认分块大小为 1M
};
// 注册默认 COS TransferManger
let cosTransferManger = await Cos.registerDefaultTransferManger(serviceConfig, transferConfig);
// 获取默认 COS TransferManger
let cosTransferManger1 = Cos.getDefaultTransferManger();

// 也可以通过 registerService 和 registerTransferManger 注册其他实例， 用于后续调用
```

```
// 一般用 region 作为注册的 key
let newRegion = "NEW_COS_REGION";
serviceConfig.region = newRegion;
let cosServiceNew = await Cos.registerService(newRegion, serviceConfig);
let cosTransferMangerNew = await Cos.registerTransferManger(newRegion, serviceCon
fig, transferConfig);
// 通过 key 获取 COS Service 和 COS TransferManger
let cosServiceNew1 = Cos.getService(newRegion);
let cosTransferMangerNew1 = Cos.getTransferManger(newRegion);
```

注意：获取 COS Service 和 COS TransferManger 之前必须先要进行注册，否则会报错。

例如可以通过封装类似下面的方法控制获取前必须注册的流程

```
const SERVICE_CONFIG = {
region: "COS_REGION",
isDebuggable: true,
}

export async function getDefaultService(): Promise<CosService> {
if(Cos.hasDefaultService()){
return Cos.getDefaultService()
} else {
//注册默认 service
return await Cos.registerDefaultService(SERVICE_CONFIG)
}
}
```

参数说明

CosXmlServiceConfig 用于配置 COS 服务，其主要成员说明如下：

参数名称	描述	类型	默认值	支持平台
region	存储桶地域 地域和访问域名	String	null	Android 和iOS
connectionTimeout	连接超时时间（单位是毫秒）	Int	Android(15000) iOS(30000)	Android 和iOS
socketTimeout	读写超时时间（单位是毫秒）	Int	30000	Android
isHttps	是否使用 https 协议	Bool	true	Android 和iOS
host	设置除了 GetService 请求外的 host	String	null	Android 和iOS

参数名称	描述	类型	默认值	支持平台
hostFormat	设置 host 的格式化字符串，sdk 会将 ${bucket} 替换为真正的 bucket，${region} 替换为真正的 region 例如将 hostFormat 设置为 ${bucket}.${region}.tencent.com，并且您的存储桶和地域分别为 bucket-1250000000 和 ap-shanghai，那么最终的请求地址为 `bucket-1250000000.ap-shanghai.tencent.com` • 注意，这个设置不会影响 GetService 请求	String	null	Android
port	设置请求的端口	Int	null	Android
isDebuggable	是否是 debug 模式（debug 模式会打印 debug 日志）	Bool	false	Android
signInUrl	是否将签名放在 URL 中，默认放在 Header 中	Bool	false	Android
userAgent	ua 拓展参数	String	null	Android 和iOS
dnsCache	是否开启 DNS 解析缓存，开启后，将 DNS 解析的结果缓存在本地，当系统 DNS 解析失败后，会使用本地缓存的 DNS 结果	Bool	true	Android
accelerate	是否使用全球加速域名	Bool	false	Android 和iOS

TransferConfig 用于配置 COS 上传服务，其主要成员说明如下：

参数名称	描述	类型	默认值	支持平台
divisionForUpload	设置启用分块上传的最小对象大小	Int	2097152	Android和iOS
sliceSizeForUpload	设置分块上传时的分块大小	Int	1048576	Android和iOS
enableVerification	分片上传时是否整体校验	Bool	true	Android和iOS
forceSimpleUpload	是否强制使用简单上传	Bool	false	Android

版权所有：腾讯云计算（北京）有限责任公司

第四步：访问 COS 服务

上传对象

```
// 获取 CosTransferManger
let cosTransferManger: CosTransferManger = Cos.getDefaultTransferManger();
//let cosTransferManger: CosTransferManger = Cos.getTransferManger(newRegion);
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
let cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
let srcPath = "本地文件的路径"; //本地文件的路径
//若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传；否则，赋值 undefined
let _uploadId = undefined;

// 上传成功回调
let successCallBack = (header?: object) => {
// todo 上传成功后的逻辑
};
//上传失败回调
let failCallBack = (clientError?: CosXmlClientError, serviceError?: CosXmlService
Error) => {
// todo 上传失败后的逻辑
if (clientError) {
console.log(clientError);
}
if (serviceError) {
console.log(serviceError);
}
};
//上传状态回调，可以查看任务过程
let stateCallBack = (state: TransferState) => {
// todo notify transfer state
};
//上传进度回调
let progressCallBack = (complete: number, target: number) => {
// todo Do something to update progress...
};
//初始化分块完成回调
let initMultipleUploadCallBack = (bucket: string, cosKey: string, uploadId: strin
g) => {
//用于下次续传上传的 uploadId
_uploadId = uploadId;
};

//开始上传
```

```
let transferTask:TransferTask = await cosTransferManger.upload(
bucket,
cosPath,
srcPath,
{
uploadId: _uploadId,
resultListener: {
successCallBack: successCallBack,
failCallBack: failCallBack
},
stateCallback: stateCallBack,
progressCallback: progressCallBack,
initMultipleUploadCallback: initMultipleUploadCallBack,
}
);
//暂停任务
transferTask.pause();
//恢复任务
transferTask.resume();
//取消任务
transferTask.cancel();
```

下载对象

```
// 高级下载接口支持断点续传，所以会在下载前先发起 HEAD 请求获取文件信息。
// 如果您使用的是临时密钥或者使用子账号访问，请确保权限列表中包含 HeadObject 的权限。

// CosTransferManger 支持断点下载，您只需要保证 bucket、cosPath、savePath
// 参数一致，SDK 便会从上次已经下载的位置继续下载。

// 获取 CosTransferManger
let cosTransferManger: CosTransferManger = Cos.getDefaultTransferManger();
//let cosTransferManger: CosTransferManger = Cos.getTransferManger(newRegion);
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
let cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
let downliadPath = "本地文件的路径"; //保存到本地文件的路径

// 下载成功回调
let successCallBack = (header?: object) => {
// todo 下载成功后的逻辑
};
//下载失败回调
let failCallBack = (clientError?: CosXmlClientError, serviceError?: CosXmlService
Error) => {
```

```
// todo 下载失败后的逻辑
if (clientError) {
console.log(clientError);
}
if (serviceError) {
console.log(serviceError);
}
};
//下载状态回调, 可以查看任务过程
let stateCallBack = (state: TransferState) => {
// todo notify transfer state
};
//下载进度回调
let progressCallBack = (complete: number, target: number) => {
// todo Do something to download progress...
};

//开始下载
let transferTask:TransferTask = = await cosTransferManger.download(
bucket,
cosPath,
downliadPath,
{
resultListener: {
successCallBack: successCallBack,
failCallBack: failCallBack
},
stateCallback: stateCallBack,
progressCallback: progressCallBack
}
);
//暂停任务
transferTask.pause();
//恢复任务
transferTask.resume();
//取消任务
transferTask.cancel();
```

React Native SDK 常见问题

最近更新时间：2023-03-30 14:22:42

客户端网络正常，但是通过 HTTP 访问 COS 非常慢，或者报错 Connection reset，该如何处理？

部分区域的运营商可能会对 COS 的域名进行劫持，因此尽量通过 HTTPS 来访问 COS。

上传进度到了 100%，最终还是回调了 failCallBack 接口，该如何处理？

上传进度这里只是代表 SDK 写入到网络中的进度，100% 并不表示上传完成，只有回调 successCallBack 接口才真正上传成功，如果在最后发送 Complete Multipart Upload 请求时产生了异常，那么会回调 failCallBack 接口。

通过 CosTransferManger 上传和下载报错权限问题，该如何处理？

CosTransferManger 下载文件时会先进行 Head 操作，因此需要同时授权 HeadObject 和 GetObject 两个权限，上传时则需要简单上传和分块上传所有接口的权限。

调用接口时，报错 lock timeout 或者 no credential for sign，或者签名过期等错误，该如何处理？

如果使用的是临时密钥的方式，请检查 initWithSessionCredentialCallback 方法中返回的密钥是否及时更新了，或者密钥是否有效，如果是临时密钥则需要带上 token。

上传时报错 calculate md5 error，该如何处理？

可能是您在上传的过程中修改了文件，导致文件的 MD5 值发生了变化，或者网络很差导致服务端收包产生错误。

请求返回 ServerError 错误，该如何处理？

可能是您通过代理访问 COS，但是代理没有做好转发，直接返回了不正确的回包，导致 SDK 解析错误，可以抓包查看客户端接收的回包是否正常。

调用接口报错 403 权限错误，该如何处理？

权限问题一般不是 SDK 的问题，请检查自己授权信息。您也可以 联系我们 处理。

版权所有：腾讯云计算（北京）有限责任公司

快速体验

最近更新时间：2023-03-30 14:22:42

背景

移动互联网时代，App 作为移动互联网服务的基础设施，往往需要上传和下载大量的数据，数据的安全性和可靠性尤为重要。现在开发者可以将数据存储相关的问题交给 腾讯云对象存储（Cloud Object Storage，COS）服务，而只需要关心自己应用的业务逻辑即可，可减少很多工作量，提升开发效率。本文主要介绍如何快速搭建一个基于 COS 的应用传输服务，在腾讯云 COS 上实现应用数据的上传下载，在您的服务器上只需要部署您自己的业务、生成和管理临时密钥。

腾讯云 COS 提供的体验 demo，您可以参照本文档来体验 COS 传输实践 demo。

相关资源

本文涉及的 Demo 都存放在 Github 仓库，用户可前往获取。

准备工作

- 系统版本：Android 5.0 及以上或 iOS 12.4 及以上。
- 临时密钥服务器或者永久密钥（SecretId、SecretKey 可在云 API 密钥 中获取）
- 腾讯云 APPID（如果需要体验添加 bucket 功能的话）。

搭建用户客户端

配置客户端

1. 在 Github 仓库 下载项目文件，然后用 IDE 打开。
2. 根据 example/src/config/config.template 文件中指引创建 config.ts 并修改相关值。
3. 运行项目，体验 COS 传输实践 demo。

> 注意：
>
> - 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。

版权所有：腾讯云计算（北京）有限责任公司

- 如果您一定要使用永久密钥，SecretId、SecretKey 明文不要暴露到不安全环境下，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

运行示例 Demo

查询存储桶列表

启动示例 App 后，将展示用户已创建的所有存储桶。

创建存储桶

点击右上角**新建存储桶**，在配置页面输入桶名称并选择存储桶的所属 地域。

查询对象列表

选择点击某个存储桶，将看到该存储桶内所有文件以及文件夹。

上传对象

在文件列表页面点击右上角的**上传**，然后选择文件进行上传。

下载对象

在对象列表中，点击文件右方的**下载**按钮，即可下载文件。

版权所有：腾讯云计算（北京）有限责任公司

存储桶操作

最近更新时间：2023-03-30 14:22:42

简介

本文档提供关于存储桶基本操作的相关 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Service（List Buckets）	查询存储桶列表	查询指定账号下所有的存储桶列表
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

查询存储桶列表

功能说明

用于查询指定账号下所有存储桶列表。

示例代码

```
try {
let listAllMyBuckets: ListAllMyBuckets = await Cos.getDefaultService().getService
();
// 存储桶列表详情请查看 ListAllMyBuckets 类
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
console.log(e);
}
```

参数说明

无

返回结果说明

- 成功：返回 ListAllMyBuckets 包含：存储桶列表和存储桶持有者信息。

- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientError 或者 CosXmlServiceError。详情请参见 异常处理。

ListAllMyBuckets 响应包体具体数据内容如下：

参数名称	描述	类型
buckets	存储桶列表	List<Bucket>
owner	存储桶持有者信息	Owner

存储桶（Bucket）中包含如下内容：

参数名称	描述	类型
name	存储桶的名称	String
location	存储桶所在地域	String
createDate	存储桶的创建时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	String

存储桶持有者信息（Owner）中包含如下内容：

参数名称	描述	类型
id	完整 ID	String
disPlayName	存储桶持有者的名字	String

创建存储桶

功能说明

创建一个存储桶（PUT Bucket）。

示例代码

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
let region = "COS_REGION";
// 是否开启多 AZ
```

```
let enableMAZ = false;
try {
await Cos.getDefaultService().putBucket(
bucket,
{
region: region,
enableMAZ: enableMAZ
}
);
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
console.log(e);
}
```

参数说明

参数名称	描述	类型	是否必选
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
enableMAZ	是否创建多 AZ 存储桶	String	否

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientError 或者 CosXmlServiceError。详情请参见 异常处理。

检索存储桶及其权限

功能说明

HEAD Bucket 请求可以确认该存储桶是否存在，是否有权限访问。有以下几种情况：

- 存储桶存在且有读取权限，返回 HTTP 状态码为200。
- 无存储桶读取权限，返回 HTTP 状态码为403。
- 存储桶不存在，返回 HTTP 状态码为404。

示例代码

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
```

```
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
let region = "COS_REGION";
try {
let header = await Cos.getDefaultService().headBucket(
bucket,
region
);
// HTTP 状态码为200，HTTP 头为 header
} catch (e) {
// e.statusCode 查看具体的 HTTP 状态码
console.log(e);
}
```

参数说明

参数名称	描述	类型
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String

返回结果说明

- 成功：返回 HTTP Header。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientError 或者 CosXmlServiceError。详情请参见 异常处理。

删除存储桶

功能说明

删除指定的存储桶（DELETE Bucket）。

> 注意：
> 删除存储桶前，请确保存储桶内的数据和未完成上传的分块数据已全部清空，否则会无法删除存储桶。

示例代码

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
let region = "COS_REGION";
try {
```

```
await Cos.getDefaultService().deleteBucket(
bucket,
region
);
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
console.log(e);
}
```

参数说明

参数名称	描述	类型
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientError 或者 CosXmlServiceError。详情请参见 异常处理。

对象操作
上传对象

最近更新时间：2023-03-30 14:27:55

简介

本文档提供关于对象的上传操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object	简单上传对象	上传一个对象至存储桶

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传操作
Upload Part	上传分块	分块上传对象
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

上传对象

封装了简单上传、分块上传接口，根据文件大小智能的选择上传方式，同时支持续传功能。

示例代码: 上传本地文件

```
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
let region = "COS_REGION";
// 创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
let serviceConfig = {
region: region,
isDebuggable: true,
```

```
isHttps: true,
};
// 创建 TransferConfig 对象，根据需要修改默认的配置参数
// TransferConfig 可以设置智能分块阈值 默认对大于或等于2M的文件自动进行分块上传，可以通过如下代码修改分块阈值
let transferConfig = {
forceSimpleUpload: false,
enableVerification: true,
divisionForUpload: 2097152, // 设置大于等于 2M 的文件进行分块上传
sliceSizeForUpload: 1048576, //设置默认分块大小为 1M
};
// 注册默认 COS TransferManger
await Cos.registerDefaultTransferManger(serviceConfig, transferConfig);

// 获取 CosTransferManger
let cosTransferManger: CosTransferManger = Cos.getDefaultTransferManger();
//let cosTransferManger: CosTransferManger = Cos.getTransferManger(newRegion);
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
let cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
let srcPath = "本地文件的路径"; //本地文件的路径
//若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传；否则，赋值 undefined
let _uploadId = undefined;

// 上传成功回调
let successCallBack = (header?: object) => {
// todo 上传成功后的逻辑
};
//上传失败回调
let failCallBack = (clientError?: CosXmlClientError, serviceError?: CosXmlServiceError) => {
// todo 上传失败后的逻辑
if (clientError) {
console.log(clientError);
}
if (serviceError) {
console.log(serviceError);
}
};
//上传状态回调，可以查看任务过程
let stateCallBack = (state: TransferState) => {
// todo notify transfer state
};
//上传进度回调
let progressCallBack = (complete: number, target: number) => {
// todo Do something to update progress...
```

```
};
//初始化分块完成回调
let initMultipleUploadCallBack = (bucket: string, cosKey: string, uploadId: strin
g) => {
//用于下次续传上传的 uploadId
_uploadId = uploadId;
};

//开始上传
let transferTask:TransferTask = await cosTransferManger.upload(
bucket,
cosPath,
srcPath,
{
uploadId: _uploadId,
resultListener: {
successCallBack: successCallBack,
failCallBack: failCallBack
},
stateCallback: stateCallBack,
progressCallback: progressCallBack,
initMultipleUploadCallback: initMultipleUploadCallBack,
}
);
//暂停任务
transferTask.pause();
//恢复任务
transferTask.resume();
//取消任务
transferTask.cancel();
```

参数说明

TransferConfig 用于配置 COS 上传服务，其主要成员说明如下：

参数名称	描述	类型	默认值	支持平台
divisionForUpload	设置启用分块上传的最小对象大小	Int	2097152	Android和iOS
sliceSizeForUpload	设置分块上传时的分块大小	Int	1048576	Android和iOS
enableVerification	分片上传时是否整体校验	Bool	true	Android和iOS
forceSimpleUpload	是否强制使用简单上传	Bool	false	Android

upload 方法参数

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String
cosPath	对象键 是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg	String
fileUri	需要上传的本地文件路径	String
uploadId	若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传	String
stroageClass	存储类型	String
trafficLimit	单链接限速（限速值设置范围为 819200 - 838860800，单位为 bit/s，即100KB/s - 100MB/s）	Int
ResultListener	上传结果回调（包括成功回调和失败回调）	ResultListener
StateCallBack	上传状态回调	StateCallBack
ProgressCallBack	上传进度回调	ProgressCallBack
InitMultipleUploadCallback	初始化分块完成回调	InitMultipleUploadCallbac

返回结果说明

- 成功：返回 TransferTask，可以对 TransferTask 进行暂停、恢复、取消操作。
- 失败：发生错误（如身份认证失败），会在 ResultListener 的失败回调中抛出异常 CosXmlClientError 或者 CosXmlServiceError。详情请参见 异常处理。

下载对象

最近更新时间：2023-03-30 14:27:55

简介

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地
HEAD Object	查询对象元数据	查询对象的元数据信息

下载对象

下载接口支持暂停、恢复以及取消下载请求，同时支持断点下载功能。

示例代码: 下载对象

```
// 高级下载接口支持断点续传，所以会在下载前先发起 HEAD 请求获取文件信息。
// 如果您使用的是临时密钥或者使用子账号访问，请确保权限列表中包含 HeadObject 的权限。

// CosTransferManger 支持断点下载，您只需要保证 bucket、cosPath、savePath
// 参数一致，SDK 便会从上次已经下载的位置继续下载。

// 获取 CosTransferManger
let cosTransferManger: CosTransferManger = Cos.getDefaultTransferManger();
//let cosTransferManger: CosTransferManger = Cos.getTransferManger(newRegion);
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
let cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
let downliadPath = "本地文件的路径"; //保存到本地文件的路径

// 下载成功回调
let successCallBack = (header?: object) => {
// todo 下载成功后的逻辑
};
//下载失败回调
let failCallBack = (clientError?: CosXmlClientError, serviceError?: CosXmlService
Error) => {
// todo 下载失败后的逻辑
if (clientError) {
```

```
console.log(clientError);
}
if (serviceError) {
console.log(serviceError);
}
};
//下载状态回调，可以查看任务过程
let stateCallBack = (state: TransferState) => {
// todo notify transfer state
};
//下载进度回调
let progressCallBack = (complete: number, target: number) => {
// todo Do something to download progress...
};

//开始下载
let transferTask:TransferTask = await cosTransferManger.download(
bucket,
cosPath,
downliadPath,
{
resultListener: {
successCallBack: successCallBack,
failCallBack: failCallBack
},
stateCallback: stateCallBack,
progressCallback: progressCallBack
}
);
//暂停任务
transferTask.pause();
//恢复任务
transferTask.resume();
//取消任务
transferTask.cancel();
```

参数说明

参数名称	描述	类型	是否必选
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是

参数名称	描述	类型	是否必选
cosPath	对象键 是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg	String	是
savePath	下载保存到本地文件的绝对路径	String	是
versionId	指定要下载的版本 ID	String	否
trafficLimit	单链接限速（限速值设置范围为 819200 - 838860800，单位为 bit/s，即100KB/s - 100MB/s）	Int	否
ResultListener	下载结果回调（包括成功回调和失败回调）	ResultListener	否
StateCallBack	下载状态回调	StateCallBack	否
ProgressCallBack	下载进度回调	ProgressCallBack	否

返回结果说明

- 成功：返回 TransferTask，可以对 TransferTask 进行暂停、恢复、取消操作。
- 失败：发生错误（如身份认证失败），会在 ResultListener 的失败回调中抛出异常 CosXmlClientError 或者 CosXmlServiceError。详情请参见 异常处理。

列出对象

最近更新时间：2023-03-30 14:27:55

简介

本文档提供关于列出对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

示例代码一: 获取第一页数据

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
try {
let bucketContents: BucketContents = await Cos.getDefaultService().getBucket(
bucket,
{
prefix: "dir/", // 前缀匹配，用来规定返回的对象前缀地址
maxKeys: 100 // 单次返回最大的条目数量，默认1000
}
);
// 表示数据被截断，需要拉取下一页数据
let isTruncated = bucketContents.isTruncated;
// nextMarker 表示下一页的起始位置
let nextMarker = bucketContents.nextMarker;
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
console.log(e);
}
```

示例代码二：请求下一页数据

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
// prevPageBucketContents 是上一页的返回结果，这里的 nextMarker 表示下一页的起始位置
let prevPageMarker = prevPageBucketContents.nextMarker;
try {
let bucketContents: BucketContents = await Cos.getDefaultService().getBucket(
bucket,
{
prefix: "dir/", // 前缀匹配，用来规定返回的对象前缀地址
marker: prevPageMarker, // 起始位置
maxKeys: 100 // 单次返回最大的条目数量，默认1000
}
);
// 表示数据被截断，需要拉取下一页数据
let isTruncated = bucketContents.isTruncated;
// nextMarker 表示下一页的起始位置
let nextMarker = bucketContents.nextMarker;
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
console.log(e);
}
```

示例代码三：获取对象列表与子目录

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
// 定界符为一个符号，如果有 Prefix，
// 则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，
// 然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始
let delimiter = "/";
try {
let bucketContents: BucketContents = await Cos.getDefaultService().getBucket(
bucket,
{
prefix: "dir/", // 前缀匹配，用来规定返回的对象前缀地址
delimiter: delimiter,
maxKeys: 100 // 单次返回最大的条目数量，默认1000
}
);
// 表示数据被截断，需要拉取下一页数据
let isTruncated = bucketContents.isTruncated;
// nextMarker 表示下一页的起始位置
let nextMarker = bucketContents.nextMarker;
```

```
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
console.log(e);
}
```

参数说明

参数名称	描述	类型	是否必选
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
prefix	对象键匹配前缀，限定响应中只包含指定前缀的对象键	String	否
delimiter	一个字符的分隔符，用于对对象键进行分组。所有对象键中从 prefix 或从头（如未指定 prefix）到首个 delimiter 之间相同的部分将作为 CommonPrefixes 下的一个 Prefix 节点。被分组的对象键不再出现在后续对象列表中，具体场景和用法可参考上面的实际案例	String	否
encodingType	规定返回值的编码方式，可选值：url，代表返回的对象键为 URL 编码（百分号编码）后的值，例如“腾讯云”将被编码为 `%E8%85%BE%E8%AE%AF%E4%BA%91`	String	否
marker	起始对象键标记，从该标记之后（不含）按照 UTF-8 字典序返回对象键条目	String	否
maxKeys	单次返回最大的条目数量，默认值为1000，最大为1000 注意：该参数会限制每一次 List 操作返回的最大条目数，COS 在每次 List 操作中将返回不超过 max-keys 所设定数值的条目（即 CommonPrefixes 和 Contents 的总和），如果单次响应中未列出所有对象，COS 会返回 NextMarker 节点，其值作为您下次 List 请求的 marker 参数，以便您列出后续对象	Int	否

返回结果说明

- 成功：返回 BucketContents 包含：对象列表和列表信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientError 或者 CosXmlServiceError。详情请参见 异常处理。

BucketContents 响应包体具体数据内容如下：

参数名称	描述	类型
name	存储桶的名称，格式为<BucketName-APPID>，例如 examplebucket-1250000000	String

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
encodingType	编码格式，对应请求中的 encodingType 参数，且仅当请求中指定了 encodingType 参数才会返回该节点	String
prefix	对象键匹配前缀，对应请求中的 prefix 参数	String
marker	起始对象键标记，从该标记之后（不含）按照 UTF-8 字典序返回对象键条目，对应请求中的 marker 参数	String
maxKeys	单次响应返回结果的最大条目数量，对应请求中的 maxKeys 参数	Int
delimiter	分隔符，对应请求中的 delimiter 参数，且仅当请求中指定了 delimiter 参数才会返回该节点	String
isTruncated	响应条目是否被截断，布尔值，例如 true 或 false	Bool
nextMarker	仅当响应条目有截断（IsTruncated 为 true）才会返回该节点，该节点的值为当前响应条目中的最后一个对象键，当需要继续请求后续条目时，将该节点的值作为下一次请求的 marker 参数传入	String
contentsList	对象条目	List<Content>
commonPrefixesList	从 prefix 或从头（如未指定 prefix）到首个 delimiter 之间相同的部分，定义为 Common Prefix。仅当请求中指定了 delimiter 参数才有可能返回该节点	List<CommonPrefixes>

对象（Content）中包含如下内容：

参数名称	描述	类型
key	对象键	String
lastModified	对象最后修改时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	String
eTag	对象的实体标签（Entity Tag），是对象被创建时标识对象内容的信息标签，可用于检查对象的内容是否发生变化，例如 `“8e0b617ca298a564c3331da28dcb50df”`，此头部并不一定返回对象的 MD5 值，而是根据对象上传和加密方式而有所不同	String
size	对象大小，单位为 Byte	Int
owner	对象持有者信息	Owner
storageClass	对象存储类型。枚举值请参见 存储类型 文档，例如 STANDARD_IA，ARCHIVE	String

版权所有：腾讯云计算（北京）有限责任公司

持有者信息（Owner） 中包含如下内容：

参数名称	描述	类型
id	对象持有者的 APPID	String
disPlayName	对象持有者的名称	String

CommonPrefixes 中包含如下内容：

参数名称	描述	类型
prefix	单条 Common Prefix 的前缀	String

版权所有：腾讯云计算（北京）有限责任公司

删除对象

最近更新时间：2023-03-30 14:27:55

简介

本文档提供关于对象的删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象

删除单个对象

功能说明

删除指定的对象（DELETE Object）。

示例代码

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
let region = "COS_REGION";
//对象在存储桶中的位置标识符，即对象键
let cosPath = "exampleobject";
try {
await Cos.getDefaultService().deleteObject(bucket, cosPath, undefined, region);
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
console.log(e);
}
```

参数说明

参数名称	描述	类型	是否必选
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是

参数名称	描述	类型	是否必选
cosPath	对象键 是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg	String	是
versionId	指定要删除的版本 ID	String	否

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientError 或者 CosXmlServiceError。详情请参见 异常处理。

单链接限速

最近更新时间：2023-03-30 14:27:55

简介

本文档提供关于调用上传下载接口时对链接进行限速。

使用说明

限速值设置范围为 **819200 - 838860800**，单位默认为 bit/s，即100KB/s - 100MB/s，如果超出该范围将返回400错误。
通过调用 upload 和 download 方法时传入的 trafficLimit 参数进行限速。

示例代码一：上传时对单链接限速

```
// 存储桶所在地域简称，例如广州地区是 ap-guangzhou
let region = "COS_REGION";
// 创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
let serviceConfig = {
region: region,
isDebuggable: true,
isHttps: true,
};
// 创建 TransferConfig 对象，根据需要修改默认的配置参数
// TransferConfig 可以设置智能分块阈值 默认对大于或等于2M的文件自动进行分块上传，可以通过如下代码修改分块阈值
let transferConfig = {
forceSimpleUpload: false,
enableVerification: true,
divisionForUpload: 2097152, // 设置大于等于 2M 的文件进行分块上传
sliceSizeForUpload: 1048576, //设置默认分块大小为 1M
};
// 注册默认 COS TransferManger
await Cos.registerDefaultTransferManger(serviceConfig, transferConfig);

// 获取 CosTransferManger
let cosTransferManger: CosTransferManger = Cos.getDefaultTransferManger();
//let cosTransferManger: CosTransferManger = Cos.getTransferManger(newRegion);
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
```

```
let cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
let srcPath = "本地文件的路径"; //本地文件的路径
//若存在初始化分块上传的 UploadId，则赋值对应的 uploadId 值用于续传；否则，赋值 undefined
let _uploadId = undefined;

// 上传成功回调
let successCallBack = (header?: object) => {
// todo 上传成功后的逻辑
};
//上传失败回调
let failCallBack = (clientError?: CosXmlClientError, serviceError?: CosXmlService
Error) => {
// todo 上传失败后的逻辑
if (clientError) {
console.log(clientError);
}
if (serviceError) {
console.log(serviceError);
}
};
//上传状态回调，可以查看任务过程
let stateCallBack = (state: TransferState) => {
// todo notify transfer state
};
//上传进度回调
let progressCallBack = (complete: number, target: number) => {
// todo Do something to update progress...
};
//初始化分块完成回调
let initMultipleUploadCallBack = (bucket: string, cosKey: string, uploadId: strin
g) => {
//用于下次续传上传的 uploadId
_uploadId = uploadId;
};

// 设置单链接限速，单位为 bit/s，示例设置为 1M/s
let _trafficLimit = 1024 * 1024 * 8;

//开始上传
let transferTask:TransferTask = await cosTransferManger.upload(
bucket,
cosPath,
srcPath,
{
uploadId: _uploadId,
trafficLimit: _trafficLimit,
resultListener: {
```

```
successCallBack: successCallBack,
failCallBack: failCallBack
},
stateCallback: stateCallBack,
progressCallback: progressCallBack,
initMultipleUploadCallback: initMultipleUploadCallBack,
}
);
```

示例代码二：下载时对单链接限速

```
// 高级下载接口支持断点续传，所以会在下载前先发起 HEAD 请求获取文件信息。
// 如果您使用的是临时密钥或者使用子账号访问，请确保权限列表中包含 HeadObject 的权限。

// CosTransferManger 支持断点下载，您只需要保证 bucket、cosPath、savePath
// 参数一致，SDK 便会从上次已经下载的位置继续下载。

// 获取 CosTransferManger
let cosTransferManger: CosTransferManger = Cos.getDefaultTransferManger();
//let cosTransferManger: CosTransferManger = Cos.getTransferManger(newRegion);
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。
https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
let cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
let downliadPath = "本地文件的路径"; //保存到本地文件的路径

// 下载成功回调
let successCallBack = (header?: object) => {
// todo 下载成功后的逻辑
};
//下载失败回调
let failCallBack = (clientError?: CosXmlClientError, serviceError?: CosXmlService
Error) => {
// todo 下载失败后的逻辑
if (clientError) {
console.log(clientError);
}
if (serviceError) {
console.log(serviceError);
}
};
//下载状态回调，可以查看任务过程
let stateCallBack = (state: TransferState) => {
// todo notify transfer state
};
//下载进度回调
```

```
let progressCallBack = (complete: number, target: number) => {
// todo Do something to download progress...
};

// 设置单链接限速，单位为 bit/s，示例设置为 1M/s
let _trafficLimit = 1024 * 1024 * 8;

//开始下载
let transferTask:TransferTask = await cosTransferManger.download(
bucket,
cosPath,
downliadPath,
{
trafficLimit: _trafficLimit,
resultListener: {
successCallBack: successCallBack,
failCallBack: failCallBack
},
stateCallback: stateCallBack,
progressCallback: progressCallBack
}
);
```

生成预签名链接

最近更新时间：2023-06-16 15:38:05

简介

本文档提供关于生成对象预签名链接相关的 SDK 示例代码。

关于使用预签名 URL 上传的说明请参见 预签名授权上传， 使用预签名 URL 下载的说明请参见 预签名授权下载。

说明：

- 签名规则。请参考 请求签名。
- 建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄露目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。

生成对象下载预签名链接

功能说明

生成用于 COS 对象下载的预签名链接。

示例代码

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。https://console.intl.cloud.tencent.com/cos5/bucket
let bucket = "examplebucket-1250000000";
//对象在存储桶中的位置标识符，即对象键
let cosPath = "exampleobject";
//url请求参数
let requestParameters = {"test1k":"test1v", "test2k":"test2v"};
try {
// 获取对象下载预签名链接
String objectUrl = await Cos.getDefaultService().getPresignedUrl(
bucket,
cosPath,
{
parameters: requestParameters
```

```
}
);
} catch (e) {
// 失败后会抛异常 根据异常进行业务处理
console.log(e);
}
```

参数说明

参数名称	描述	类型	是否必选
bucket	桶名称，Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述	String	是
cosPath	对象键 是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg	String	是
signValidTime	设置签名有效期(单位为秒)，注意这里是签名有效期，您需要自行保证密钥有效期	Int	否
signHost	是否签入Header Host，不签可能导致请求失败或安全漏洞	Bool	否
parameters	http请求参数，传入的请求参数需与实际请求相同，能够防止用户篡改此HTTP请求的参数	Map	否

返回结果说明

- 成功：返回对象下载预签名链接。
- 失败：发生错误（如身份认证失败），抛出异常 CosXmlClientError 或者 CosXmlServiceError。详情请参见 异常处理。

异常处理

最近更新时间：2023-03-30 14:22:42

简介

调用 SDK 接口请求 COS 服务失败时，系统将抛出 CosXmlClientError（客户端错误）或者 CosXmlServiceError （服务端错误）。

- CosXmlClientError 是由于客户端无法和 COS 服务端正常进行交互所引起。如客户端无法连接到服务端，无法解析服务端返回的数据，读取本地文件发生 IO 错误等。
- CosXmlServiceError 是客户端和 COS 服务端交互正常，但操作 COS 资源失败。如客户端访问一个不存在 Bucket ，删除一个不存在的文件，没有权限进行某个操作等。

客户端错误

CosXmlClientError 例如本地文件发生 IO 错误等。CosXmlClientError 包含了定义的客户端错误码和具体的错误描述信息。捕获错误后，建议对整个错误进行打印，错误包含了必须的排查因素。以下是错误成员变量的描述：

成员	描述	类型
errorCode	客户端错误码，如10000表示参数检验失败，更多详情请参见 SDK 错误码	int
message	客户端错误信息	string

服务端错误

CosXmlServiceError 例如客户端访问一个不存在 Bucket，删除一个不存在的文件，没有权限进行某个操作，服务端故障错误等。CosXmlServiceError 包含了服务端返回的状态码，requestid，出错明细等。捕获错误后，建议对整个错误进行打印，错误包含了必须的排查因素。以下是错误成员变量的描述：

成员	描述	类型
requestId	请求 ID，用于表示一个请求，对于排查问题十分重要	string
statusCode	response 的 status 状态码，4xx 是指请求因客户端而失败，5xx 是服务端异常导致的失败，更多详情请参见 COS 错误信息	string
errorCode	请求失败时 body 返回的 Error Code，更多详情请参见 COS 错误信息	string

版权所有：腾讯云计算（北京）有限责任公司

成员	描述	类型
errorMessage	请求失败时 body 返回的 Error Message，更多详情请参见 COS 错误信息	string

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。
   COS 自助诊断工具
   点击自助诊断
   输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**。
4. 稍侯片刻，便能看到相应的智能诊断结果。

设置访问域名

最近更新时间：2023-03-30 14:22:42

简介

本文档提供关于如何使用非默认域名请求 COS 服务。

自定义源站域名

关于如何设置自定义源站域名请参见 自定义源站域名。

以下代码展示了如何使用自定义源站域名访问 COS 服务。

示例代码

```
// 存储桶 region 可以在 COS 控制台指定存储桶的概览页查看 https://console.intl.cloud.tencent.com/cos5/bucket/ , 关于地域的详情见 https://cloud.tencent.com/document/product/436/6224
let region = "ap-beijing"; // 您的存储桶地域
let customDomain = "exampledomain.com"; // 自定义域名
// 创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
let cosXmlServiceConfig = {
region: region,
isDebuggable: false,
isHttps: true,
hostFormat: customDomain // 修改请求的域名
};
// 注册默认 COS Service
await Cos.registerDefaultService(cosXmlServiceConfig)
```

全球加速域名

关于全球加速功能请参考 全球加速功能概述。

以下代码展示了如何使用全球加速域名访问 COS 服务。

示例代码

```
// 存储桶 region 可以在 COS 控制台指定存储桶的概览页查看 https://console.intl.cloud.tencent.com/cos5/bucket/ , 关于地域的详情见 https://cloud.tencent.com/document/product/436/6224
let region = "ap-beijing"; // 您的存储桶地域
let accelerate = true; // 使能全球加速域名
// 创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
let cosXmlServiceConfig = {
region: region,
isDebuggable: false,
isHttps: true,
accelerate: accelerate
};
// 注册默认 COS Service
await Cos.registerDefaultService(cosXmlServiceConfig)
```

Mini Program SDK 快速入门

最近更新时间：2023-01-30 11:50:32

下载与安装

相关资源

- 对象存储 COS 的 XML 小程序 SDK 源码下载地址：XML 小程序 SDK。
- SDK 快速下载地址：XML 小程序 SDK。
- 演示示例 Demo 下载地址：XML 小程序 SDK Demo。
- SDK 文档中的所有示例代码请参见 SDK 代码示例。
- SDK 更新日志请参见 ChangeLog。
- SDK 常见问题请参见：小程序 SDK 常见问题。

> 说明：
> 如果您在使用 SDK 时遇到函数或方法不存在等错误，请先将 SDK 升级到最新版再重试。

环境依赖

1. 该 SDK 只适用于微信小程序环境。
2. 登录 对象存储控制台 创建存储桶后，获取存储桶名称和 地域信息。
3. 登录 访问管理控制台 获取您的项目 SecretId 和 SecretKey。

> 说明：
> - 关于本文中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参见 COS 术语信息。
> - 关于跨端框架（例如 uni-app）的使用说明，使用小程序 SDK 开发后无法打包成正常使用的移动应用，例如 Android App、iOS App，需要使用对应的 Android SDK、iOS SDK。
> - XCosSecurityToken 字段说明：v1.2.0之前版本的 SDK 只支持 XCosSecurityToken，v1.2.0及之后的版本请使用 SecurityToken 代替。

安装 SDK

安装小程序 SDK 有两种方式：手动安装和 npm 安装，具体安装方法如下。

版权所有：腾讯云计算（北京）有限责任公司

手动安装

复制源码文件中的 cos-wx-sdk-v5.js 到自己小程序代码根目录下任意路径，并用相对路径引用：

```
var COS = require('./lib/cos-wx-sdk-v5.js')
```

npm 安装

如果小程序代码使用了 webpack 打包，则通过 npm 安装依赖即可：

```
npm install cos-wx-sdk-v5
```

其中，程序代码使用 `var COS = require('cos-wx-sdk-v5');` 进行引用。更多详情请参见 npm 支持。

开始使用

小程序域名白名单配置

小程序里请求 COS 需要登录到 微信公众平台，选择**开发 > 开发设置 > 服务器域名**，配置域名白名单。SDK 使用到了两个接口：

1. cos.postObject 使用 wx.uploadFile 方法。
2. 其他方法使用 wx.request 方法。

需要在对应白名单里，配置 COS 域名，白名单域名格式有两种：

1. 如果是标准请求，可以配置存储桶域名作为白名单域名，例如：`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com` 。
2. 如果小程序使用的存储桶多，可以选择后缀式请求 COS，只需要在 SDK 实例化时传入 `ForcePathStyle: true` ，这种方式需要配置地域域名作为白名单，例如： `cos.ap-guangzhou.myqcloud.com` 。

初始化

注意：

- 建议用户 使用临时密钥 调用 SDK，通过临时授权的方式进一步提高 SDK 使用的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。
- 如果您一定要使用永久密钥，建议遵循 最小权限指引原则 对永久密钥的权限范围进行限制。

```
var COS = require('./lib/cos-wx-sdk-v5.js')
```

```
var cos = new COS({
// ForcePathStyle: true, // 如果使用了很多存储桶，可以通过打开后缀式，减少配置白名单域名数量，请求时会用地域域名
SimpleUploadMethod: 'putObject', // 强烈建议，高级上传、批量上传内部对小文件做简单上传时使用putObject,sdk版本至少需要v1.3.0
getAuthorization: function (options, callback) {
// 初始化时不会调用，只有调用 cos 方法（例如 cos.putObject）时才会进入
// 异步获取临时密钥
wx.request({
url: 'https://example.com/server/sts.php',
data: {
bucket: options.Bucket,
region: options.Region,
},
dataType: 'json',
success: function (result) {
var data = result.data;
var credentials = data && data.credentials;
if (!data || !credentials) return console.error('credentials invalid');
callback({
TmpSecretId: credentials.tmpSecretId,
TmpSecretKey: credentials.tmpSecretKey,
// v1.2.0之前版本的 SDK 使用 XCosSecurityToken 而不是 SecurityToken
SecurityToken: credentials.sessionToken,
// 建议返回服务器时间作为签名的开始时间，避免用户浏览器本地时间偏差过大导致签名错误
StartTime: data.startTime, // 时间戳，单位秒，如：1580000000
ExpiredTime: data.expiredTime, // 时间戳，单位秒，如：1580000900
});
}
});
}
});
// 接下来可以通过 cos 实例调用 COS 请求。
// TODO
```

配置项

使用示例

创建一个 COS SDK 实例，COS SDK 支持以下几种格式创建：

- 格式一（推荐）：后端通过获取临时密钥给到前端，前端计算签名。

```
var cos = new COS({
SimpleUploadMethod: 'putObject', // 强烈建议，高级上传、批量上传内部对小文件做简单上传时使
```

```
用 putObject,SDK 版本至少需要v1.3.0
// 必选参数
getAuthorization: function (options, callback) {
// 服务端 JS 和 PHP 示例：https://github.com/tencentyun/cos-js-sdk-v5/blob/master/server/
// 服务端其他语言参考 COS STS SDK ：https://github.com/tencentyun/qcloud-cos-sts-sdk
// STS 详细文档指引看：https://intl.cloud.tencent.com/document/product/436/14048
wx.request({
url: 'https://example.com/server/sts.php',
data: {
// 可从 options 取需要的参数
},
success: function (result) {
var data = result.data;
var credentials = data && data.credentials;
if (!data || !credentials) return console.error('credentials invalid');
callback({
TmpSecretId: credentials.tmpSecretId,
TmpSecretKey: credentials.tmpSecretKey,
// v1.2.0之前版本的 SDK 使用 XCosSecurityToken 而不是 SecurityToken
SecurityToken: credentials.sessionToken,
// 建议返回服务器时间作为签名的开始时间，避免用户浏览器本地时间偏差过大导致签名错误
StartTime: data.startTime, // 时间戳，单位秒，如：1580000000
ExpiredTime: data.expiredTime, // 时间戳，单位秒，如：1580000900
});
}
});
}
});
```

说明：
临时密钥生成和使用可参见 临时密钥生成及使用指引。

- 格式二（推荐）：细粒度控制权限，后端通过获取临时密钥给到前端，前端只有相同请求才重复使用临时密钥，后端可以通过 Scope 细粒度控制权限。

```
var cos = new COS({
SimpleUploadMethod: 'putObject', // 强烈建议，高级上传、批量上传内部对小文件做简单上传时使用 putObject,SDK 版本至少需要v1.3.0
// 必选参数
getAuthorization: function (options, callback) {
// 服务端示例：https://github.com/tencentyun/qcloud-cos-sts-sdk/edit/master/scope.m
```

```
d
wx.request({
url: 'https://example.com/server/sts-scope.php',
data: JSON.stringify(options.Scope),
success: function (result) {
var data = result.data;
var credentials = data && data.credentials;
if (!data || !credentials) return console.error('credentials invalid');
callback({
TmpSecretId: credentials.tmpSecretId,
TmpSecretKey: credentials.tmpSecretKey,
// v1.2.0之前版本的 SDK 使用 XCosSecurityToken 而不是 SecurityToken
SecurityToken: credentials.sessionToken,
// 建议返回服务器时间作为签名的开始时间，避免用户浏览器本地时间偏差过大导致签名错误
StartTime: data.startTime, // 时间戳，单位秒，如：1580000000
ExpiredTime: data.expiredTime, // 时间戳，单位秒，如：1580000900
ScopeLimit: true, // 细粒度控制权限需要设为 true，会限制密钥只在相同请求时重复使用
});
}
});
}
});
```

说明：
临时密钥生成和使用可参见 临时密钥生成及使用指引。

- 格式三（不推荐）：前端每次请求前都需要通过 getAuthorization 获取签名，后端使用固定密钥或临时密钥计算签名返回给前端。该格式分块上传权限不易控制，不推荐您使用此格式。

```
var cos = new COS({
SimpleUploadMethod: 'putObject', // 强烈建议，高级上传、批量上传内部对小文件做简单上传时使用 putObject,SDK 版本至少需要v1.3.0
// 必选参数
getAuthorization: function (options, callback) {
// 服务端获取签名，请参考对应语言的 COS SDK：https://cloud.tencent.com/document/product/436/6474
// 注意：这种有安全风险，后端需要通过 method、pathname 严格控制好权限，例如不允许 put / 等
wx.request({
url: 'https://example.com/server/auth.php',
data: JSON.stringify(options.Scope),
success: function (result) {
var data = result.data;
```

```
if (!data || !data.authorization) return console.error('authorization invalid');
callback({
Authorization: data.authorization,
// v1.2.0之前版本的 SDK 使用 XCosSecurityToken 而不是 SecurityToken
// SecurityToken: data.sessionToken, // 如果使用临时密钥，需要把 sessionToken 传给 SecurityToken
});
}
});
},
});
```

- 格式四（不推荐）：前端使用固定密钥计算签名，该格式适用于前端调试，若使用此格式，请避免泄露密钥。

```
// SECRETID 和 SECRETKEY 请登录 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
var cos = new COS({
SecretId: 'SECRETID',
SecretKey: 'SECRETKEY',
SimpleUploadMethod: 'putObject', // 强烈建议，高级上传、批量上传内部对小文件做简单上传时使用putObject,sdk版本至少需要v1.3.0
});
```

构造函数参数说明

参数名	参数描述	类型	是否必填
SecretId	用户的 SecretId	String	否
SecretKey	用户的 SecretKey，建议只在前端调试时使用，避免暴露密钥	String	否
FileParallelLimit	同一个实例下上传的文件并发数，默认值3	Number	否
ChunkParallelLimit	同一个上传文件的分块并发数，默认值3	Number	否
ChunkRetryTimes	分块上传及分块复制时，出错重试次数，默认值2（加第一次，请求共3次）	Number	否
ChunkSize	分块上传时，每块的字节数大小，默认值1048576（1MB）	Number	否
SliceSize	使用 uploadFiles 批量上传时，文件大小大于该数值就使用分块上传 sliceUploadFile，否则使用简单上传 putObject，默认值1048576（1MB）	Number	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
CopyChunkParallelLimit	进行分块复制操作中复制分块上传的并发数，默认值20	Number	否
CopyChunkSize	使用 sliceCopyFile 分块复制文件时，每块的大小字节数，默认值10485760（10MB）	Number	否
CopySliceSize	使用 sliceCopyFile 分块复制文件时，文件大小大于该数值就会使用分块复制 sliceCopyFile，否则使用简单复制 putObjectCopy，默认值10485760（10MB）	Number	否
ProgressInterval	上传进度的回调方法 onProgress 的回调频率，单位 ms，默认值1000	Number	否
Protocol	发请求时用的协议，可选项 `https:` 、 `http:` ，默认判断当前页面是 `http:` 时使用 `http:` ，否则使用 `https:`	String	否
ServiceDomain	调用 getService 方法时，请求的域名，例如 `service.cos.myqcloud.com`	String	否
Domain	调用操作存储桶和对象的 API 时自定义请求域名。可以使用模板，例如 `"{Bucket}.cos.{Region}.myqcloud.com"` ，即在调用 API 时会使用参数中传入的 Bucket 和 Region 进行替换	String	否
UploadQueueSize	上传队列最长大小，超出的任务如果状态不是 waiting、checking、uploading 会被清理，默认10000	Number	否
ForcePathStyle	强制使用后缀式模式发请求，后缀式模式中 Bucket 会放在域名后的 pathname 里，并且 Bucket 会加入签名 pathname 计算，默认为 false	Boolean	否
UploadCheckContentMd5	强制上传文件校验 Content-MD5，会对文件请求 Body 计算 md5 放在 header 的 Content-MD5 字段里，默认为 false	Boolean	否
getAuthorization	获取签名的回调方法，如果没有 SecretId、SecretKey 时，这个参数必选。 **注意: 该回调方法在初始化实例时传入，在使用实例调用接口时才会执行并获取签名。**	Function	否
UseAccelerate	是否启用全球加速域名，默认为 false。若改为 true，需要存储桶开启全球加速功能，详情请参见 开启全球加速。	Boolean	否

参数名	参数描述	类型	是否必填
SimpleUploadMethod	高级上传、批量上传内部对小文件做简单上传的方法名，可选'postObject'或'putObject'，默认为'postObject'（该参数v1.3.0版本开始支持）	String	否

getAuthorization 回调函数说明的函数说明（使用格式一）

```
getAuthorization: function(options, callback) { ... }
```

getAuthorization 的回调参数说明：

参数名	参数描述	类型
options	获取临时密钥需要的参数对象	Object
- Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
- Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
callback	临时密钥获取完成后的回传方法	Function

获取完临时密钥后，callback 回传一个对象，回传对象的属性列表如下：

属性名	参数描述	类型	是否必填
TmpSecretId	获取回来的临时密钥的 tmpSecretId	String	是
TmpSecretKey	获取回来的临时密钥的 tmpSecretKey	String	是
SecurityToken	获取回来的临时密钥的 sessionToken，对应 header 的 x-cos-security-token 字段。v1.2.0之前版本的 SDK 使用 XCosSecurityToken 而不是 SecurityToken	String	是
StartTime	密钥获取的开始时间，即获取时刻的时间戳，单位秒，startTime，如：1580000000，用于签名开始时间，传入该参数可避免前端时间偏差签名过期问题	String	否
ExpiredTime	获取回来的临时密钥的 expiredTime，超时时刻的时间戳，单位秒，如：1580000900	String	是

getAuthorization 回调函数说明（使用格式二）

```
getAuthorization: function(options, callback) { ... }
```

getAuthorization 的函数说明回调参数说明：

参数名	参数描述	类型
options	获取签名需要的参数对象	Object
- Method	当前请求的 Method	Object
- Pathname	请求路径，用于签名计算	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，了解更多可参见 对象概述。 **注意:当使用实例请求的接口不是对象操作相关接口时，该参数为空。**	String
- Query	当前请求的 query 参数对象，{key: 'val'} 的格式	Object
- Headers	当前请求的 header 参数对象，{key: 'val'} 的格式	Object
callback	临时密钥获取完成后的回调	Function

getAuthorization 计算完成后，callback 回传参数支持两种格式：
格式一：回传鉴权凭证字符串 Authorization。
格式二：回传一个对象，对象属性列表如下：

属性名	参数描述	类型	是否必填
Authorization	计算得到的签名字符串	String	是
SecurityToken	获取回来的临时密钥的 sessionToken，对应 header 的 x-cos-security-token 字段。v1.2.0之前版本的 SDK 使用 XCosSecurityToken 而不是 SecurityToken	String	否

获取鉴权凭证

实例本身鉴权凭证可以通过实例化时传入的参数控制如何获取，有三种获取方式：

1. 实例化时，传入 SecretId、SecretKey，每次需要签名都由实例内部计算。
2. 实例化时，传入 getAuthorization 回调，每次需要签名通过这个回调计算完返回签名给实例。
3. 实例化时，传入 getSTS 回调，每次需要临时密钥通过这个回调回去完返回给实例，在每次请求时实例内部使用临时密钥计算得到签名。

开启 beacon 上报

为了持续跟踪和优化 SDK 的质量，给您带来更好的使用体验，我们在 SDK 中引入了 腾讯灯塔 SDK。

> 说明：
> 腾讯灯塔只对 COS 侧的请求性能进行监控，不会上报业务侧数据。

若是想开启该功能，请先确保 SDK 版本升级到1.4.0及以上，然后在初始化中指定 EnableTracker 为 true。
并在小程序域名白名单中 request 合法域名里添加：`https://h.trace.qq.com;https://oth.str.beacon.qq.com;https://otheve.beacon.qq.com;` 。

```
new COS({
EnableTracker: true,
})
```

使用方式

回调方式

文档里默认使用回调方式，使用代码如下:

```
// 这里省略初始化过程和上传参数
var cos = new COS({ ... });
cos.uploadFile({ ... }, function(err, data) {
if (err) {
console.log('上传出错', err);
} else {
console.log('上传成功', data);
}
});
```

Promise

sdk同样支持 Promise 方式调用，例如上述回调方式的代码等同于以下代码:

```
// 这里省略初始化过程和上传参数
var cos = new COS({ ... });
cos.uploadFile({ ... }).then(data => {
console.log('上传成功', data);
}).catch(err => {
console.log('上传出错', err);
});
```

版权所有：腾讯云计算（北京）有限责任公司

同步方式

同步方式基于 JavaScript 的 async 和 await，上述回调方式的代码等同于以下代码:

```
async function upload() {
// 这里省略初始化过程和上传参数
var cos = new COS({ ... });
try {
var data = await cos.uploadFile({ ... });
return { err: null, data: data }
} catch (err) {
return { err: err, data: null };
}
}
// 可以同步拿到请求的返回值,这里举例说明,实际返回的数据格式可以自定义
var uploadResult = await upload();
if (uploadResult.err) {
console.log('上传出错', uploadResult.err);
} else {
console.log('上传成功', uploadResult.data);
}
```

> 注意：
> cos.getObjectUrl 目前只支持回调方式。

使用技巧

通常情况下我们只需要创建一个 COS SDK 实例，然后在需要调用SDK方法的地方直接使用这个实例即可，示例代码如下：

```
var cos = new COS({
....
});
/* 自己封装的上传方法 */
function myUpload() {
// 不需要在每个方法里创建一个 COS SDK 实例
// var cos = new COS({
// ...
// });
cos.putObject({
....
});
}
```

```
/* 自己封装的删除方法 */
function myDelete() {
// 不需要在每个方法里创建一个 COS SDK 实例
// var cos = new COS({
// ...
// });
cos.deleteObject({
....
});
}
```

以下是部分常用接口示例，更详细的初始化方法请参见 demo 示例。

创建存储桶

```
cos.putBucket({
Bucket: 'examplebucket-1250000000',
Region: 'ap-beijing',
}, function (err, data) {
console.log(err || data);
});
```

> 注意：
> 如果需要在小程序创建存储桶，但存储桶名称未知时，无法将存储桶名称配置为域名白名单，可以使用后缀式调用，相关处理措施请参见 常见问题。

查询存储桶列表

```
cos.getService(function (err, data) {
console.log(data && data.Buckets);
});
```

上传对象

强烈推荐使用高级上传接口uploadFile，自动针对小文件使用简单上传，大文件使用分块上传，性能更好。详情请参见 高级上传 文档。
若使用临时密钥方式，需同时授权 `简单上传对象` 和 `分块上传` 的权限。请参考授权指引。
常见上传错误排查，请参考 常见问题。

```
/* 初始化cos */
var cos = new COS({
```

```
// getAuthorization: funciton() {}, // 参考上方初始化
SimpleUploadMethod: 'putObject', // 强烈建议，高级上传、批量上传内部对小文件做简单上传时使用putObject
});
function handleFileInUploading(fileName, filePath) {
cos.uploadFile({
Bucket: 'examplebucket-1250000000', /* 填写自己的 bucket，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: fileName, /* 存储在桶里的对象键（例如:1.jpg，a/b/test.txt，图片.jpg）支持中文，必须字段 */
FilePath: filePath, /* 上传文件路径，必须字段 */
SliceSize: 1024 * 1024 * 5, /* 触发分块上传的阈值，超过5MB使用分块上传，小于5MB使用简单上传。可自行设置，非必须 */
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
if (err) {
console.log('上传失败', err);
} else {
console.log('上传成功');
}
});
}
/* 选择文件，得到临时路径（以选择图片为例，选择其他文件请参考小程序官方api） */
wx.chooseImage({
count: 1, // 默认9
sizeType: ['original'], // 可以指定是原图还是压缩图，默认用原图
sourceType: ['album', 'camera'], // 可以指定来源是相册还是相机，默认二者都有
success: function (res) {
var filePath = res.tempFiles[0].path;
var fileName = filePath.substr(filePath.lastIndexOf('/') + 1);
handleFileInUploading(fileName, filePath);
}
});
```

查询对象列表

```
cos.getBucket({
Bucket: 'examplebucket-1250000000',
Region: 'ap-beijing',
Prefix: 'exampledir/', // 这里传入列出的文件前缀
}, function (err, data) {
console.log(err || data.Contents);
});
```

下载对象

> 注意：
> 该接口用于读取对象内容，如果需要发起浏览器下载文件，可以通过 cos.getObjectUrl 获取 url 再触发浏览器下载，具体参见 预签名 URL 文档。

```
cos.getObject({
Bucket: 'examplebucket-1250000000',
Region: 'ap-beijing',
Key: 'exampleobject.txt',
}, function (err, data) {
console.log(err || data.Body);
});
```

删除对象

```
cos.deleteObject({
Bucket: 'examplebucket-1250000000',
Region: 'ap-beijing',
Key: 'picture.jpg',
}, function (err, data) {
console.log(err || data);
});
```

版权所有：腾讯云计算（北京）有限责任公司

小程序 SDK 常见问题

最近更新时间：2023-02-07 14:57:27

使用临时密钥操作时报错 403，该如何处理？

请检查申请临时密钥时填写的 action 和 allowPrefix 是否正确。

1. 例如调用 cos.putObject()，但是 action 里并没有填写**name/cos:PutObject**，即没有 putObject 权限导致报错 403。
2. 例如操作的 Key 是 `1.jpg`，但是 allowPrefix 填写的是 `test/*`（只允许操作 `test/*` 路径），即没有对应路径的操作权限导致报错 403。

若 aciton 和 allowPrefix 都正确，请参考 临时密钥生成及使用指引 和 访问 COS 时返回 403 错误码。
字段说明：不同语言的 STS SDK，action 和 allowPrefix 所使用的字段不同，例如 STS JAVA SDK 使用的是 allowActions 和 allowPrefixes 字段，请注意留意 STS SDK 中的示例。

小程序 SDK 报错请求过期：Request has expired (Status Code: 403; Error Code: AccessDenied)，该如何处理？

由于签名过期导致，重新生成签名即可解决；若重新生成签名仍报相同的错误，可以再检查机器的本地时间是否为标准的本地时间。

小程序里请求多个域名，或者存储桶名称不确定，怎么解决白名单配置和限制问题？

SDK 实例化时，可以使用 `ForcePathStyle:true` 打开后缀式。只需要真正请求的 url，其格式如下：

```
https://cos-ap-beijing.myqcloud.com/<BucketName-APPID>/<Key>
```

后缀式请求在签名时，存储桶名称 `/<bucketname-appid>` 也会加入签名计算。

小程序如何保存图片到本地？

1. 通过 `cos.getObjectUrl` 获取图片 url。
2. 调用 `wx.downloadFile` 下载图片得到临时路径。
3. 单击保存按钮，调用 `wx.saveImageToPhotosAlbum` 保存到相册。

QQ 小程序可以使用小程序 SDK 上传文件至 COS 吗？

COS 目前仅支持微信小程序，由于 QQ 小程序和微信小程序不互通，无法使用小程序 SDK。

小程序用 cos-wx-sdk-v5 上传文件到存储桶中，getAuthorization 这个函数有什么作用？

小程序 SDK 中的 getAuthorization 函数是后端通过获取临时密钥给到前端，前端计算签名，然后进行上传、删除等操作。推荐您使用临时密钥的形式，防止密钥泄露。详情请参见 小程序 SDK 创建 COS 实例示例。

版权所有：腾讯云计算（北京）有限责任公司

存储桶操作

最近更新时间：2022-07-07 14:27:58

简介

本文档提供关于存储桶的基本操作和访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

基本操作

API	操作名	操作描述
GET Service	查询存储桶列表	查询指定账号下所有的存储桶列表
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

访问控制列表

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶的访问权限控制列表
GET Bucket acl	查询存储桶 ACL	查询指定存储桶的访问权限控制列表

基本操作

查询存储桶列表

功能说明

GET Service 接口是用来查询请求者名下的所有存储桶列表或特定地域下的存储桶列表（Bucket list）。了解更多请参见 GET Service。

使用示例

示例一：列出存储桶列表。

```
cos.getService(function(err, data) {
console.log(err || data);
```

```
});
```

示例二：列出指定地域的存储桶列表。

```
cos.getService({
Region: 'ap-beijing',
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Owner	代表存储桶所有者的对象	Object
- - ID	存储桶持有者的完整 ID，格式为 `qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]`，例如 `qcs::cam::uin/100000000001:uin/100000000001`，其中100000000001为 uin	String
- - DisplayName	存储桶持有者的名称	String
- Buckets	存储桶信息列表	Object

参数名	参数描述	类型
- - Name	存储桶的名称，格式为 BucketName-APPID，例如 examplebucket-1250000000	String
- - Location	存储桶所在地域。枚举值请参见 地域和访问域名 文档，例如 ap-guangzhou，ap-beijing，ap-hongkong 等	String
- - CreationDate	存储桶创建时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	String

创建存储桶

功能说明

PUT Bucket 接口请求可以在指定账号下创建一个存储桶。

使用示例

```
cos.putBucket({
Bucket: 'examplebucket-1250000000',
Region: 'ap-beijing',
ACL: 'private',
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
ACL	定义存储桶的访问控制列表（ACL）属性。枚举值请参见 ACL 概述 文档中存储桶的预设 ACL 部分，例如 private，public-read 等，默认为 private	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	必填
GrantRead	赋予被授权者读取存储桶的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否
GrantWrite	赋予被授权者写入存储桶的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否
GrantReadAcp	赋予被授权者读取存储桶的访问控制列表（ACL）和存储桶策略（Policy）的权限。可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否
GrantWriteAcp	赋予被授权者写入存储桶的访问控制列表（ACL）和存储桶策略（Policy）的权限，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	必填
GrantFullControl	赋予被授权者操作存储桶的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

检索存储桶及其权限

功能说明

HEAD Bucket 请求可以确认该存储桶是否存在，是否有权限访问。有以下几种情况：

- 当该存储桶存在时，返回 HTTP 状态码200。
- 当该存储桶无访问权限时，返回 HTTP 状态码403。
- 当该存储桶不存在时，返回 HTTP 状态码404。

使用示例

检索存储桶信息：

```
cos.headBucket({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

判断存储桶是否存在：

```
function doesBucketExist() {
cos.headBucket({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
if (data) {
console.log('存储桶存在');
} else if (err.statusCode == 404) {
console.log('存储桶不存在');
} else if (err.statusCode == 403) {
console.log('没有该存储桶读权限');
}
});
}
```

参数说明

参数名	参数描述	类型	必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

删除存储桶

功能说明

删除指定账号下的空存储桶。注意，如果删除成功，则返回的 HTTP 状态码为200或204。

> 说明：
> 删除存储桶前，请确保存储桶内的数据和未完成上传的分块数据已全部清空，否则会无法删除存储桶。

使用示例

```
cos.deleteBucket({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

访问控制列表

设置存储桶 ACL

功能说明

PUT Bucket acl 接口用来设置指定存储桶访问权限控制列表（ACL）。

使用示例

设置存储桶公有读：

```
cos.putBucketAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
ACL: 'public-read'
}, function(err, data) {
console.log(err || data);
});
```

为某个用户赋予存储桶所有权限：

```
cos.putBucketAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
GrantFullControl: 'id="qcs::cam::uin/100000000001:uin/100000000001",id="qcs::ca
```

```
m::uin/100000000011:uin/100000000011"' // 100000000001是 uin
}, function(err, data) {
console.log(err || data);
});
```

通过 AccessControlPolicy 修改存储桶权限：

```
cos.putBucketAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
AccessControlPolicy: {
"Owner": { // AccessControlPolicy 里必须有 owner
"ID": 'qcs::cam::uin/100000000001:uin/100000000001' // 100000000001 是 Bucket 所属用户的 Uin
},
"Grants": [{
"Grantee": {
"ID": "qcs::cam::uin/100000000011:uin/100000000011", // 100000000011 是 Uin
},
"Permission": "WRITE"
}]
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
ACL	定义存储桶的访问控制列表（ACL）属性。枚举值请参见 ACL 概述 文档中存储桶的预设 ACL 部分，例如 private，public-read 等，默认为 private	String
GrantRead	赋予被授权者读取存储桶的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
GrantWrite	赋予被授权者写入存储桶的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantReadAcp	赋予被授权者读取存储桶的访问控制列表（ACL）和存储桶策略（Policy）的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantWriteAcp	赋予被授权者写入存储桶的访问控制列表（ACL）和存储桶策略（Policy）的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantFullControl	赋予被授权者操作存储桶的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
AccessControlPolicy	跨域资源共享配置的所有信息列表	Object
- Owner	存储桶持有者的信息	Object

参数名	参数描述	类型
- - ID	存储桶持有者的完整 ID，格式为 `qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]`，例如 `qcs::cam::uin/100000000001:uin/100000000001`，其中100000000001为 uin	String
- Grants	被授权者信息与权限信息列表	ObjectA
- - Permission	授予的权限信息，可选项 READ、WRITE、READ_ACP、WRITE_ACP、FULL_CONTROL。枚举值详情参见 ACL 概述 文档中存储桶的操作部分	String
- - Grantee	被授权者的信息	Object
- - - ID	被授权者的完整 ID，格式为 `qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]`，例如 `qcs::cam::uin/100000000001:uin/100000000001`，其中100000000001为 uin	String
- - - DisplayName	被授权者的名称，一般填写成和 ID 一致的字符串	String
- - - URI	预设用户组，请参见 ACL 概述 文档中预设用户组部分，例如 `http://cam.qcloud.com/groups/global/AllUsers` 或 `http://cam.qcloud.com/groups/global/AuthenticatedUsers`	String

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

查询存储桶 ACL

功能说明

GET Bucket acl 接口用来查询存储桶的访问控制列表（ACL）。该 API 的请求者需要对存储桶有写入 ACL 权限。

使用示例

```
cos.getBucketAcl({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"GrantFullControl": "",
"GrantWrite": "",
"GrantRead": "",
"GrantReadAcp": "id=\"qcs::cam::uin/100000000011:uin/100000000011\"",
"GrantWriteAcp": "id=\"qcs::cam::uin/100000000011:uin/100000000011\"",
"ACL": "private",
"Owner": {
"ID": "qcs::cam::uin/100000000001:uin/100000000001",
"DisplayName": "qcs::cam::uin/100000000001:uin/100000000001"
},
"Grants": [{
"Grantee": {
"ID": "qcs::cam::uin/100000000011:uin/100000000011",
"DisplayName": "qcs::cam::uin/100000000011:uin/100000000011"
},
"Permission": "READ"
}],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名	参数描述	类型	必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Ob
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Nu
- headers	请求返回的头部信息	Ob
data	请求成功时返回的对象，如果请求发生错误则为空	Ob
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Nu
- headers	请求返回的头部信息	Ob
- ACL	定义存储桶的访问控制列表（ACL）属性。枚举值请参见 ACL 概述 文档中存储桶的预设 ACL 部分，例如 private， public-read 等，默认为 private	Str
- GrantRead	具有读取存储桶权限的被授权者 ID 信息	Str
- GrantWrite	具有写入存储桶权限的被授权者 ID 信息	Str
- GrantReadAcp	具有读取存储桶的访问控制列表（ACL）和存储桶策略（Policy）权限的被授权者 ID 信息	Str
- GrantWriteAcp	具有写入存储桶的访问控制列表（ACL）和存储桶策略（Policy）权限的被授权者 ID 信息	Str
- GrantFullControl	具有存储桶所有权限的被授权者 ID 信息	Str
- Owner	存储桶持有者的信息	Ob
- - DisplayName	存储桶持有者的名称	Str
- - ID	存储桶持有者的完整 ID 格式为 `qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	Str
- Grants	被授权者信息与权限信息列表	Ob

参数名	参数描述	类型
- - Permission	指明授予被授权者的权限信息，枚举值：READ、WRITE、READ_ACP、WRITE_ACP、FULL_CONTROL	Stri
- - Grantee	被授权者的信息	Ob
- - - DisplayName	被授权者的名称	Stri
- - - ID	被授权者的完整 ID： • 如果是主账号，格式为 `qcs::cam::uin/<owneruin>:uin/<owneruin>` 或 `qcs::cam::anyone:anyone` （指代所有用户） • 如果是子账号，格式为 `qcs::cam::uin/<owneruin>:uin/<subuin>`	Stri
- - - URI	预设用户组，请参见 ACL 概述 文档中预设用户组部分，例如 `http://cam.qcloud.com/groups/global/AllUsers` 或 `http://cam.qcloud.com/groups/global/AuthenticatedUsers`	Stri

对象操作
上传对象

最近更新时间：2023-05-25 17:38:32

简介

本文档重点提供关于对象的简单操作、其他操作相关的 API 概览以及 SDK 示例代码，并且举例如何使用。

> 说明：
> 常见上传错误排查，请参考 常见问题。

简单操作

API	操作名	操作描述
PUT Object	简单上传对象	上传一个对象至存储桶
Append Object	追加上传对象	将对象以分块追加的方式上传至存储桶
POST Object	表单上传对象	使用表单请求上传对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传任务
Upload Part	上传分块	分块上传对象
List Parts	查询已上传块	查询特定分块上传操作中已上传的块
Complete Multipart Upload	完成分块上传	完成整个对象的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

简单操作

版权所有：腾讯云计算（北京）有限责任公司

简单上传对象

功能说明

PUT Object 接口可以上传一个对象至指定存储桶中。该操作需要请求者对存储桶有 WRITE 权限。

注意：

1. Key（文件名）不能以 `/` 结尾，否则会被识别为文件夹。
2. Key（文件名）同名上传默认为覆盖操作。若您未开启版本控制且不想覆盖云上文件时，请确保上传时的Key不重复。
3. 每个主账号（即同一个 APPID），存储桶的 ACL 规则数量最多为1000条，对象 ACL 规则数量不限制。如果您不需要进行对象 ACL 控制，请在上传时不要设置，默认继承存储桶权限。
4. 上传之后，您可以用同样的 Key 生成预签名链接（下载请指定 method 为 GET，具体接口说明见下文，分享到其他端来进行下载。但注意如果您的文件是私有读权限，那么预签名链接只有一定的有效期。

使用示例

选择文件并上传(使用Body参数):

```
// 此处以选择图片api(wx.chooseImage)为参考，其他api请参考小程序官方文档
wx.chooseImage({
count: 1,
success: function(res) {
var file = res.tempFiles[0];
// 微信小程序里获取文件管理器
var wxfs = wx.getFileSystemManager();
wxfs.readFile({
filePath: file.path,
success: function (res) {
cos.putObject({
Bucket: config.Bucket,
Region: config.Region,
Key: file.name,
Body: res.data, // Body里传入的是文件内容
}, function(err, data) {
console.log(err || data);
});
},
fail: function(err) {
console.error(err)
},
});
},
```

```
fail: function(err) {
console.error(err)
},
});
```

选择文件并上传(使用FilePath参数 需要sdk版本至少达到v1.3.0):

```
// 此处以选择图片api(wx.chooseImage)为参考，其他api请参考小程序官方文档
wx.chooseImage({
count: 1,
success: function(res) {
var file = res.tempFiles[0];
cos.putObject({
Bucket: config.Bucket,
Region: config.Region,
Key: file.name,
FilePath: file.path, // FilePath传入的是文件路径
}, function(err, data) {
console.log(err || data);
});
},
fail: function(err) {
console.error(err)
},
});
```

传字符串作为文件内容：

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
Key: 'picture.jpg', /* 必须 */
Body: 'hello!',
}, function(err, data) {
console.log(err || data);
});
```

创建目录：

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
Key: 'a/', /* 必须 */
Body: '',
}, function(err, data) {
```

版权所有：腾讯云计算（北京）有限责任公司

```
console.log(err || data);
});
```

上传对象（单链接限速）：

> 说明：
> 关于上传对象的限速说明，请参见 单链接限速。

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'exampleobject', /* 必须 */
StorageClass: 'STANDARD',
Body: 'hello!', // 上传文件对象,字符串或选择的文件
Headers: {
'x-cos-traffic-limit': 819200, // 限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，如果超出该范围将返回400错误。
},
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域枚举值请参见 地域和访问域名	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
Body	创建的文件的文本内容，可以为字符串、ArrayBuffer	String\Ar
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String

参数名	参数描述	类型
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，例如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String
GrantRead	赋予被授权者读取对象的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantReadAcp	赋予被授权者读取对象的访问控制列表（ACL）的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantWriteAcp	赋予被授权者写入对象的访问控制列表（ACL）的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD。更多存储类型，请参见 存储类型概述	String
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据保存。大小限制2KB	String
onTaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function
- taskId	上传任务的编号	String
onProgress	进度的回调函数，进度回调响应对象（progressData）属性如下	Function
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number
- progressData.percent	文件上传的百分比，以小数形式呈现，例如：上传50%即为0.5	Number

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String
- Location	创建对象的外网访问域名	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

追加上传对象

功能说明

将对象以分块追加的方式上传至存储桶（APPEND Object）。

注意：

- COS 小程序 SDK 版本需要大于等于 v1.1.1。
- 对象属性为 appendable 时才能使用本接口追加上传。
- 对象首次使用 APPEND Object 接口上传时，该对象的属性自动为 appendable。
- 可以使用 GET Object 或 HEAD Object 接口获取 x-cos-object-type 响应头来判断对象属性。

使用示例

初次追加上传对象:

```
cos.appendObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'test.txt', /* 必须 */
Body: fileObject, // 上传文件对象
Position: 0, // 初次上传为0
```

```
}, function(err, data) {
console.log(err || data);
});
```

判断存储桶内的对象是否可追加对象:

```
cos.headObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'test.txt', /* 必须 */
}, function(err, data) {
if (err) return console.log(err);
// data.headers没有x-cos-object-type字段需要配置expose-headers，参考文档：https://cloud.tencent.com/document/product/436/13318
var objectType = data.headers['x-cos-object-type'];
console.log(objectType === 'appendable');
});
```

查询可追加对象的Position并追加上传:

```
cos.headObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'test.txt', /* 必须 */
}, function(err, data) {
if (err) return console.log(err);
// 首先取到要追加的文件当前长度，即需要上送的Position
var position = data.headers['content-length'];
cos.appendObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'test.txt', /* 必须 */
Body: '66666',
Position: position,
},
function(err, data) {
if (err) return console.log(err);
// 也可以取到下一次上传的position继续追加上传
// data.headers没有x-cos-next-append-position字段需要配置expose-headers，参考文档：https://cloud.tencent.com/document/product/436/13318
var nextPosition = data.headers['x-cos-next-append-position'];
console.log(nextPosition);
})
});
```

参数说明

参数名	参数描述	类型
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
Body	创建的文件的文本内容，可以为字符串	String\Array
Position	追加操作的起始点，单位为字节。首次追加则设置 Position=0，后续追加则设置 Position 为当前 Object 的 content-length	Number
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，例如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String
GrantRead	赋予被授权者读取对象的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String

参数名	参数描述	类型
GrantReadAcp	赋予被授权者读取对象的访问控制列表（ACL）的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantWriteAcp	赋予被授权者写入对象的访问控制列表（ACL）的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，默认值：STANDARD。更多存储类型请参见 存储类型概述	String
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据保存。大小限制2KB	String

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- RequestId	请求的唯一 ID	String

表单上传对象

POST Object 接口请求可以将用户 wx.chooseImage 选择的文件对象（Object）上传至指定存储桶中。该操作需要请求者对存储桶有 WRITE 权限。

> 注意：
> onProgress 进度反馈依赖小程序 UploadTask.onProgressUpdate，在部分安卓机型上会有进度不准确的问题。

使用示例

简单上传文件

```
cos.postObject({
Bucket: 'examplebucket-1250000000',
Region: 'ap-beijing',
Key: filename,
FilePath: tmpFilePath, // wx.chooseImage 选择文件得到的 tmpFilePath
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function (err, data) {
console.log(err || data);
});
```

上传文件到指定目录：

```
var folder = 'examplefolder/';
cos.postObject({
Bucket: 'examplebucket-1250000000',
Region: 'ap-beijing',
Key: folder + filename, /* 必须 */
FilePath: tmpFilePath, // wx.chooseImage 选择文件得到的 tmpFilePath
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	是
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String	否
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String	否
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String	否
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String	否
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String	否
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String	否
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD。更多存储类型，请参见 存储类型概述	String	否
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据保存，大小限制2K	String	否
FilePath	上传文件的临时文件路径，可通过 wx.chooseImage 方法选择得到	String	是
onProgress	进度回调函数，被调用时第一个参数是 processData 对象	Function	否
- progressData.loaded	已经下载的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的下载速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件下载的百分比，以小数形式呈现，例如：下载50%即为0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- ETag	返回对象的 MD5 算法校验值。ETag 的值可以用于检查 Object 在上传过程中是否有损坏， **注意：这里的 ETag 值字符串前后带有双引号，例如** `"09cba091df696af91549de27b8e7d0f6"`	String
- Location	返回创建对象的外网访问域名	String
- VersionId	在启用版本控制的存储桶中，返回对象的版本 ID	String

分块操作

查询分块上传

功能说明

List Multiparts Uploads 用来查询正在进行中的分块上传信息。单次最多列出1000个正在进行中的分块上传。

使用示例

获取前缀为 exampleobject 的未完成的 UploadId 列表，示例如下：

```
cos.multipartList({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Prefix: 'exampleobject', /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

参数名	参数描述	类型	是否必填
Prefix	对象键前缀匹配，限定返回中只包含指定前缀的对象键。注意使用 prefix 查询时，返回的对象键 中仍会包含 Prefix	String	否
Delimiter	定界符为一个分隔符号，用于对对象键进行分组。一般是传 `/` 。所有对象键从 Prefix 或从头（如未指定 Prefix）到首个 delimiter 之间相同部分的路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix	String	否
EncodingType	规定返回值的编码格式，合法值：url	String	否
MaxUploads	设置最大返回的条目数量，合法取值为1 - 1000，默认1000	String	否
KeyMarker	与 upload-id-marker 一起使用 • 当 upload-id-marker 未被指定时： ObjectName 字母顺序大于 key-marker 的条目将被列出 • 当 upload-id-marker 被指定时： ◦ ObjectName 字母顺序大于 key-marker 的条目被列出 ◦ ObjectName 字母顺序等于 key-marker 且 UploadID 大于 upload-id-marker 的条目将被列出	String	否
UploadIdMarker	与 key-marker 一起使用 • 当 key-marker 未被指定时： upload-id-marker 将被忽略 • 当 key-marker 被指定时： ◦ ObjectName 字母顺序大于 key-marker 的条目被列出 ◦ ObjectName 字母顺序等于 key-marker 且 UploadID 大于 upload-id-marker 的条目将被列出	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- headers	请求返回的头部信息	Object
- Bucket	分块上传的目标存储桶	String
- Encoding-Type	规定返回值的编码格式，合法值：url	String
- KeyMarker	列出条目从该 key 值开始	String
- UploadIdMarker	列出条目从该 UploadId 值开始	String
- NextKeyMarker	假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点	String
- NextUploadIdMarker	假如返回条目被截断，则返回 UploadId 就是下一个条目的起点	String
- MaxUploads	设置最大返回的条目数量，合法取值范围为1 - 1000	String
- IsTruncated	返回条目是否被截断，'true' 或者 'false'	String
- Prefix	对象键前缀匹配，限定返回中只包含指定前缀的对象键。	String
- Delimiter	定界符为一个分隔符号，用于对对象键进行分组。一般是传 `/` 。所有对象键从 Prefix 或从头（如未指定 Prefix）到首个 delimiter 之间相同部分的路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix	String
- CommonPrefixs	将 prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	ObjectArray
- - Prefix	显示具体的 Common Prefixs	String
- Upload	分块上传的信息集合	ObjectArray
- - Key	对象的名称，即对象键	String
- - UploadId	表示本次分块上传的 ID	String
- - StorageClass	用于表示分块的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE 等，更多存储类型请参见 存储类型概述 文档	String
- - Initiator	用于表示本次上传发起者的信息	Object
- - - DisplayName	上传发起者的名称	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- - - ID	上传发起者 ID，格式：`qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String
- - Owner	表示这些分块持有者的信息	Object
- - - DisplayName	分块持有者的名称	String
- - - ID	分块持有者 ID，格式：`qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String
- - Initiated	分块上传的起始时间	String

初始化分块上传

功能说明

Initiate Multipart Uploads 请求实现初始化分块上传，成功执行此请求后会返回 Upload ID ，用于后续的 Upload Part 请求。

使用示例

```
cos.multipartInit({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: 'exampleobject', /* 必须 */
}, function(err, data) {
console.log(err || data);
if (data) {
uploadId = data.UploadId;
}
});
```

参数说明

参数名	参数描述	类型	是
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

参数名	参数描述	类型	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
CacheControl	RFC 2616 中定义的缓存策略，将作为对象的元数据保存	String	否
ContentDisposition	RFC 2616 中定义的文件名称，将作为对象的元数据保存	String	否
ContentEncoding	RFC 2616 中定义的编码格式，将作为对象的元数据保存	String	否
ContentType	RFC 2616 中定义的内容类型（MIME），将作为对象的元数据保存	String	否
Expires	RFC 2616 中定义的过期时间，将作为对象的元数据保存	String	否
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String	否
GrantRead	赋予被授权者读取对象的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE 等，更多存储类型请参见 存储类型概述 文档。默认值：STANDARD	String	否
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据返回。大小限制2KB	String	否

回调函数说明

版权所有：腾讯云计算（北京）有限责任公司

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Bucket	分块上传的目标存储桶，格式为 BucketName-APPID，例如 examplebucket-1250000000	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
UploadId	在后续上传中使用的 ID	String

上传分块

功能说明

Upload Part 接口请求实现在初始化以后的分块上传，支持的块的数量为1 - 10000，块的大小为1MB - 5GB。

- 分块上传首先要进行初始化，用 Initiate Multipart Upload 接口初始化分块上传，得到一个 uploadId，该 ID 不但唯一标识这一分块数据，也标识了这分块数据在整个文件内的相对位置。
- 在每次请求 Upload Part 时候，需要携带 partNumber 和 uploadId，partNumber 为块的编号，支持乱序上传。
- 当传入 uploadId 和 partNumber 都相同的时候，后传入的块将覆盖之前传入的块。当 uploadId 不存在时会返回404错误，错误码为 NoSuchUpload。

使用示例

```
cos.multipartUpload({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'exampleobject', /* 必须 */
UploadId: 'exampleUploadId',
PartNumber: 1,
Body: fileObject
}, function(err, data) {
console.log(err || data);
if (data) {
eTag = data.ETag;
}
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	是
PartNumber	分块的编号	Number	是
UploadId	本次分块上传任务的编号	String	是
Body	上传文件分块的内容，可以为字符串或者 ArrayBuffer 对象	String/ArrayBuffer	是
Expect	RFC 2616 中定义的 HTTP 请求内容长度（字节）。当使用 `Expect: 100-continue` 时，在收到服务端确认后，才会发送请求内容	String	否
ContentMD5	RFC 1864中定义的经过 Base64 编码的128-bit 内容 MD5 校验值，此头部用来校验文件内容是否发生变化	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

查询已上传块

功能说明

List Parts 用来查询特定分块上传中的已上传的块，即列出指定 UploadId 所属的所有已上传成功的分块。

使用示例

```
cos.multipartListPart({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: 'exampleobject', /* 必须 */
UploadId: 'exampleUploadId', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
UploadId	标识本次分块上传的 ID，使用 Initiate Multipart Upload 接口初始化分块上传时得到的 UploadId	String	是
EncodingType	规定返回值的编码方式	String	否
MaxParts	单次返回最大的条目数量，默认为1000	String	否
PartNumberMarker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Bucket	分块上传的目标存储桶	String
- Encoding-type	规定返回值的编码方式	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- UploadId	标识本次分块上传的 ID，使用 Initiate Multipart Upload 接口初始化分块上传时得到的 UploadId	String
- Initiator	用来表示本次上传发起者的信息	Object
- - DisplayName	上传发起者的名称	String
- - ID	上传发起者 ID，格式：`qcs::cam::uin/<owneruin>:uin/<subuin>` 如果是主账号，<OwnerUin> 和 <SubUin> 是同一个值	String
- Owner	用来表示这些分块所有者的信息	Object
- - DisplayName	存储桶持有者的名称	String
- - ID	存储桶持有者 ID，一般为用户的 UIN	String
- StorageClass	用于表示这些分块的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE 等，更多存储类型请参见 存储类型概述 文档	String
- PartNumberMarker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
- NextPartNumberMarker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- MaxParts	单次返回最大的条目数量	String
- IsTruncated	返回条目是否被截断，'true' 或者 'false'	String
- Part	分块信息列表	ObjectArray
- - PartNumber	块的编号	String
- - LastModified	块最后修改时间	String
- - ETag	块的 MD5 算法校验值	String
- - Size	块大小，单位为 Byte	String

完成分块上传

功能说明

Complete Multipart Upload 接口请求用来实现完成整个分块上传。当使用 Upload Parts 上传完所有块以后，必须调用该 API 来完成整个文件的分块上传。在使用该 API 时，您必须在请求 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。
由于分块上传完后需要合并，而合并需要数分钟时间，因而当合并分块开始的时候，COS 就立即返回200的状态码，在合并的过程中，COS 会周期性的返回空格信息来保持连接活跃，直到合并完成，COS 会在 Body 中返回合并后块的内容。

- 当上传块小于1MB，在调用该 API 时，会返回400 EntityTooSmall。
- 当上传块编号不连续，在调用该 API 时，会返回400 InvalidPart。
- 当请求 Body 中的块信息未按序号从小到大排列，在调用该 API 时，会返回400 InvalidPartOrder。
- 当 UploadId 不存在，在调用该 API 时，会返回404 NoSuchUpload。

> 注意：
> 建议您及时完成分块上传或者舍弃分块上传，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

使用示例

```
cos.multipartComplete({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: 'exampleobject', /* 必须 */
```

```
UploadId: 'exampleUploadId', /* 必须 */
Parts: [
{PartNumber: 1, ETag: 'exampleETag'},
]
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
UploadId	上传任务编号	String	是
Parts	用来说明本次分块上传中块的信息列表	ObjectArray	是
- PartNumber	分块的编号	Number	是
- ETag	每个块文件的 MD5 算法校验值 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Location	创建对象的外网访问域名	String
- Bucket	分块上传的目标存储桶	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- ETag	合并后文件的唯一 ID，格式："uuid-<分块数>" 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880-3"`，**注意前后携带双引号**	String

终止分块上传

功能说明

Abort Multipart Upload 用来实现终止一个分块上传操作并删除已上传的块。当您调用 Abort Multipart Upload 时，如果有正在使用这个 UploadId 上传块的请求，则 Upload Parts 会返回失败。当该 UploadId 不存在时，会返回404 NoSuchUpload。

> 注意：
> 建议您及时完成分块上传或者舍弃分块上传，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

使用示例

```
cos.multipartAbort({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: 'exampleobject', /* 必须 */
UploadId: 'exampleUploadId' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
UploadId	标识本次分块上传的 ID，使用 Initiate Multipart Upload 接口初始化分块上传时得到的 UploadId	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

高级接口（推荐）

该类方法是对上面原生方法的封装，实现了分块上传的全过程，支持并发分块上传，支持断点续传，支持上传任务的取消，暂停和重新开始等。

高级上传

功能说明

Upload File 实现高级上传，传入参数 SliceSize 可以控制文件大小超出一个数值（默认1MB）时自动使用分块上传，否则使用简单上传。

使用示例

```
var uploadFile = function(file) {
cos.uploadFile({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: file.name, /* 必须 */
FilePath: file.path, /* 必须 */
FileSize: file.size, /* v1.4.3之前的版本必须，v1.4.3及以后的版本非必须 */
SliceSize: 1024 * 1024 * 5, /* 触发分块上传的阈值，超过5MB使用分块上传，非必须，按需调整，最小支持1MB */
onTaskReady: function(taskId) { /* 非必须 */
console.log(taskId);
},
onProgress: function (progressData) { /* 非必须 */
console.log(JSON.stringify(progressData));
},
onFileFinish: function (err, data, options) { /* 非必须 */
console.log(options.Key + '上传' + (err ? '失败' : '完成'));
},
}, function(err, data) {
console.log(err || data);
});
}

wx.chooseMessageFile({
count: 10,
type: 'all',
success: function(res) {
uploadFile(res.tempFiles[0]);
}
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

参数名	参数描述	类型	是否必填
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
FilePath	上传文件路径	String	是
FileSize	上传文件大小(v1.4.3之前的版本必须，v1.4.3及以后的版本非必须)	Number	是
SliceSize	表示文件大小超出一个数值时使用分块上传，单位 Byte，默认值1048576（1MB），小于等于该数值会使用 putObject 上传，大于该数值会使用 sliceUploadFile 上传	Number	否
AsyncLimit	分块的并发量，仅在触发分块上传时有效	Number	否
StorageClass	对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，更多存储类型请参见 存储类型概述	String	否
UploadAddMetaMd5	当上传时，给对象的元数据信息增加 x-cos-meta-md5 赋值为对象内容的 MD5 值，格式为 32 位小写字符串。例如：4d00d79b6733c9cc066584a02ed03410	String	否
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String	否
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String	否
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String	否
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	否
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String	否
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String	否
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
onTaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function	否
- taskId	上传任务的编号	String	否
onProgress	上传文件的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如，上传50%即为0.5	Number	否
onFileFinish	每个文件完成或错误回调	Function	否
- err	上传的错误信息	Object	否
- data	文件完成的信息	Object	否
- options	当前完成文件的参数信息	Object	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- headers	请求返回的头部信息	Object
- Location	上传完的文件访问地址	String
- Bucket	分块上传的目标存储桶,仅在触发分块上传时返回	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述，仅在触发分块上传时返回	String
- ETag	合并后文件的唯一 ID，格式："uuid-<分块数>" 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880-3"`，**注意前后携带双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

分块上传对象（断点续传）

功能说明

Slice Upload File 可用于实现文件的分块上传，适用于大文件上传。

使用示例

```
var sliceUploadFile = function (file) {
var key = file.name;
cos.sliceUploadFile({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 必须 */
Key: 'exampleobject', /* 必须 */
FilePath: file.path, /* 必须 */
FileSize: file.size, /* 非必须 */
CacheControl: 'max-age=7200', /* 非必须 */
Headers: { /* 非必须 */
aa: 123,
},
Query: { /* 非必须 */
bb: 123,
},
onTaskReady: function(taskId) { /* 非必须 */
console.log(taskId);
},
onHashProgress: function(info) { /* 非必须 */
console.log('check hash', JSON.stringify(info));
},
```

```
onProgress: function(info) { /* 非必须 */
console.log(JSON.stringify(info));
}
}, requestCallback);
};
wx.chooseMessageFile({
count: 10,
type: 'all',
success: function(res) {
sliceUploadFile(res.tempFiles[0]);
}
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
FilePath	上传文件路径	String	是
SliceSize	分块大小	Number	否
AsyncLimit	分块的并发量，仅在触发分块上传时有效	Number	否
StorageClass	对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE 等，更多存储类型请参见 存储类型概述 文档	String	否
CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String	否
ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String	否
ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String	否
ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	否
ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String	否
Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否
onTaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function	否
- taskId	上传任务的编号	String	否
onHashProgress	计算文件 MD5 值的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经校验的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的校验速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的校验百分比，以小数形式呈现，例如：校验50%即为0.5	Number	否
onProgress	上传文件的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如：上传50%即为0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Location	创建对象的外网访问域名	String
- Bucket	分块上传的目标存储桶	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- ETag	合并后文件的唯一 ID，格式：`"uuid-<分块数>"` 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880-3"`，**注意前后携带双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

批量上传

功能说明

方法一：
批量上传可以直接多次调用 putObject 和 sliceUploadFile，达到批量上传效果。通过实例化参数 FileParallelLimit 控制文件并发数，默认3个并发。

方法二：
可以调用 cos.uploadFiles 实现批量上传，传入参数 SliceSize 可以控制文件大小超出一个数值时使用分块上传。以下是 uploadFiles 方法说明。

方法原型

调用 uploadFiles 操作：

```
var uploadFiles = function(files) {
var fileList = files.map(function(file) {
return Object.assign(file, {
FilePath: file.path, /* 必须 */
```

```
FileSize: file.size, /* v1.4.3之前的版本必须，v1.4.3及以后的版本非必须 */
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION',/* 存储桶所在地域，必须字段 */
Key: file.name, /* 必须 */
onTaskReady: function(taskId) {
/* taskId可通过队列操作来取消上传cos.cancelTask(taskId)、停止上传cos.pauseTask(taskId)、重新开始上传cos.restartTask(taskId) */
console.log(taskId);
}
});
});
cos.uploadFiles({
files: fileList,
SliceSize: 1024 * 1024 * 10, /* 设置大于10MB采用分块上传，按需调整，最小支持1MB */
onProgress: function (info) {
var percent = parseInt(info.percent * 10000) / 100;
var speed = parseInt(info.speed / 1024 / 1024 * 100) / 100;
console.log('进度：' + percent + '%; 速度：' + speed + 'Mb/s;');
},
onFileFinish: function (err, data, options) {
console.log(options.Key + '上传' + (err ? '失败' : '完成'));
},
}, function (err, data) {
console.log(err || data);
});
}
wx.chooseMessageFile({
count: 10,
type: 'all',
success: function(res) {
uploadFiles(res.tempFiles);
}
});
```

参数说明

参数名	参数描述	类型	是否必填
files	文件列表，每一项是传给 putObject 和 sliceUploadFile 的参数对象	Object	是
- Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
- Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
- FilePath	上传文件路径	String	是
- FileSize	上传文件大小(v1.4.3之前的版本必须，v1.4.3及以后的版本非必须)	Number	是
- CacheControl	RFC 2616中定义的缓存策略，将作为对象的元数据保存	String	否
- ContentDisposition	RFC 2616中定义的文件名称，将作为对象的元数据保存	String	否
- ContentEncoding	RFC 2616中定义的编码格式，将作为对象的元数据保存	String	否
- ContentLength	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	否
- ContentType	RFC 2616中定义的内容类型（MIME），将作为对象的元数据保存	String	否
- Expires	RFC 2616中定义的过期时间，将作为对象的元数据保存	String	否
- Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否
- onTaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function	否
-- taskId	上传任务的编号	String	否
SliceSize	表示文件大小超出一个数值时使用分块上传，单位 Byte，默认值1048576（1MB），小于等于该数值会使用 putObject 上传，大于该数值会使用 sliceUploadFile 上传	Number	是
onProgress	所有任务 进度汇总计算出来的上传进度	String	是
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如：上传50%即为0.5	Number	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
onFileFinish	每个文件完成或错误回调	Function	否
- err	上传的错误信息	Object	否
- data	文件完成的信息	Object	否
- options	当前完成文件的参数信息	Object	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- files	每个文件的 error 或 data	ObjectArray
- - error	上传的错误信息	Object
- - data	文件完成的信息	Object
- - options	当前完成文件的参数信息	Object

上传队列

小程序 SDK 针对 putObject 发起的上传任务都有记录在队列中，队列相关方法如下。

1. var taskList = cos.getTaskList() 可以获取任务列表。
2. cos.pauseTask()、cos.restartTask()、cos.cancelTask() 操作任务。
3. cos.on('list-update', callback); 可以监听列表和进度变化。

完整的队列使用例子，请参见 demo-queue。

取消上传任务

根据 taskId 取消上传任务。

使用示例

```
var taskId = 'xxxxx'; /* 必须 */
cos.cancelTask(taskId);
```

参数说明

参数名	参数描述	类型	是否必填
taskId	文件上传任务的编号，在调用 putObject 方法时，其 TaskReady 回调会返回该上传任务的 taskId	String	是

暂停上传任务

根据 taskId 暂停上传任务。

使用示例

```
var taskId = 'xxxxx'; /* 必须 */
cos.pauseTask(taskId);
```

参数说明

参数名	参数描述	类型	是否必填
taskId	文件上传任务的编号，在调用 putObject 方法时，其 TaskReady 回调会返回该上传任务的 taskId	String	是

重启上传任务

根据 taskId 重新开始上传任务，可以用于开启用户手动停止的（调用 pauseTask 停止）或者因为上传错误而停止的上传任务。

使用示例

```
var taskId = 'xxxxx'; /* 必须 */
cos.restartTask(taskId);
```

参数说明

参数名	参数描述	类型	是否必填
taskId	文件上传任务的编号，在调用 putObject 方法时，其 TaskReady 回调将返回该上传任务的 taskId	String	是

版权所有：腾讯云计算（北京）有限责任公司

下载对象

最近更新时间：2022-03-30 12:44:57

简介

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地

功能说明

GET Object 接口请求可以获取存储桶里指定文件的内容，得到文件内容是字符串格式。

使用示例

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data.Body);
});
```

指定 Range 获取文件内容：

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Range: 'bytes=1-3', /* 非必须 */
}, function(err, data) {
console.log(err || data.Body);
});
```

下载对象（单链接限速）：

> 说明：
> 关于下载对象的限速说明，请参见 单链接限速。

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'exampleobject', /* 必须 */
Headers: {
'x-cos-traffic-limit': 819200, // 限速值设置范围为819200 - 838860800，即100KB/s - 10
0MB/s，如果超出该范围将返回400错误。
},
}, function(err, data) {
console.log(err || data.Body);
});
```

下载对象到本地（使用getObjectUrl实现）：

```
cos.getObjectUrl({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Sign: true
}, function (err, data) {
if (err) return console.log(err);
wx.downloadFile({
url: data.Url, // 需要加 url 的域名作为下载白名单
success (res) {
console.log(res.statusCode, res.tempFilePath);
},
fail: function (err) {
console.log(err);
},
});
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
ResponseContentType	设置响应头部中的 Content-Type 参数	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
ResponseContentLanguage	设置返回头部中的 Content-Language 参数	String	否
ResponseExpires	设置返回头部中的 Content-Expires 参数	String	否
ResponseCacheControl	设置返回头部中的 Cache-Control 参数	String	否
ResponseContentDisposition	设置返回头部中的 Content-Disposition 参数	String	否
ResponseContentEncoding	设置返回头部中的 Content-Encoding 参数	String	否
Range	RFC 2616 中定义的字节范围，范围值必须使用 bytes=first-last 格式，first 和 last 都是基于0开始的偏移量。例如 bytes=0-9 表示下载对象的开头10个字节的数据，如果不指定，则表示下载整个对象	String	否
IfModifiedSince	当对象在指定时间后被修改，则返回对应对象的元数据信息，否则返回304（not modified）	String	否
IfUnmodifiedSince	当对象在指定时间后未被修改，则返回对象，否则返回412（precondition failed）	String	否
IfMatch	当 ETag 与指定的内容一致，才返回对象，否则返回412（precondition failed）	String	否
IfNoneMatch	当 ETag 与指定的内容不一致，才返回对象，否则返回304（not modified）	String	否
VersionId	指定要下载的对象的版本 ID	String	否
onProgress	进度的回调函数，进度回调响应对象（progressData）属性如下	Function	否
- progressData.loaded	已经下载的对象部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个对象的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	对象的下载速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	对象下载的百分比，以小数形式呈现，例如：下载50%即为0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，304，403，404等	Number
- headers	请求返回的头部信息	Object
- CacheControl	RFC 2616 中定义的缓存指令，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- ContentDisposition	RFC 2616 中定义的文件名称，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- ContentEncoding	RFC 2616 中定义的编码格式，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- Expires	RFC 2616 中定义的缓存失效时间，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	String
- x-cos-storage-class	对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE，**注意：如果没有返回该头部，则说明文件存储类型为 STANDARD （标准存储）**。更多存储类型，请参见 存储类型概述	String
- x-cos-meta-*	用户自定义的元数据	String
- NotModified	如果请求时带有 IfModifiedSince 则返回该属性，如果文件未被修改，则为 true，否则为 false	Boolean
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String
- Body	返回的文件内容，默认为 String 形式	String

列出对象

最近更新时间：2022-02-18 16:08:51

简介

本文档提供关于列出对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象

功能说明

查询存储桶下的部分或者全部对象。

> 注意：
> 在 COS 中，列出的对象键为 project/，表示是一个内容为空的对象，展示方式上模拟了传统文件夹。相关介绍可参见 文件夹和目录。

使用示例

示例一：列出目录 a 的所有文件。

```
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a/', /* Prefix表示列出的object的key以prefix开始，非必须 */
}, function(err, data) {
console.log(err || data.Contents);
});
```

返回值格式：

```
{
"Name": "examplebucket-1250000000",
"Prefix": "",
"Marker": "a/",
"MaxKeys": "1000",
"Delimiter": "",
"IsTruncated": "false",
```

```
"Contents": [{
"Key": "a/3mb.zip",
"LastModified": "2018-10-18T07:08:03.000Z",
"ETag": "\"05a9a30179f3db7b63136f30aa6aacae-3\"",
"Size": "3145728",
"Owner": {
"ID": "1250000000",
"DisplayName": "1250000000"
},
"StorageClass": "STANDARD"
}],
"statusCode": 200,
"headers": {}
}
```


示例二：列出目录 a 的文件，不深度遍历。

```
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a/', /* Prefix表示列出的object的key以prefix开始，非必须 */
Delimiter: '/', /* Deliter表示分隔符，设置为/表示列出当前目录下的object，设置为空表示列出所有的object，非必须 */
}, function(err, data) {
console.log(err || data.CommonPrefixes);
});
```

返回值格式：


```
{
"Name": "examplebucket-1250000000",
"Prefix": "a/",
"Marker": "",
"MaxKeys": "1000",
"Delimiter": "/",
"IsTruncated": "false",
"CommonPrefixes": [{
"Prefix": "a/1/"
}],
"Contents": [{
"Key": "a/3mb.zip",
"LastModified": "2018-10-18T07:08:03.000Z",
"ETag": "\"05a9a30179f3db7b63136f30aa6aacae-3\"",
"Size": "3145728",
"Owner": {
"ID": "1250000000",
```


版权所有：腾讯云计算（北京）有限责任公司

```
"DisplayName": "1250000000"
},
"StorageClass": "STANDARD"
}],
"statusCode": 200,
"headers": {}
}
```

示例三：列出目录下所有文件。

```
var listFolder = function(marker) {
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a/', /* Prefix表示列出的object的key以prefix开始，非必须 */
Marker: marker,
MaxKeys: 1000,
}, function(err, data) {
if (err) {
return console.log('list error:', err);
} else {
console.log('list result:', data.Contents);
if (data.IsTruncated === 'true') listFolder(data.NextMarker);
else return console.log('list complete');
}
});
};
listFolder();
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Prefix	对象键前缀匹配，限定返回中只包含指定前缀的对象键	String	否
Delimiter	定界符。为一个分隔符号，用于对对象键进行分组。一般是传 `/` 。所有对象键从 Prefix 或从头（如未指定 Prefix）到首个 delimiter 之间相同部分的路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Marker	起始对象键标记，列出从 Marker 开始 MaxKeys 条目，默认顺序为 UTF-8 字典序	String	否
MaxKeys	单次返回最大的条目数量，默认1000，最大为1000	String	否
EncodingType	规定返回值的编码方式，可选值：url，代表返回的对象键为 URL 编码（百分号编码）后的值，例如“腾讯云”将被编码为 `%E8%85%BE%E8%AE%AF%E4%BA%91`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- headers	请求返回的头部信息	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- Name	存储桶的名称，格式为 BucketName-APPID，例如 examplebucket-1250000000	String
- Prefix	对象键前缀匹配，从该标记之后（不含）按照 UTF-8 字典序返回对象键条目	String
- Marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 Marker 开始	String
- MaxKeys	单次响应请求内返回结果的最大的条目数量	String
- Delimiter	定界符	String
- IsTruncated	响应请求条目是否被截断，值为 'true' 或者 'false'	String
- NextMarker	假如返回条目被截断，则返回 NextMarker 表示下一个条目的起点	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- CommonPrefixes	将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	ObjectArray
- - Prefix	单条 Common Prefix 的前缀	String
- EncodingType	返回值的编码方式，作用于 Delimiter，Marker，Prefix，NextMarker，Key	String
- Contents	对象元数据信息列表	ObjectArray
- - Key	对象键，即对象的名称	String
- - ETag	根据对象内容计算出的 MD5 算法校验值， 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- - Size	对象大小，单位 Byte	String
- - LastModified	对象最后修改时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	String
- - Owner	对象持有者信息	Object
- - - ID	对象持有者的完整 ID，格式为 `qcs::cam::uin/[OwnerUin]:uin/[OwnerUin]`， 例如 `qcs::cam::uin/100000000001:uin/100000000001`，其中100000000001为 uin	String
- - - DisplayName	对象持有者的名称	String
- - StorageClass	对象的存储类型，枚举值为：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE，详情参见 存储类型 文档	String

删除对象

最近更新时间：2021-12-27 14:18:29

简介

本文档提供关于对象的删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

删除单个对象

功能说明

DELETE Object 接口请求可以在 COS 的存储桶中将一个对象（Object）删除。该操作需要请求者对存储桶有 WRITE 权限。

使用示例

```
cos.deleteObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
VersionId	要删除的对象版本 ID 或 DeleteMarker 版本 ID	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，204，403，404等，**如果删除成功或者文件不存在则返回204或200，如果找不到指定的 Bucket，则返回404**	Number
- headers	请求返回的头部信息	Object

删除多个对象

功能说明

DELETE Multiple Objects 接口请求实现在指定存储桶中批量删除对象，单次请求最大支持批量删除1000个对象。对于响应结果，COS 提供 Verbose 和 Quiet 两种模式： Verbose 模式将返回每个对象的删除结果，Quiet 模式只返回报错的对象信息。

使用示例

删除多个文件：

```
cos.deleteMultipleObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Objects: [
{ Key: '1.jpg' }, /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
{ Key: '2.txt' }, /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
]
}, function(err, data) {
```

```
console.log(err || data);
});
```

按前缀删除多个对象（删除指定目录a下的文件）：

```
var deleteFiles = function (marker) {
cos.getBucket({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Prefix: 'a/',
Marker: marker,
MaxKeys: 1000,
}, function (listError, listResult) {
if (listError) return console.log('list error:', listError);
var nextMarker = listResult.NextMarker;
var objects = listResult.Contents.map(function (item) {
return {Key: item.Key}
});
cos.deleteMultipleObject({
Bucket: bucket,
Region: region,
Objects: objects,
}, function (delError, deleteResult) {
if (delError) {
console.log('delete error', delError);
console.log('delete stop');
} else {
console.log('delete result', deleteResult);
if (listResult.IsTruncated === 'true') deleteFiles(nextMarker);
else console.log('delete complete');
}
});
});
}
deleteFiles();
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

参数名	参数描述	类型	是否必填
Quiet	布尔值，这个值决定了是否启动 Quiet 模式。值为 true 启动 Quiet 模式，值为 false 则启动 Verbose 模式，默认值为 false	Boolean	否
Objects	要删除的对象列表	ObjectArray	是
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
- VersionId	要删除的对象版本 ID 或 DeleteMarker 版本 ID	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200，204，403，404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，204，403，404等	Number
- headers	请求返回的头部信息	Object
- Deleted	说明本次删除成功的对象信息列表	ObjectArray
- - Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- - VersionId	如果参数传入了 VersionId，返回也会带上 VersionId，表示刚操作的对象版本或 DeleteMarker 版本	String
- - DeleteMarker	如果开启了版本控制，并且参数没有 VersionId，本次删除不会真正抹去文件内容，只新增一个 DeleteMarker 代表可见的文件已删除，枚举值：true、false	String
- - DeleteMarkerVersionId	当返回的 DeleteMarker 为 true 时，返回刚新增的 DeleteMarker 的 VersionId	String

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- Error	说明本次删除失败的对象信息列表	ObjectArray
- - Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- - Code	删除失败的错误码	String
- - Message	删除错误信息	String

复制与移动对象

最近更新时间：2023-01-30 11:50:32

简介

本文档提供关于对象的复制、移动操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
PUT Object - Copy	设置对象复制（修改对象属性）	复制文件到目标路径

分块操作

API	操作名	操作描述
Initiate Multipart Upload	初始化分块复制	初始化分块复制任务
Upload Part - Copy	复制分块	将其他对象复制为一个分块
Complete Multipart Upload	完成分块复制	完成整个对象的分块复制

简单操作

复制对象

功能说明

PUT Object - Copy 请求创建一个已存在 COS 的对象的副本，即将一个对象从源路径（对象键）复制到目标路径（对象键）。在复制的过程中，对象元数据和访问控制列表（ACL）可以被修改。
用户可以通过该接口创建副本、修改对象元属性（源对象和目标文件的属性相同）、移动或重命名对象（先复制，再单独调用删除接口）。

> 注意：
> 建议对象大小1MB - 5GB，超过5GB的对象请使用高级接口复制对象 Slice Copy File。

使用示例

复制对象：

```
/* 把a/1.jpg复制一份到b/1.jpg */
cos.putObjectCopy({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'b/1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/a/1.jpg', /* 必须 */
/* CopySource中的Key含中文时，需要自行转义 */
// CopySource: `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/${encodeURIComponent('a/中文文件名.jpg')}`,
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
Region	存储桶所在地域，枚举值请参见 地域和访问域名
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述
CopySource	源对象 URL 路径，可以通过 URL 参数 ?versionId=<versionId> 参数指定指定历史版本
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，例如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**
GrantRead	赋予被授权者读取对象的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`

参数名	参数描述
GrantWrite	赋予被授权者写入对象的权限。格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账户授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`
MetadataDirective	是否拷贝元数据，枚举值：Copy，Replaced，默认值 Copy。假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制；假如标记为 Replaced，按 Header 信息修改元数据。**当目标路径和原路径一致，即用户试图修改元数据时，必须为 Replaced**
CopySourceIfModifiedSince	当对象在指定时间后被修改，则执行操作，否则返回412。**可与 CopySourceIfNoneMatch 一起使用，与其他条件联合使用返回冲突**
CopySourceIfUnmodifiedSince	当对象在指定时间后未被修改，则执行操作，否则返回412。**可与 CopySourceIfMatch 一起使用，与其他条件联合使用返回冲突**
CopySourceIfMatch	当对象的 Etag 和给定一致时，则执行操作，否则返回412。**可与 CopySourceIfUnmodifiedSince 一起使用，与其他条件联合使用返回冲突**
CopySourceIfNoneMatch	当对象的 Etag 和给定不一致时，则执行操作，否则返回412。**可与 CopySourceIfModifiedSince 一起使用，与其他条件联合使用返回冲突**
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA 等，默认值：STANDARD。更多存储类型请参见 存储类型概述
x-cos-meta-*	其他自定义的文件头部

回调函数说明

版权所有：腾讯云计算（北京）有限责任公司

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- ETag	文件的 MD5 算法校验值，例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- LastModified	返回对象最后被修改时间，例如2017-06-23T12:33:27.000Z	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

移动对象

功能说明

移动对象主要包括两个操作：复制源对象到目标位置，删除源对象。

由于 COS 通过存储桶名称（Bucket）和对象键（ObjectKey）来标识对象。移动对象也就意味着修改这个对象的标识，COS目前没有提供修改对象唯一标识名的单独接口，但是可以通过组合复制对象加上删除对象的基本操作，来达到修改对象标识的目的，从而实现移动对象。

使用示例

```
/* 把a/1.jpg移动到b/1.jpg */
/* 也可替换为高级复制cos.sliceCopyFile实现 */
cos.putObjectCopy({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'b/1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/a/1.jpg', /*
```

版权所有：腾讯云计算（北京）有限责任公司

```
必须 */
}, function(err, data) {
if (err) return console.log(err);
/* 删除a/1.jpg */
cos.deleteObject({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: 'a/1.jpg' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
});
```

修改对象存储类型

功能说明

修改对象存储类型通过复制对象（源和目的是同一个对象）时设置StorageClass实现。

使用示例

```
/* 把根目录下的1.jpg设置为归档存储类型 */
/* 也可替换为高级复制cos.sliceCopyFile实现 */
cos.putObjectCopy({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/1.jpg', /* 必
须 */
StorageClass: 'ARCHIVE', /* 设置为归档存储 */
}, function(err, data) {
console.log(err || data);
});
```

分块操作

初始化分块复制

功能说明

Initiate Multipart Uploads 请求实现初始化分块复制，成功执行此请求后会返回 Upload ID ，用于后续的 Upload Part - Copy 请求。

使用示例

```
cos.multipartInit({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
if (data) {
uploadId = data.UploadId;
}
});
```

参数说明

参数名	参数描述	类型	是否
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
CacheControl	RFC 2616 中定义的缓存策略，将作为对象的元数据保存	String	否
ContentDisposition	RFC 2616 中定义的文件名称，将作为对象的元数据保存	String	否
ContentEncoding	RFC 2616 中定义的编码格式，将作为对象的元数据保存	String	否
ContentType	RFC 2616 中定义的内容类型（MIME），将作为对象的元数据保存	String	否
Expires	RFC 2616 中定义的过期时间，将作为对象的元数据保存	String	否
ACL	定义对象的访问控制列表（ACL）属性，枚举值请参见 ACL 概述 文档中对象的预设 ACL 部分，如 default，private，public-read 等 **注意：如果您不需要进行对象 ACL 控制，请设置为 default 或者此项不进行设置，默认继承存储桶权限**	String	否

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否
GrantRead	赋予被授权者读取对象的权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否
GrantFullControl	赋予被授权者操作对象的所有权限，格式：id="[OwnerUin]"，可使用半角逗号（,）分隔多组被授权者： • 当需要给子账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<subuin>"` • 当需要给主账号授权时，`id="qcs::cam::uin/<owneruin>:uin/<owneruin>"` 例如 `'id="qcs::cam::uin/100000000001:uin/100000000001", id="qcs::cam::uin/100000000001:uin/100000000011"'`	String	否
StorageClass	设置对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE 等，更多存储类型请参见 存储类型概述 文档。默认值：STANDARD	String	否
x-cos-meta-*	允许用户自定义的头部信息，将作为对象的元数据返回。大小限制2KB	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Bucket	分块上传的目标存储桶，格式为 BucketName-APPID，例如 examplebucket-1250000000	String
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
UploadId	在后续上传中使用的 ID	String

复制分块

功能说明

Upload Part - Copy 接口请求实现将一个对象的分块内容从源路径复制到目标路径。

> 注意：
> 使用上传分块对象，必须先初始化分块上传。在初始化分块上传的响应中，会返回一个唯一的描述符（upload ID），您需要在分块上传请求中携带此 ID。

使用示例

```
cos.uploadPartCopy({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceObject',
/* 必须 */
UploadId: 'exampleUploadId', /* 必须 */
PartNumber: '1', /* 必须 */
}, function(err, data) {
console.log(err || data);
if (data) {
eTag = data.ETag;
}
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
CopySource	源对象 URL 路径，可以通过 URL 参数 ?versionId=<versionId> 参数指定指定历史版本	String	是
PartNumber	分块拷贝的块号	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
UploadId	使用上传分块文件，必须先初始化分块上传。在初始化分块上传的响应中，会返回一个唯一的描述符（upload ID），您需要在分块上传请求中携带此 ID	String	是
CopySourceRange	源对象的字节范围，范围值必须使用 bytes=first-last 格式，first 和 last 都是基于0开始的偏移量。例如 bytes=0-9 表示您希望拷贝源对象的开头10个字节的数据，如果不指定，则表示拷贝整个对象	String	否
CopySourceIfMatch	当对象的 Etag 和给定一致时，则执行操作，否则返回412，可与 x-cos-copy-source-If-Unmodified-Since 一起使用，与其他条件联合使用返回冲突	String	否
CopySourceIfNoneMatch	当对象的 Etag 和给定不一致时，则执行操作，否则返回412，可与 x-cos-copy-source-If-Modified-Since 一起使用，与其他条件联合使用返回冲突	String	否
CopySourceIfUnmodifiedSince	当对象在指定时间后未被修改，则执行操作，否则返回412，可与 x-cos-copy-source-If-Match 一起使用，与其他条件联合使用返回冲突	String	否
CopySourceIfModifiedSince	当对象在指定时间后被修改，则执行操作，否则返回412，可与 x-cos-copy-source-If-None-Match 一起使用，与其他条件联合使用返回冲突	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- ETag	文件的 MD5 算法校验值，例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- LastModified	返回对象最后修改时间，GMT 格式	String

完成分块复制

功能说明

Complete Multipart Upload 接口请求用来实现完成整个分块复制。当使用 Copy Parts 复制完所有块以后，必须调用该 API 来完成整个文件的分块复制。在使用该 API 时，您必须在请求 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。
由于分块复制完后需要合并，而合并需要数分钟时间，因而当合并分块开始的时候，COS 就立即返回200的状态码，在合并的过程中，COS 会周期性的返回空格信息来保持连接活跃，直到合并完成，COS 会在 Body 中返回合并后块的内容。

- 当上传块小于1MB，在调用该 API 时，会返回400 EntityTooSmall。
- 当上传块编号不连续，在调用该 API 时，会返回400 InvalidPart。
- 当请求 Body 中的块信息未按序号从小到大排列，在调用该 API 时，会返回400 InvalidPartOrder。
- 当 UploadId 不存在，在调用该 API 时，会返回404 NoSuchUpload。

> 注意：
> 建议您及时完成分块上传或者舍弃分块上传，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

使用示例

```
cos.multipartComplete({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
UploadId: 'exampleUploadId', /* 初始化分块任务时拿到，必须 */
Parts: [
{PartNumber: '1', ETag: 'exampleETag'},
]
}, function(err, data) {
```

```
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
UploadId	上传任务编号	String	是
Parts	用来说明本次分块上传中块的信息列表	ObjectArray	是
- PartNumber	分块的编号	String	是
- ETag	每个块文件的 MD5 算法校验值 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- Location	创建对象的外网访问域名	String
- Bucket	分块上传的目标存储桶	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- ETag	合并后文件的唯一 ID，格式："uuid-<分块数>" 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880-3"`，**注意前后携带双引号**	String

高级接口（推荐）

强烈建议您使用高级接口，该类方法是对上面原生方法的封装，实现了分块复制的全过程。

分块复制对象

功能说明

Slice Copy File 可用于实现通过分块复制将一个文件从源路径复制到目标路径。在拷贝的过程中，对象元属性和 ACL 可以被修改。用户可以通过该接口实现文件移动，文件重命名，修改文件属性和创建副本。

方法原型

调用 Slice Copy File 操作：

```
/* 把a/1.jpg复制到b/1.jpg */
cos.sliceCopyFile({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: 'b/1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
CopySource: 'sourcebucket-1250000000.cos.ap-guangzhou.myqcloud.com/a/1.jpg', /*
必须 */
onProgress:function (progressData) { /* 非必须 */
console.log(JSON.stringify(progressData));
}
},function (err,data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
CopySource	源对象 URL 路径，可以通过 URL 参数 ?versionId=<versionId> 参数指定指定历史版本	String	是
ChunkSize	分块复制时，每片的大小字节数，默认值 1048576（1MB）	Number	否
SliceSize	表示文件大小超出一个数值时使用分块复制，单位 Byte，默认值5G。小于等于该数值会使用 putObjectCopy 上传，大于该数值会使用 sliceCopyFile 上传	Number	否
onProgress	上传文件的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经上传的文件部分大小，以字节（Bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（Bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如：上传50%即为0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Location	创建对象的外网访问域名	String
- Bucket	分块上传的目标存储桶	String
- Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String
- ETag	合并后文件的 MD5 算法校验值 例如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

恢复归档对象

最近更新时间：2022-10-19 14:26:29

简介

本文档提供关于恢复归档对象操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
POST Object restore	恢复归档对象	将归档类型的对象取回访问

恢复归档对象

功能说明

POST Object restore 接口可以对一个通过 COS 归档为 archive 类型的对象进行恢复，恢复出的可读取对象是临时的，您可以设置需要保持可读，以及随后删除该临时副本的时间。您可以用 Days 参数来指定临时对象的过期时间，若超出该时间且期间您没有发起任何复制、延长等操作，该临时对象将被系统自动删除。临时对象仅为 archive 类型对象的副本，被归档的源对象在此期间将始终存在。

使用示例

```
cos.restoreObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
RestoreRequest: {
Days: 1,
CASJobParameters: {
Tier: 'Expedited'
}
},
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
RestoreRequest	用于恢复数据的容器	Object	是
- Days	设置临时副本的过期时间	Number	是
- CASJobParameters	归档存储工作参数的容器	Object	是
- - Tier	恢复数据时，Tier 可以指定为 COS 提供的三种恢复模式，分别为 Standard（标准取回模式，恢复任务在3 - 5小时内完成）、Expedited（快速取回模式，恢复任务在15分钟内可完成）以及 Bulk（批量取回模式，恢复任务在5 - 12小时内完成）	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

查询对象元数据

最近更新时间：2021-12-27 14:18:30

简介

本文档提供关于查询对象元数据操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

查询对象元数据

功能说明

查询对象的元数据信息。

使用示例

```
cos.headObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
IfModifiedSince	当对象在指定时间后被修改，则返回对应对象的元数据信息，否则返回304	String	否

版权所有：腾讯云计算（北京）有限责任公司

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，304等，如果在指定时间后未被修改，则返回304	Number
- headers	请求返回的头部信息	Object
- x-cos-object-type	用来表示对象是否可以被追加上传，枚举值：normal、appendable，默认 normal 不显示在返回中	String
- x-cos-storage-class	对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD 不显示在返回中。更多存储类型，请参见 存储类型概述	String
- x-cos-meta-*	用户自定义的 meta	String
- NotModified	对象是否在指定时间后未被修改	Boolean
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

检查对象是否存在

最近更新时间：2022-07-07 14:16:47

简介

本文档提供快捷查询存储桶中某个对象是否存在的示例代码。示例代码实际调用了 HEAD Object COS API，是该接口的简化版。

HEAD Object 除了检查对象是否存在，主要功能为返回对象元数据。如需查看包含了 HEAD Object 完整功能的 SDK 接口，请参见 查询对象元数据。

API	操作名	操作描述
HEAD Object	查询对象元数据	查询对象的元数据信息

功能说明

检查存储桶中是否存在某个对象,可通过封装cos.headObject方法实现。

使用示例

```
function doesObjectExist() {
cos.headObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
if (data) {
console.log('对象存在');
} else if (err.statusCode == 404) {
console.log('对象不存在');
} else if (err.statusCode == 403) {
console.log('没有该对象读权限');
}
});
}
```

参数说明

参数名	参数描述	类型	是否必填

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
IfModifiedSince	当对象在指定时间后被修改，返回对应对象的元数据信息，否则返回304	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200，304等，如果在指定时间后未被修改，则返回304	Number
- headers	请求返回的头部信息	Object
- x-cos-object-type	用来表示对象是否可以被追加上传，枚举值：normal、appendable，默认 normal 不显示在返回中	String
- x-cos-storage-class	对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE、DEEP_ARCHIVE 等，默认值：STANDARD 不显示在返回中。更多存储类型请参见 存储类型概述	String
- x-cos-meta-*	用户自定义的 meta	String
- NotModified	对象是否在指定时间后未被修改	Boolean

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查对象在上传过程中是否有损坏 例如 `"09cba091df696af91549de27b8e7d0f6"`，**注意：这里的 ETag 值字符串前后带有双引号**	String
- VersionId	在开启过版本控制的存储桶中上传对象返回对象的版本 ID，存储桶从未开启则不返回该参数	String

对象访问 URL

最近更新时间：2023-02-07 14:54:36

简介

本文档提供获取已上传到存储的对象访问 URL 的代码示例。

获取对象访问 URL

功能说明

查询对象访问的 URL，该接口不会判断对象是否真实存在。

> 说明：
>
> - 如何使生成的对象 URL 在浏览器中打开是预览，而不是下载：在获取的 url 后拼接参数 response-content-disposition=inline。
> - 如何使生成的对象 URL 在浏览器中打开是下载，而不是预览：在获取的 url 后拼接参数 response-content-disposition=attachment。
> - 下载时如何重命名文件：在获取的 URL 后拼接参数 filename=这里填自定义的文件名。
> - 若使用临时密钥生成预签名 URL，请确保申请临时密钥的 action 添加了 `"name/cos:GetObject"` 权限。

使用示例

获取在浏览器默认下载的 url：

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: true, /* 获取带签名的对象 URL */
},
function (err, data) {
if (err) return console.log(err);
/* url为对象访问 url */
var url = data.Url;
/* 复制 downloadUrl 的值到浏览器打开会自动触发下载 */
```

```
var downloadUrl =
data.Url +
(data.Url.indexOf('?') > -1 ? '&' : '?') +
'response-content-disposition=attachment'; // 补充强制下载的参数
}
);
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
Sign	是否返回带有签名的 Url，默认为 true，当对象为私有读时，获取到不带签名的 url 依然没有权限访问	Boolean	否
Protocol	可选填为 `http:` 或 `https:`，默认为 `http:`（带冒号）	String	否
Domain	存储桶访问域名，默认为 {BucketName-APPID}.cos.{Region}.myqcloud.com	String	否
Method	操作方法，例如 GET，POST，DELETE，HEAD 等 HTTP 方法，默认为 GET	String	否
Query	签名中要签入的请求参数，{key: 'val'} 的格式	Object	否
Headers	签名中要签入的请求头部，{key: 'val'} 的格式	Object	否
Expires	签名几秒后失效，默认为 900 秒	Number	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- Url	计算得到的 Url	String

版权所有：腾讯云计算（北京）有限责任公司

生成预签名链接

最近更新时间：2023-02-08 14:41:38

简介

小程序 SDK 提供获取对象 URL、获取请求预签名 URL 接口。

> 说明：
>
> - 建议用户使用临时密钥生成预签名，通过临时授权的方式进一步提高预签名上传、下载等请求的安全性。申请临时密钥时，请遵循 最小权限指引原则，防止泄漏目标存储桶或对象之外的资源。申请临时密钥的 action 需添加 `"name/cos:GetObject"` 权限。
> - 如果您一定要使用永久密钥来生成预签名，建议永久密钥的权限范围仅限于上传或下载操作，以规避风险。

计算签名

COS XML API 的请求里，私有资源操作都需要鉴权凭证 Authorization，用于判断当前请求是否合法。

鉴权凭证使用方式有两种：

1. 放在 header 参数里使用，字段名：authorization。
2. 放在 url 参数里使用，字段名：sign。

COS.getAuthorization 方法用于计算鉴权凭证（Authorization），用以验证请求合法性的签名信息。

> 注意：
> 该方法推荐只在前端调试时使用，项目上线不推荐使用前端计算签名的方法，有暴露密钥的风险。

使用示例

获取对象下载的鉴权凭证：

```
// SECRETID 和 SECRETKEY 请登录 https://console.intl.cloud.tencent.com/cam/capi 进行查看和管理
var Authorization = COS.getAuthorization({
```

```
SecretId: 'SECRETID', // 用户的 SecretId，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
SecretKey: 'SECRETKEY', // 用户的 SecretKey，建议使用子账号密钥，授权遵循最小权限指引，降低使用风险。子账号密钥获取可参见 https://cloud.tencent.com/document/product/598/37140
Method: 'get',
Key: 'exampleobject',
Expires: 60,
Query: {},
Headers: {},
});
```

参数说明

参数名	参数描述	类型	是否必填
SecretId	用户的 SecretId	String	是
SecretKey	用户的 SecretKey	String	是
Method	操作方法，例如 GET，POST，DELETE，HEAD 等 HTTP 方法	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，**如果请求操作是对文件的，则为文件名，且为必须参数**。如果操作是对于存储桶，则为空	String	否
Query	签名中要签入的请求参数，{key: 'val'} 的格式	Object	否
Headers	签名中要签入的请求头部，{key: 'val'} 的格式	Object	否
Expires	签名几秒后失效，默认为 900 秒	Number	否

返回值说明

返回值是计算得到的鉴权凭证字符串 authorization。

获取请求预签名 URL

下载请求示例

示例一：获取不带签名的对象的 Url

```
var url = cos.getObjectUrl({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
```

```
Sign: false,
});
```

示例二：获取带签名的对象的 Url

```
var url = cos.getObjectUrl({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
});
```

示例三：通过 callback 获取带签名 Url

> 说明：
> 如果签名过程是异步获取，需要通过 callback 获取带签名 Url。

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: false,
},
function (err, data) {
console.log(err || data.Url);
}
);
```

示例四：指定链接有效时间

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: true,
Expires: 3600, // 单位秒
},
function (err, data) {
console.log(err || data.Url);
}
);
```

版权所有：腾讯云计算（北京）有限责任公司

示例五：获取对象的 Url 并下载对象

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: true,
},
function (err, data) {
if (!err) return console.log(err);
wx.downloadFile({
url: data.Url, // 需要加 url 的域名作为下载白名单
success(res) {
console.log(res.statusCode, res.tempFilePath);
},
fail: function (err) {
console.log(err);
},
});
}
);
```

示例六：生成预签名 URL，并在签名中携带 Query 和 Header

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Sign: true,
/* 传入的请求参数需与实际请求相同，能够防止用户篡改此 HTTP 请求的参数 */
Query: {
'imageMogr2/thumbnail/200x/': '',
},
/* 传入的请求头部需包含在实际请求中，能够防止用户篡改签入此处的 HTTP 请求头部 */
Headers: {
host: 'xxx', /* 指定 host 访问，非指定的 host 访问会报错403 */
},
},
function (err, data) {
console.log(err || data.Url);
}
);
```

版权所有：腾讯云计算（北京）有限责任公司

上传请求示例

示例一：获取预签名 Put Object 上传 Url

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如 ap-beijing，必须字段 */
Key: '头像.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），支持中文，必须字段 */
Method: 'PUT',
Sign: true,
},
function (err, data) {
if (err) return console.log(err);
console.log(data.Url);
// 获取到 Url 后，小程序可以这样发起上传
wx.request({
url: data.Url, /* 预签名 url */
method: 'PUT', /* PUT 和 getObjectUrl 填写的 Method 对应 */
header: {},
dataType: 'text',
/* 可参考 demo 里的 putObject 方法获取 file https://github.com/tencentyun/cos-wx-sdk-v5/blob/master/demo/demo-sdk.js */
data: file, /* 小程序里选择的上传文件 */
success: function (response) {
console.log('上传成功', response);
},
fail: function (response) {
console.log('上传出错', response);
},
});
}
);
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名规则为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，**如果请求操作是对文件的，则为文件名，且为必须参数**。如果操作是对于存储桶，则为空	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Sign	是否返回带有签名的 Url，默认为 true	Boolean	否
Protocol	可选填为 `http:` 或 `https:`，默认为 `http:`（带冒号）	String	否
Domain	存储桶访问域名，默认为 {BucketName-APPID}.cos.{Region}.myqcloud.com	String	否
Method	操作方法，例如 GET，POST，DELETE，HEAD 等 HTTP 方法，默认为 GET	String	否
Query	参与签名计算的 query 参数对象，{key: 'val'} 的格式	Object	否
Headers	参与签名计算的 header 参数对象，{key: 'val'} 的格式	Object	否
Expires	签名几秒后失效，默认为 900 秒	Number	否

返回值说明

返回值是一个字符串，有以下两种情况：

1. 如果签名计算可以同步计算（例如，实例化传入了 SecretId 和 SecretKey），则默认返回带签名的 Url。
2. 否则返回不带签名的 Url。

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- Url	计算得到的 Url	String

预请求跨域配置

最近更新时间：2021-12-27 14:18:32

简介

本文档提供关于预请求跨域配置操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
Options Object	预请求跨域配置	用预请求来确认是否可以发送真正的跨域请求

功能说明

OPTIONS Object 接口实现对对象进行跨域访问配置的预请求。即在发送跨域请求之前会发送一个 OPTIONS 请求并带上特定的来源域，HTTP 方法和 HEADER 信息等给 COS，以决定是否可以发送真正的跨域请求。当 CORS 配置不存在时，请求返回403 Forbidden。**用户可以通过 PUT Bucket cors 接口来开启存储桶的 CORS 支持。**

使用示例

```
cos.optionsObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Origin: 'https://www.qq.com', /* 必须 */
AccessControlRequestMethod: 'PUT', /* 必须 */
AccessControlRequestHeaders: 'origin,accept,content-type' /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	存储桶的名称，命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型	是否必填
Origin	模拟跨域访问的请求来源域名	String	是
AccessControlRequestMethod	模拟跨域访问的请求 HTTP 方法	String	是
AccessControlRequestHeaders	模拟跨域访问的请求头部	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- headers	请求返回的头部信息	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- AccessControlAllowOrigin	模拟跨域访问的请求来源域名，中间用逗号间隔，当来源不允许的时候，此 Header 不返回，例如 `*`	String
- AccessControlAllowMethods	模拟跨域访问的请求 HTTP 方法，中间用逗号间隔，当请求方法不允许的时候，此 Header 不返回。例如 PUT，GET，POST，DELETE，HEAD	String
- AccessControlAllowHeaders	模拟跨域访问的请求头部，中间用逗号间隔，当模拟任何请求头部不允许的时候，此 Header 不返回该请求头部，例如 accept，content-type，origin，authorization	String
- AccessControlExposeHeaders	跨域支持返回头部，中间用逗号间隔。例如 ETag	String
- AccessControlMaxAge	设置 OPTIONS 请求得到结果的有效期，例如3600	String
- OptionsForbidden	OPTIONS 请求是否被禁止，如果返回的 HTTP 状态码为403，则为 true	Boolean

版权所有：腾讯云计算（北京）有限责任公司

单链接限速

最近更新时间：2021-12-27 14:18:32

简介

本文档提供关于调用上传下载接口时对链接进行限速。

使用说明

限速值设置范围为**819200 - 838860800**，单位默认为 bit/s，即100KB/s - 100MB/s，如果超出该范围将返回400错误。

> 说明：
> 关于下载对象的限速说明，请参见 单链接限速。

示例一：上传时对单链接限速

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
StorageClass: 'STANDARD',
Body: fileObject, // 上传文件对象
Headers: {
'x-cos-traffic-limit': 819200, // 限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，如果超出该范围将返回400错误。
},
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

示例二：下载时对单链接限速

```
cos.getObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
```

```
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Headers: {
'x-cos-traffic-limit': 819200, // 限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，如果超出该范围将返回400错误。
},
}, function(err, data) {
console.log(err || data.Body);
});
```

服务端加密

最近更新时间：2022-03-21 14:24:19

简介

本文档提供关于如何使用在上传对象时开启服务端加密。服务端加密的密钥分为三种：

- COS 托管加密密钥
- 客户提供的加密密钥
- KMS 托管加密密钥

使用 COS 托管加密密钥的服务端加密（SSE-COS）保护数据

由腾讯云 COS 托管主密钥和管理数据。COS 会帮助您在数据写入数据中心时自动加密，并在您取用该数据时自动解密。目前支持使用 COS 主密钥对数据进行 AES-256 加密。

小程序 SDK 通过调用 API 时传入 `ServerSideEncryption` 参数完成。

putObject代码示例

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: 'hello!',
ServerSideEncryption: 'AES256',
}, function(err, data) {
console.log(err || data);
});
```

postObject代码示例

```
cos.postObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
FilePath: tmpFilePath, // 使用小程序api获得，例如使用wx.chooseImage 选择文件得到的 tmpFilePath
ServerSideEncryption: 'AES256',
}, function(err, data) {
console.log(err || data);
});
```

版权所有：腾讯云计算（北京）有限责任公司

使用客户提供的加密密钥的服务端加密 （SSE-C）保护数据

加密密钥由用户自己提供，用户在上传对象时，COS 将使用用户提供的加密密钥对用户的数据进行 AES-256 加密。小程序 SDK 通过调用 API 时传入 `SSECustomerKey` 参数完成。

注意：

- 该加密所运行的服务需要使用 HTTPS 请求。
- customerKey：用户提供的密钥，传入一个32字节的字符串的base64编码，字符串支持数字、字母、字符的组合，不支持中文。
- 如果上传源文件（putObject）使用了该参数，那么在使用 headObject（查询）、getObject（下载）、multipartInit（初始化分块上传）、multipartUpload（上传分块）、putObjectCopy（复制）时对源对象操作的时候也要调用该方法。

putObject代码示例

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: 'hello!',
SSECustomerAlgorithm: 'AES256',
SSECustomerKey: 'MDEyMzQ1Njc4OUFCQ0RFRjAxMjM0NTY3ODlBQkNERUY',
SSECustomerKeyMD5: 'U5L61r7jcwdNvT7frmUG8g==',
}, function(err, data) {
console.log(err || data);
});
```

postObject代码示例

```
cos.postObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
FilePath: tmpFilePath, // 使用小程序api获得，例如使用wx.chooseImage 选择文件得到的 tmpFilePath
SSECustomerAlgorithm: 'AES256',
SSECustomerKey: 'MDEyMzQ1Njc4OUFCQ0RFRjAxMjM0NTY3ODlBQkNERUY',
SSECustomerKeyMD5: 'U5L61r7jcwdNvT7frmUG8g==',
}, function(err, data) {
console.log(err || data);
});
```

版权所有：腾讯云计算（北京）有限责任公司

使用 KMS 托管加密密钥的服务端加密（SSE-KMS）保护数据

SSE-KMS 加密即使用 KMS 托管密钥的服务端加密。KMS 是腾讯云推出的一款安全管理类服务，使用经过第三方认证的硬件安全模块 HSM（Hardware Security Module）来生成和保护密钥。它能够帮助用户轻松创建和管理密钥，满足用户多应用多业务的密钥管理需求以及满足监管和合规要求。关于如何开通 KMS 服务请参考：服务端加密概述。

putObject代码示例

```
cos.putObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Body: 'hello!',
ServerSideEncryption: 'cos/kms', /* 上传或复制对象（包括简单上传/复制与分块上传/复制）时必选，下载对象时不能指定此头部 */
// SSEKMSKeyId: 用于指定 KMS 的用户主密钥 CMK，如不指定，则使用 COS 默认创建的 CMK，更多详细信息可参见 SSE-KMS 加密
SSEKMSKeyId: 'xxxx', /* 这里填入自己的key，非必须 */
// SSEContext: 用于指定加密上下文，值为 JSON 格式加密上下文键值对的 Base64 编码。
SSEContext: 'eyJhIjoiYXNkZmEiLCJiIjoiMTIzMzIxIn0=', /* 这里指定加密上下文，需要自行使用Base64编码，非必须 */
}, function(err, data) {
console.log(err || data);
});
```

postObject代码示例

```
cos.postObject({
Bucket: 'examplebucket-1250000000', /* 填入您自己的存储桶，必须字段 */
Region: 'COS_REGION', /* 存储桶所在地域，例如ap-beijing，必须字段 */
Key: '1.jpg', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
FilePath: tmpFilePath, // 使用小程序api获得，例如使用wx.chooseImage 选择文件得到的 tmpFilePath
ServerSideEncryption: 'cos/kms', /* 上传或复制对象（包括简单上传/复制与分块上传/复制）时必选，下载对象时不能指定此头部 */
// SSEKMSKeyId: 用于指定 KMS 的用户主密钥 CMK，如不指定，则使用 COS 默认创建的 CMK，更多详细信息可参见 SSE-KMS 加密
SSEKMSKeyId: 'xxxx', /* 这里填入自己的key，非必须 */
// SSEContext: 用于指定加密上下文，值为 JSON 格式加密上下文键值对的 Base64 编码。
SSEContext: 'eyJhIjoiYXNkZmEiLCJiIjoiMTIzMzIxIn0=', /* 这里指定加密上下文，需要自行使用Base64编码，非必须 */
}, function(err, data) {
```

版权所有：腾讯云计算（北京）有限责任公司

```
console.log(err || data);
});
```

版权所有：腾讯云计算（北京）有限责任公司

异地容灾
版本控制

最近更新时间：2020-05-07 18:02:58

简介

本文档提供关于版本控制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

设置版本控制

功能说明

PUT Bucket versioning 接口实现启用或者暂停存储桶的版本控制功能。

1. 如果您从未在存储桶上启用过版本控制，则 GET Bucket versioning 请求不返回版本状态值。
2. 开启版本控制功能后，只能暂停，不能关闭。
3. 设置版本控制状态值为 Enabled 或者 Suspended，表示开启版本控制和暂停版本控制。
4. 设置存储桶的版本控制功能，您需要有存储桶的写权限。

请求示例

```
cos.putBucketVersioning({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
VersioningConfiguration: {
Status: "Enabled"
}
}, function (err, data) {
console.log(err || data);
});
```

版权所有：腾讯云计算（北京）有限责任公司

参数说明

参数名称	描述	类型	是否必填
Bucket	开启或暂停版本控制的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
VersioningConfiguration	定义存储桶的版本控制配置信息	Object	是
- Status	版本控制是否打开的状态，枚举值：Enabled、Suspended。 • Enabled 表示打开 • Suspended 表示暂停	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询版本控制

功能说明

查询存储桶的版本控制信息。

请求示例

```
cos.getBucketVersioning({
Bucket: 'examplebucket-1250000000', /* 必须 */
```

```
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function (err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询版本控制的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- VersioningConfiguration	存储桶的版本控制配置信息，若从未开启过，则为空对象 '{}'	Object
- - Status	版本控制是否打开的状态，枚举值：Enabled、Suspended	String

版权所有：腾讯云计算（北京）有限责任公司

跨地域复制

最近更新时间：2020-05-07 18:02:58

简介

本文档提供关于跨地域复制的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

设置跨地域复制

功能说明

设置存储桶的跨地域复制规则。

> 存储桶必须开启版本控制才能使用跨地域复制功能。

请求示例

```
cos.putBucketReplication({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
ReplicationConfiguration: { /* 必须 */
Role: "qcs::cam::uin/100000000001:uin/100000000001",
Rules: [{
ID: "1",
Status: "Enabled",
Prefix: "sync/",
Destination: {
Bucket: "qcs::cos:ap-beijing::destinationbucket-1250000000",
StorageClass: "Standard",
}
}]
```

```
}
}, function (err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型
Bucket	源存储桶，格式：BucketName-APPID	String
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String
ReplicationConfiguration	定义跨地域复制规则	Object
- Role	复制过程以什么角色的身份 格 式：`qcs::cam::uin/100000000001:uin/100000000011` 其中100000000001是主账号，100000000011是子账号	Object
- Rules	复制具体规则列表	ObjectArray
- - ID	标注具体 Rule 的 ID	String
- - Status	标识 Rule 是否生效，枚举值：Enabled、Disabled	String
- - Prefix	前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空	String
- - Destination	目标存储桶信息	Object
- - - Bucket	目标存储桶的名称 格式：`qcs::cos:<Region>::<BucketName-APPID>` 例如：`qcs::cos:ap-beijing::destinationbucket-1250000000`	Object
- - - StorageClass	复制后的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD	Object

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object

参数名称	描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询跨地域复制

功能说明

查询存储桶的跨地域复制规则。

请求示例

```
cos.getBucketReplication({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"ReplicationConfiguration": {
"Role": "qcs::cam::uin/100000000001:uin/100000000001",
"Rules": {
"ID": "1",
"Status": "Enabled",
"Prefix": "sync/",
"Destination": {
"Bucket": "qcs::cos:ap-beijing::destinationbucket-1250000000",
"StorageClass": "Standard"
}
}
},
"statusCode": 200,
```

版权所有：腾讯云计算（北京）有限责任公司

```
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询跨地域复制的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- ReplicationConfiguration	跨地域复制规则	Object
- - Role	复制过程以什么角色的身份 格 式：`qcs::cam::uin/100000000001:uin/100000000011` 其中100000000001是主账号，100000000011是子账号	Object
- - Rules	复制具体规则列表	ObjectArr
- - - ID	标注具体 Rule 的 ID	String
- - - Status	标识 Rule 是否生效，枚举值：Enabled、Disabled	String
- - - Prefix	前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空	String
- - - Destination	目标存储桶信息	Object
- - - - Bucket	目标存储桶的名称 格式：`qcs::cos:<Region>::<BucketName-APPID>` 例如：`qcs::cos:ap-beijing::destinationbucket-1250000000`	Object
- - - - StorageClass	复制后的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD	Object

版权所有：腾讯云计算（北京）有限责任公司

删除跨地域复制

功能说明

删除存储桶的跨地域复制规则。

请求示例

```
cos.deleteBucketReplication({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	删除跨地域复制的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

数据管理
生命周期

最近更新时间：2020-05-07 18:03:00

简介

本文档提供关于生命周期的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket lifecycle	设置生命周期	设置存储桶的生命周期管理的配置
GET Bucket lifecycle	查询生命周期	查询存储桶生命周期管理的配置
DELETE Bucket lifecycle	删除生命周期	删除存储桶生命周期管理的配置

设置生命周期

功能说明

设置指定存储桶的生命周期配置信息（PUT Bucket lifecycle）。

请求示例

示例一：上传30天后，当前版本沉降至低频存储。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "1",
"Status": "Enabled",
"Filter": {},
"Transition": {
"Days": "30",
"StorageClass": "STANDARD_IA"
}
}],
}, function(err, data) {
console.log(err || data);
});
```

示例二：指定目录前缀 `dir/`，上传90天后，当前版本沉降至归档存储。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "2",
"Filter": {
"Prefix": "dir/",
},
"Status": "Enabled",
"Transition": {
"Days": "90",
"StorageClass": "ARCHIVE"
}
}],
}, function(err, data) {
console.log(err || data);
});
```

示例三：上传180天后，清理过期文件删除标记。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "3",
"Status": "Enabled",
"Filter": {},
"Expiration": {
"Days": "180"
}
}],
}, function(err, data) {
console.log(err || data);
});
```

示例四：上传30天后，删除碎片。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "4",
"Status": "Enabled",
```

版权所有：腾讯云计算（北京）有限责任公司

```
"Filter": {},
"AbortIncompleteMultipartUpload": {
"DaysAfterInitiation": "30"
}
}],
}, function(err, data) {
console.log(err || data);
});
```

示例五：历史版本生成30天后沉降至归档存储。

```
cos.putBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Rules: [{
"ID": "5",
"Status": "Enabled",
"Filter": {},
"NoncurrentVersionTransition": {
"NoncurrentDays": "30",
"StorageClass": 'ARCHIVE'
}
}],
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置生命周期的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Rules	指定生命周期规则列表	ObjectArray	是
- ID	规则的唯一标识 ID	String	是
- Status	规则的开启状态，枚举值：Enabled、Disabled	String	是
- Filter	指定过滤条件	Object	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型	是否必填
- - Prefix	规则要匹配上的对象键前缀	String	否
- Transition	规则转换属性，表示对象存储级别何时转换为 Standard_IA 或 Archive	Object	否
- - Days	指明规则对应的动作在对象最后的修改日期过后多少天执行，该字段有效值是非负整数，最大支持3650天	Number	是
- - StorageClass	表示规则对应的动作执行后变更对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE，默认值：STANDARD	String	是
- NoncurrentVersionTransition	指明历史版本对象转换属性	ObjectArray	否
- - NoncurrentDays	指明历史版本对象在该值确定的生效天数后进行转换	Number	是
- - StorageClass	表示规则对应的动作执行后变更对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE	String	是
- Expiration	规则过期属性	Object	否
- - ExpiredObjectDeleteMarker	删除过期对象删除标记，枚举值 true，false，与 Days 不可并存	Boolean	是
- - Days	指明规则对应的动作在对象最后的修改日期过后多少天进行删除操作，与 ExpiredObjectDeleteMarker 不可并存	Number	是
- AbortIncompleteMultipartUpload	表示删除碎片	Object	否
- - DaysAfterInitiation	碎片在该值所确定的生效天数后被删除，按文件上传时间开始算，必须为正整数	Number	是

回调函数说明

```
function(err, data) { ... }
```

参数名	描述	类型

参数名	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询生命周期

功能说明

GET Bucket lifecycle 接口可以查询存储桶生命周期管理的配置。

请求示例

```
cos.getBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"Rules": [{
"ID": "1",
"Filter": "",
"Status": "Enabled",
"Transition": {
"Days": "30",
"StorageClass": "STANDARD_IA"
}
}, {
"ID": "2",
"Filter": {
```

```
"Prefix": "dir/"
},
"Status": "Enabled",
"Transition": {
"Days": "90",
"StorageClass": "ARCHIVE"
}
}, {
"ID": "3",
"Filter": "",
"Status": "Enabled",
"Expiration": {
"Days": "180"
}
}, {
"ID": "4",
"Filter": "",
"Status": "Enabled",
"AbortIncompleteMultipartUpload": {
"DaysAfterInitiation": "30"
}
}, {
"ID": "5",
"Filter": "",
"Status": "Enabled",
"NoncurrentVersionTransition:": {
"NoncurrentDays": "30",
"StorageClass": "ARCHIVE"
}
}],
"statusCode": 200,
"headers": {}
}
```


参数说明

参数名称	描述	类型	是否必填
Bucket	查询生命周期的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Rules	指定生命周期规则列表	ObjectArray
- - ID	规则的唯一标识 ID	String
- - Status	规则的开启状态，枚举值：Enabled、Disabled	String
- - Filter	指定过滤条件	Object
- - - Prefix	规则要匹配上的对象键前缀	String
- - Transition	规则转换属性，表示对象存储级别何时转换为 Standard_IA 或 Archive	ObjectArray
- - - Days	指明规则对应的动作在对象最后的修改日期过后多少天执行，该字段有效值是非负整数，最大支持3650天	Number
- - - StorageClass	表示规则对应的动作执行后变更对象的存储类型，枚举值：STANDARD、STANDARD_IA、ARCHIVE	String
- - NoncurrentVersionTransition	指明历史版本对象转换属性	ObjectArray
- - - NoncurrentDays	指明历史版本对象在该值确定的生效天数后进行转换	Number
- - - StorageClass	表示规则对应的动作执行后变更对象的存储级别，枚举值：STANDARD、STANDARD_IA、ARCHIVE	String
- - Expiration	规则过期属性	Object
- - - ExpiredObjectDeleteMarker	删除过期对象删除标记，枚举值 true，false，与 Days 不可并存	Boolean

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- - - Days	指明规则对应的动作在对象最后的修改日期过后多少天进行删除操作，与 ExpiredObjectDeleteMarker 不可并存	Number
- - AbortIncompleteMultipartUpload	表示删除碎片	Object
- - - DaysAfterInitiation	碎片在该值所确定的生效天数后被删除，按文件上传时间开始算，必须为正整数	Number

删除生命周期

功能说明

DELETE Bucket lifecycle 接口可以删除存储桶生命周期管理的配置。

请求示例

```
cos.deleteBucketLifecycle({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	被删除生命周期配置的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object

参数名称	描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

日志管理

最近更新时间：2020-10-10 11:19:08

简介

本文档提供关于日志管理的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket logging	设置日志管理	为源存储桶开启日志记录
GET Bucket logging	查询日志管理	查询源存储桶的日志配置信息

设置日志管理

功能说明

PUT Bucket Logging 用于为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中。

> ⚠ **注意：**
>
> 只有源存储桶拥有者才可进行该请求操作。

请求示例

示例1：设置将源存储桶 `sourcebucket-1250000000` 的日志信息投递到目标存储桶 `targetbucket-1250000000` 的 `bucket-logging-prefix/` 路径下。

```
cos.putBucketLogging({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
BucketLoggingStatus: { /* 必须 */
LoggingEnabled: {
TargetBucket: 'targetbucket-1250000000',
TargetPrefix: 'bucket-logging-prefix/'
}
}
}, function(err, data) {
console.log(err || data);
});
```

版权所有：腾讯云计算（北京）有限责任公司

示例2：关闭目标存储桶 `sourcebucket-1250000000` 的日志投递。

```
cos.putBucketLogging({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
BucketLoggingStatus: {} /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置日志管理的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
BucketLoggingStatus	说明日志记录配置的状态，如果为空字符串则表示关闭日志记录	Object	是
- LoggingEnabled	存储桶 logging 设置的具体信息，主要是目标存储桶	Object	否
- - TargetBucket	存放日志的目标存储桶，可以是同一个存储桶（但不推荐），或同一账户下、同一地域的存储桶	String	否
- - TargetPrefix	日志存放在目标存储桶的指定路径	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

- headers	请求返回的头部信息	Object

查询日志管理

功能说明

GET Bucket logging 用于查询源存储桶的日志配置信息。

> ⚠ **注意：**
>
> 只有源存储桶拥有者才可进行该请求操作。

请求示例

```
cos.getBucketLogging({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"BucketLoggingStatus": {
"LoggingEnabled": {
"TargetBucket": "targetbucket-1250000000",
"TargetPrefix": "bucket-logging-prefix/"
}
},
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询日志管理的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

版权所有：腾讯云计算（北京）有限责任公司

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- BucketLoggingStatus	说明日志记录配置的状态，如果为空字符串则意为关闭日志记录	Object
- - LoggingEnabled	存储桶 logging 设置的具体信息，主要是目标存储桶	Object
- - - TargetBucket	存放日志的目标存储桶，可以是同一个存储桶（但不推荐），或同一账户下、同一地域的存储桶	String
- - - TargetPrefix	日志存放在目标存储桶的指定路径	String

存储桶标签

最近更新时间：2020-05-07 18:03:01

简介

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket tagging	设置存储桶标签	为已存在的存储桶设置标签
GET Bucket tagging	查询存储桶标签	查询指定存储桶下已有的存储桶标签
DELETE Bucket tagging	删除存储桶标签	删除指定的存储桶标签

设置存储桶标签

功能说明

为已存在的存储桶设置标签。

请求示例

```
cos.putBucketTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Tagging: {
Tags: [
{"Key": "k1", "Value": "v1"},
{"Key": "k2", "Value": "v2"}
]
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置存储桶标签的存储桶，格式：BucketName-APPID	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型	是否必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Tagging	标签信息	Object	是
- Tags	标签信息	ObjectArray	是
- - Key	标签名	String	是
- - Value	标签值	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询存储桶标签

功能说明

查询指定存储桶下已有的存储桶标签。

请求示例

```
cos.getBucketTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"Tags": [
{"Key": "k1", "Value": "v1"},
{"Key": "k2", "Value": "v2"}
],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询存储桶标签的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
- Tags	标签信息	ObjectArray
- - Key	标签名	String
- - Value	标签值	String

版权所有：腾讯云计算（北京）有限责任公司

删除存储桶标签

功能说明

删除指定的存储桶标签。

请求示例

```
cos.deleteBucketTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	删除存储桶标签的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

对象标签

最近更新时间：2022-10-25 14:26:56

简介

本文档提供关于对象标签的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Object tagging	设置对象标签	为已存在的对象设置标签
GET Object tagging	查询对象标签	查询指定对象下已有的对象标签
DELETE Object tagging	删除对象标签	删除指定的对象标签

设置对象标签

功能说明

为已存在的对象设置标签。

请求示例

```
cos.putObjectTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.png', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
Tags: [
{"Key": "k1", "Value": "v1"},
{"Key": "k2", "Value": "v2"}
],
// VersionId: 'MTg0NDUwOTMyODM1MDg2MTA1xxx' /* 对象版本 ID，非必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	设置对象标签的存储桶，格式：BucketName-APPID	String	是

参数名	参数描述	类型	是否必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是
Tags	标签信息	ObjectArray	是
- Key	标签名	String	是
- Value	标签值	String	是
VersionId	当启用版本控制时，指定要操作的对象版本 ID，如不指定则将添加标签到最新版本的对象	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询对象标签

功能说明

查询指定对象下已有的对象标签。

请求示例

```
cos.getObjectTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
```

```
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.png', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"Tags": [
{"Key": "k1", "Value": "v1"},
{"Key": "k2", "Value": "v2"}
],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名	参数描述	类型	是否必填
Bucket	查询对象标签的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- headers	请求返回的头部信息	Object
- Tags	标签信息	ObjectArray
- - Key	标签名	String
- - Value	标签值	String

删除对象标签

功能说明

删除指定的对象标签。

请求示例

```
cos.deleteObjectTagging({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Key: '1.png', /* 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	描述	类型	是否必填
Bucket	删除对象标签的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Key	对象键（Object 的名称），对象在存储桶中的唯一标识，详情请参见 对象概述	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

静态网站

最近更新时间：2020-10-10 11:19:09

简介

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket website	设置静态网站	为已存在的存储桶设置静态网站配置信息
GET Bucket website	查询静态网站	查询指定存储桶的静态网站配置信息
DELETE Bucket website	删除静态网站	删除指定存储桶的静态网站配置信息

设置静态网站

功能说明

为已存在的存储桶配置静态网站。

请求示例

```
cos.putBucketWebsite({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
WebsiteConfiguration: {
IndexDocument: {
Suffix: "index.html"
},
ErrorDocument: {
Key: "error.html"
},
RedirectAllRequestsTo: {
Protocol: "https"
},
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置静态网站的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
WebsiteConfiguration	静态网站配置，包括索引文档、错误文档、协议转换和重定向规则	Object	是
- IndexDocument	索引文档	Object	是
- - Suffix	指定索引文档	String	是
- ErrorDocument	错误文档	Object	否
- - Key	指定通用错误返回	String	否
- RedirectAllRequestsTo	重定向所有请求	Object	否
- - Protocol	指定全站重定向的协议，只能设置为 https	String	否
- RoutingRules	设置重定向规则，最多设置100条	ObjectArray	否
- - Condition	指定重定向发生的条件，前缀匹配重定向和错误码重定向只能指定一个	Object	否
- - - HttpErrorCodeReturnedEquals	指定重定向错误码，只支持配置4XX返回码，优先级高于 ErrorDocument	String	否
- - - KeyPrefixEquals	指定前缀重定向的路径，替换指定的 folder/	String	否
- - Redirect	指定满足重定向 conditon 时重定向的具体替换规则	Object	否
- - - ReplaceKeyWith	替换整个 Key 为指定的内容	String	否
- - - ReplaceKeyPrefixWith	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型

err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询静态网站配置

功能说明

查询与指定存储桶关联的静态网站配置信息。

请求示例

```
cos.getBucketWebsite({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"WebsiteConfiguration": {
"IndexDocument": {
"Suffix": "index.html"
},
"ErrorDocument": {
"Key": "error.html"
},
"RedirectAllRequestsTo": {
"Protocol": "https"
},
},
"statusCode": 200,
```

```
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询静态网站配置的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
- WebsiteConfiguration	静态网站配置，包括索引文档、错误文档、协议转换和重定向规则	Object
- - IndexDocument	索引文档	Object
- - - Suffix	指定索引文档	String
- - ErrorDocument	错误文档	Object
- - - Key	指定通用错误返回	String
- - RedirectAllRequestsTo	重定向所有请求	Object
- - - Protocol	指定全站重定向的协议，只能设置为 https	String
- - RoutingRules	设置重定向规则，最多设置100条	ObjectArray

- - - Condition	指定重定向发生的条件，前缀匹配重定向和错误码重定向只能指定一个	Object
- - - - HttpErrorCodeReturnedEquals	指定重定向错误码，只支持配置4XX 返回码，优先级高于 ErrorDocument	String
- - - - KeyPrefixEquals	指定前缀重定向的路径，替换指定的 folder/	String
- - - Redirect	指定满足重定向 conditon 时重定向的具体替换规则	Object
- - - - ReplaceKeyWith	替换整个 Key 为指定的内容	String
- - - - ReplaceKeyPrefixWith	替换匹配到的前缀为指定的内容，Conditon 为 KeyPrefixEquals 才可设置	String

删除静态网站配置

功能说明

删除指定存储桶中的静态网站配置。

请求示例

```
cos.deleteBucketWebsite({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	被删除静态网站配置的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型

err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

清单

最近更新时间：2020-10-10 11:19:10

简介

本文档提供关于清单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket inventory	设置清单任务	设置存储桶的清单任务
GET Bucket inventory	查询清单任务	查询存储桶的清单任务
DELETE Bucket inventory	删除清单任务	删除存储桶的清单任务

设置清单任务

功能说明

PUT Bucket inventory 用于在存储桶中创建清单任务，您可以对清单任务命名后，使用该请求创建清单任务，详细信息请参见 清单功能概述。

⚠ 注意：

- COS 支持在每个存储桶中创建最多1000条清单任务。
- 您必须在目标存储桶中写入存储桶策略，以供 COS 将清单任务的结果文件写入该存储桶中。
- 调用该请求时，请确保您有足够的权限对存储桶的清单任务进行操作。存储桶所有者默认拥有该权限，若您无该项权限，请先向存储桶所有者申请该项操作的权限。

请求示例

示例1：创建清单任务，并设置服务端加密。

```
cos.putBucketInventory({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
Id: 'inventory_test', /* 必须 */
InventoryConfiguration: {
Id: 'inventory_test',
IsEnabled: 'true',
```

```
Destination: {
COSBucketDestination: {
Format: 'CSV',
AccountId: '100000000001',
Bucket: 'qcs::cos:ap-beijing::targetbucket-1250000000',
Prefix: 'inventory_test_prefix',
Encryption: {
SSECOS: ''
}
}
},
Schedule: {
Frequency: 'Daily'
},
Filter: {
Prefix: 'filter_prefix'
},
IncludedObjectVersions: 'All',
OptionalFields: [
'Size',
'LastModifiedDate',
'StorageClass',
'ETag'
]
}
}, function(err, data) {
console.log(err || data);
});
```

示例2：创建清单任务，不设置服务端加密。

```
cos.putBucketInventory({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
Id: 'inventory_test', /* 必须 */
InventoryConfiguration: {
Id: 'inventory_test',
IsEnabled: 'true',
Destination: {
COSBucketDestination: {
Format: 'CSV',
AccountId: '100000000001',
Bucket: 'qcs::cos:ap-beijing::targetbucket-1250000000',
Prefix: 'inventory_test_prefix'
}
},
Schedule: {
```

```
Frequency: 'Daily'
},
Filter: {
Prefix: 'filter_prefix'
},
IncludedObjectVersions: 'All',
OptionalFields: [
'Size',
'LastModifiedDate',
'StorageClass',
'ETag'
]
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置清单任务的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Id	清单任务的名称。 缺省值：None 合法字符：`a-z，A-Z，0-9，-，_，.`	String	是
InventoryConfiguration	包含清单的配置参数	Object	是
- Id	清单任务的名称。 缺省值：None 合法字符：`a-z，A-Z，0-9，-，_，.`	String	是
- IsEnabled	清单是否启用的标识： • 如果设置为 true，清单功能将生效 • 如果设置为 false，将不生成任何清单	String	是
- IncludedObjectVersions	是否在清单中包含对象版本： • 如果设置为 All，清单中将会包含所有对象版本，并在清单中增加 VersionId，IsLatest，DeleteMarker 这几个字段 • 如果设置为 Current，则清单中不包含对象版本信息	String	是
- Filter	筛选待分析对象。清单功能将分析符合 Filter 中设置的前缀的对象	Object	否

版权所有：腾讯云计算（北京）有限责任公司

- - Prefix	需要分析的对象的前缀	String	否
- OptionalFields	清单结果中可选包含的分析项目名称，可选字段包括：Size，LastModifiedDate，StorageClass，ETag，IsMultipartUploaded，ReplicationStatus	Array	否
- Schedule	配置清单任务周期	Object	是
- - Frequency	清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly	String	是
- Destination	描述存放清单结果的信息	Object	是
- - COSBucketDestination	清单结果导出后存放的存储桶信息	Object	是
- - - Bucket	清单分析结果的存储桶名	String	是
- - - AccountId	存储桶的所有者 ID，例如100000000001	String	否
- - - Prefix	清单分析结果的前缀。如果您指定了清单投递的前缀，COS 后端会自动在您指定的前缀后边加上 `/` 。 例如，您指定了 'Prefix' 作为前缀，则 COS 后端投递的清单报告路径为 `Prefix/inventory_report`	String	否
- - - Format	清单分析结果的文件形式，可选项为 CSV 格式	String	是
- - - Encryption	为清单结果提供服务端加密的选项	Object	否
- - - - SSECOS	COS 托管密钥的加密方式，无需填充，设置为空字符串	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

版权所有：腾讯云计算（北京）有限责任公司

- headers	请求返回的头部信息	Object

查询清单任务

功能说明

GET Bucket inventory 用于查询存储桶中用户的清单任务信息。用户在发起该请求时，需要用户提供清单任务的名称，发起该请求时需获得请求签名，表明该请求已获得许可。有关清单的详细介绍，请参见 清单功能概述。

⚠ 注意：

- 调用该请求时，请确保您有足够的权限对存储桶的清单任务进行操作。
- 存储桶所有者默认拥有该权限，若您无该项权限，请先向存储桶所有者申请该项操作的权限。

请求示例

```
cos.getBucketInventory({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
Id: 'inventory_test' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"InventoryConfiguration": {
"Id":"inventory_test",
"IsEnabled":"true",
"Destination":{
"COSBucketDestination": {
"Format":"CSV",
"AccountId":"100000000001",
"Bucket":"qcs::cos:ap-beijing::targetbucket-1250000000",
"Prefix":"inventory_test_prefix",
"Encryption": {
"SSECOS":""
}
}
},
"Schedule": {
```

```
"Frequency":"Daily"
},
"Filter": {
"Prefix":"filter_prefix"
},
"IncludedObjectVersions":"All",
"OptionalFields": [
"Size",
"LastModifiedDate",
"StorageClass",
"ETag"
]
}
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询清单任务的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
Id	清单任务的名称。 缺省值：None 合法字符：`a-z，A-Z，0-9，-，_，.`	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

- headers	请求返回的头部信息	Object
- InventoryConfiguration	包含清单的配置参数	Object
- - Id	清单任务的名称。 缺省值：None 合法字符：`a-z，A-Z，0-9，-，_，.`	String
- - IsEnabled	清单是否启用的标识： • 如果设置为 true，清单功能将生效 • 如果设置为 false，将不生成任何清单	String
- - IncludedObjectVersions	是否在清单中包含对象版本： • 如果设置为 All，清单中将会包含所有对象版本，并在清单中增加 VersionId，IsLatest，DeleteMarker 这几个字段 • 如果设置为 Current，则清单中不包含对象版本信息	String
- - Filter	筛选待分析对象。清单功能将分析符合 Filter 中设置的前缀的对象	Object
- - - Prefix	需要分析的对象的前缀	String
- - OptionalFields	清单结果中可选包含的分析项目名称，可选字段包括：Size，LastModifiedDate，StorageClass，ETag，IsMultipartUploaded，ReplicationStatus	Array
- - Schedule	配置清单任务周期	Object
- - - Frequency	清单任务周期，可选项为按日或者按周，枚举值：Daily、Weekly	String
- - Destination	描述存放清单结果的信息	Object
- - - COSBucketDestination	清单结果导出后存放的存储桶信息	Object
- - - - Bucket	清单分析结果的存储桶名	String
- - - - AccountId	存储桶的所有者 ID，例如100000000001	String
- - - - Prefix	清单分析结果的前缀。如果您指定了清单投递的前缀，COS后端会自动在您指定的前缀后边加上 `/` 。 例如，您指定了 'Prefix' 作为前缀，则 COS 后端投递的清单报告路径为 `Prefix/inventory_report`	String
- - - - Format	清单分析结果的文件形式，可选项为 CSV 格式	String
- - - - Encryption	为清单结果提供服务端加密的选项	Object
- - - - - SSECOS	COS 托管密钥的加密方式，无需填充，设置为空字符串	String

版权所有：腾讯云计算（北京）有限责任公司

删除清单任务

功能说明

DELETE Bucket inventory 用于删除存储桶中指定的清单任务，用户需提供待删除的清单任务的名称。
有关清单的详细介绍，请参见 清单功能概述。

⚠ 注意：

- 调用该请求时，请确保您有足够的权限对存储桶的清单任务进行操作。
- 存储桶所有者默认拥有该权限，如您无该项权限，请先向存储桶所有者申请该项操作的权限。

请求示例

```
cos.deleteBucketInventory({
Bucket: 'sourcebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
Id: 'inventory_test' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	被删除清单任务的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域。枚举值请参见 地域和访问域名	String	是
Id	清单任务的名称。缺省值：None 合法字符：`a-z, A-Z, 0-9, -, _, .`	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number

- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

访问管理
跨域访问

最近更新时间：2020-05-07 18:03:03

简介

本文档提供关于跨域访问的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域名访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域名访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域名访问配置信息

设置跨域配置

1. 如果要在前端修改跨域访问配置，需要该存储桶本身支持跨域，您可以在控制台进行跨域访问配置，详情请参见 快速入门 文档。
2. 在修改跨域访问配置时，请注意不要影响到当前的 Origin 下的跨域请求。

功能说明

设置存储桶的跨域名访问权限，您可以通过传入 XML 格式的配置文件来实现配置，文件大小限制为64KB。默认情况下，存储桶的持有者直接有权限使用该 API 接口，存储桶持有者也可以将权限授予其他用户。

请求示例

```
cos.putBucketCors({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
CORSRules: [{
"AllowedOrigin": ["*"],
"AllowedMethod": ["GET", "POST", "PUT", "DELETE", "HEAD"],
"AllowedHeader": ["*"],
"ExposeHeader": ["ETag", "x-cos-acl", "x-cos-version-id", "x-cos-delete-marker",
"x-cos-server-side-encryption"],
```

```
"MaxAgeSeconds": "5"
}]
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置跨域配置的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
CORSRules	说明跨域资源共享配置的所有信息列表	ObjectArray	否
- ID	配置规则的 ID	String	否
- AllowedMethods	允许的 HTTP 操作，枚举值：GET、PUT、HEAD、POST、DELETE 等	StringArray	是
- AllowedOrigins	允许的访问来源，支持通配符 `*`，格式为：`协议://域名[:端口]`，例如：`http://www.qq.com`	StringArray	是
- AllowedHeaders	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 `*`	StringArray	否
- ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	StringArray	否
- MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询跨域配置

功能说明

GET Bucket cors 接口实现查询存储桶的跨域名访问配置信息。（CORS 是一个 W3C 标准，全称是"跨域资源共享"（Cross-origin Resource Sharing））。默认情况下，存储桶的持有者直接有权限使用该 API 接口，存储桶持有者也可以将权限授予其他用户。

请求示例

```
cos.getBucketCors({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"CORSRules": [{
"MaxAgeSeconds": "5",
"AllowedOrigins": ["*"],
"AllowedHeaders": ["*"],
"AllowedMethods": ["GET", "POST", "PUT", "DELETE", "HEAD"],
"ExposeHeaders": ["ETag", "Content-Length", "x-cos-acl", "x-cos-version-id", "x-cos-request-id", "x-cos-delete-marker", "x-cos-server-side-encryption"]
}],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询跨域配置的存储桶，格式：BucketName-APPID	String	是

版权所有：腾讯云计算（北京）有限责任公司

参数名称	描述	类型	是否必填
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- CORSRules	说明跨域资源共享配置的所有信息列表	ObjectArray
- - AllowedMethods	允许的 HTTP 操作，枚举值：GET、PUT、HEAD、POST、DELETE	StringArray
- - AllowedOrigins	允许的访问来源，支持通配符 `*` ，格式为： `协议://域名[:端口]` 例如： `http://www.qq.com`	StringArray
- - AllowedHeaders	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 `*`	StringArray
- - ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	StringArray
- - MaxAgeSeconds	设置 OPTIONS 跨域信息缓存秒数	String
- - ID	配置规则的 ID	String

删除跨域配置

功能说明

删除存储桶的跨域名访问配置信息。

1. 删除当前存储桶的跨域访问配置信息，会导致所有请求跨域失败，请谨慎操作。
2. 不推荐在浏览器端使用该方法。

请求示例

```
cos.deleteBucketCors({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	被删除跨域配置的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

存储桶策略

最近更新时间：2020-10-15 11:59:46

简介

本文档提供关于存储桶策略的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket policy	设置存储桶策略	设置指定存储桶的权限策略
GET Bucket policy	查询存储桶策略	查询指定存储桶的权限策略
DELETE Bucket policy	删除存储桶策略	删除指定存储桶的权限策略

设置存储桶策略

功能说明

设置指定存储桶的权限策略。

请求示例

```
cos.putBucketPolicy({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
Policy: {
"version": "2.0",
"Statement": [{
"Effect": "allow",
"Principal": {
"qcs": ["qcs::cam::uin/100000000001:uin/100000000001"]
},
"Action": [
"name/cos:PutObject",
"name/cos:InitiateMultipartUpload",
"name/cos:ListMultipartUploads",
"name/cos:ListParts",
"name/cos:UploadPart",
"name/cos:CompleteMultipartUpload"
],
"Resource": ["qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"],
```

```
}]
},
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	必
Bucket	设置存储桶策略的存储桶，格式：BucketName-APPID	String	必
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	必
Policy	权限策略，详情请参见 访问管理策略语法	Object	必
- version	版本号，固定2.0	String	必
- statement	权限策略生命列表	ObjectArray	必
- - effect	效力，枚举值：allow、deny	String	必
- - principal	身份信息	ObjectArray	必
- - - qcs	身份信息标识字符串 格 式：`qcs::cam::uin/100000000001:uin/100000000011` 其中100000000001 是主账号，100000000011是子账号	String	必
- - action	策略生效的相关 Action 列表，支持通配符 `*`	StringArray	必
- - resource	相关的资源标识字符串列表 格式：`qcs::cos:<Region>:uid/<AppId>:<ShortBucketName>/*` 例如：`qcs::cos:ap-beijing:uid/1250000000:examplebucket/*`	StringArray	必
- - condition	约束条件，可以不填，具体说明请参见 condition 说明	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为	Object

	空，详情请参见 错误码 文档	
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询存储桶策略

功能说明

查询指定存储桶的权限策略。

请求示例

```
cos.getBucketPolicy({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"Policy": {
"version": "2.0",
"Statement": [{
"Action": [
"name/cos:PutObject",
"name/cos:InitiateMultipartUpload",
"name/cos:ListMultipartUploads",
"name/cos:ListParts",
"name/cos:UploadPart",
"name/cos:CompleteMultipartUpload"
],
"Effect": "allow",
"Principal": {
"qcs": ["qcs::cam::uin/100000000001:uin/100000000001"]
},
```

```
"Resource": ["qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*"],
"Sid": "costs-1539833197000000307620-46518-39"
}]
},
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询存储桶权限策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- Policy	权限策略，详情请参见 访问管理策略语法	Object
- - version	版本号，固定2.0	String
- - Statement	权限策略生命列表	ObjectArray
- - - effect	效力，枚举值：allow、deny	String
- - - principal	身份信息	ObjectArray
- - - - qcs	身份信息标识字符串 格式：`qcs::cam::uin/100000000001:uin/100000000011` 其中100000000001是主账号，100000000011是子账号	String
- - - action	策略生效的相关 Action 列表，支持通配符 `*`	StringArray
- - - resource	相关的资源标识字符串列表 格式：`qcs::cos:<Region>:uid/&ltAppId>:<ShortBucketName>/*`	StringArray

	例如：`qcs::cos:ap-beijing:uid/1250000000:examplebucket/*`	
- - - condition	约束条件，可以不填，具体说明请参见 condition 说明	ObjectArray

删除存储桶策略

功能说明

DELETE Bucket policy 请求删除指定存储桶的权限策略。

> 只有 Bucket 所有者有权限发起该请求。如果权限策略不存在，将返回204 No Content。

请求示例

```
cos.deleteBucketPolicy({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	被删除存储桶权限策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

自定义域名

最近更新时间：2021-03-23 17:28:35

简介

本文档提供关于自定义域名的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket domain	设置自定义域名	设置存储桶的自定义域名信息
GET Bucket domain	查询自定义域名	查询存储桶的自定义域名信息
DELETE Bucket domain	删除自定义域名	删除存储桶的自定义域名信息

设置自定义域名

功能说明

为已存在的存储桶绑定自定义域名。

请求示例

```
cos.putBucketDomain({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
DomainRule: [{
Status: "DISABLED",
Name: "www.example.com",
Type: "REST"
},
{
Status: "DISABLED",
Name: "www.example.net",
Type: "WEBSITE",
}]
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置自定义域名的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
DomainRule	自定义域名配置	Object	是
- Status	域名上线/下线状态，枚举值： ENABLED、DISABLED	String	是
- Name	用户的自定义域名	String	是
- Type	绑定的源站类型，枚举值：REST、WEBSITE	String	是
- ForcedReplacement	替换已存在的配置，枚举值：CNAME、TXT。填写则强制校验域名所有权后，再下发配置。	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询自定义域名

功能说明

查询与指定存储桶关联的自定义域名信息。

请求示例

```
cos.getBucketDomain({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"DomainRule": [{
"Status": "DISABLED",
"Name": "www.example.com",
"Type": "REST"
}, {
"Status": "DISABLED",
"Name": "www.example.net",
"Type": "WEBSITE"
}],
"statusCode": 200,
"headers": {}
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询自定义域名的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
- DomainRule	自定义域名配置	Object
- - Status	域名上线/下线状态，枚举值： ENABLED、DISABLED	String
- - Name	用户的自定义域名	String
- - Type	绑定的源站类型，枚举值：REST、WEBSITE	String
- - ForcedReplacement	替换已存在的配置，枚举值：CNAME、TXT。填写则强制校验域名所有权后，再下发配置。	String

删除自定义域名

功能说明

删除指定存储桶中的自定义域名配置。

请求示例

```
cos.deleteBucketDomain({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'ap-beijing', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	被删除自定义域名的存储桶，格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object

版权所有：腾讯云计算（北京）有限责任公司

防盗链

最近更新时间：2022-01-10 12:36:25

简介

本文档提供关于存储桶 Referer 白名单或者黑名单的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
PUT Bucket referer	设置存储桶 Referer	设置存储桶 Referer 白名单或者黑名单
GET Bucket referer	查询存储桶 Referer	查询存储桶 Referer 白名单或者黑名单

设置存储桶 Referer

功能说明

设置指定存储桶的 Referer 白名单或者黑名单（PUT Bucket referer）。

请求示例

```
cos.putBucketReferer({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
RefererConfiguration: {
Status: 'Enabled',
RefererType: 'White-List',
DomainList: {
Domains: [
'*.qq.com',
'*.qcloud.com',
]
},
EmptyReferConfiguration: 'Allow',
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名称	描述	类型	是否必填
Bucket	设置存储桶策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是
RefererConfiguration	防盗链配置信息，详情请参见 PUT Bucket referer	Object	是
- Status	是否开启防盗链，枚举值：Enabled、Disabled	String	是
- RefererType	防盗链类型，枚举值：Black-List、White-List	String	是
- DomainList	生效域名列表， 支持多个域名且为前缀匹配， 支持带端口的域名和 IP， 支持通配符*，做二级域名或多级域名的通配	Object	是
- - Domains	生效域名，支持两种写法：单条'*.qq.com'或多条['*.qq.com', '*.qcloud.com']	String\Array	是
- EmptyReferConfiguration	是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，例如200、403、404等	Number
- headers	请求返回的头部信息	Object

查询存储桶 Referer

功能说明

查询指定存储桶 Referer 白名单或者黑名单（GET Bucket referer）。

请求示例

```
cos.getBucketReferer({
Bucket: 'examplebucket-1250000000', /* 必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
}, function(err, data) {
console.log(err || data);
});
```

返回示例

```
{
"RefererConfiguration": {
"Status": "Enabled",
"RefererType": "White-List",
"DomainList": {
"Domains": [
"*.qq.com",
"*.qcloud.com"
]
},
"EmptyReferConfiguration": "Allow"
},
"statusCode": 200,
"headers": {},
}
```

参数说明

参数名称	描述	类型	是否必填
Bucket	查询存储桶权限策略的存储桶，格式：BucketName-APPID	String	是
Region	存储桶所在地域，枚举值请参见 地域和访问域名	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- RefererConfiguration	防盗链配置信息，详情请参见 GET Bucket referer	Object
- - Status	是否开启防盗链，枚举值：Enabled、Disabled	String
- - RefererType	防盗链类型，枚举值：Black-List、White-List	String
- - DomainList	生效域名列表， 支持多个域名且为前缀匹配， 支持带端口的域名和IP， 支持通配符*，做二级域名或多级域名的通配	Object
- - - Domains	生效域名	Array
- - EmptyReferConfiguration	是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny	String

数据校验
CRC64 校验

最近更新时间：2023-12-11 18:00:19

简介

数据在客户端和服务器间传输时可能会出现错误，COS 除了可以通过 MD5 和自定义属性 验证数据完整性外，还可以通过 CRC64 检验码来进行数据校验。
COS 会对新上传的对象进行 CRC64 计算，并将结果作为对象的属性进行存储，随后在返回的响应头部中携带 x-cos-hash-crc64ecma，该头部表示上传对象的 CRC64 值，根据 ECMA-182标准 计算得到。对于 CRC64 特性上线前就已经存在于 COS 的对象，COS 不会对其计算 CRC64 值，所以获取此类对象时不会返回其 CRC64 值。

操作说明

目前支持 CRC64 的 API 如下：
简单上传接口
PUT Object 和 POST Object：用户可在返回的响应头中获得文件 CRC64 校验值。
分块上传接口
Upload Part：用户可以根据 COS 返回的 CRC64 值与本地计算的数值进行比较验证。
Complete Multipart Upload：如果每个分块都有 CRC64 属性，则会返回整个对象的 CRC64 值，如果某些分块不具备 CRC64 值，则不返回。
执行 Upload Part - Copy 时，会返回对应的 CRC64 值。
执行 PUT Object - Copy 时，如果源对象存在 CRC64 值，则返回 CRC64，否则不返回。
执行 HEAD Object 和 GET Object 时，如果对象存在 CRC64，则返回。用户可以根据 COS 返回的 CRC64 值和本地计算的 CRC64 进行比较验证。

SDK 示例

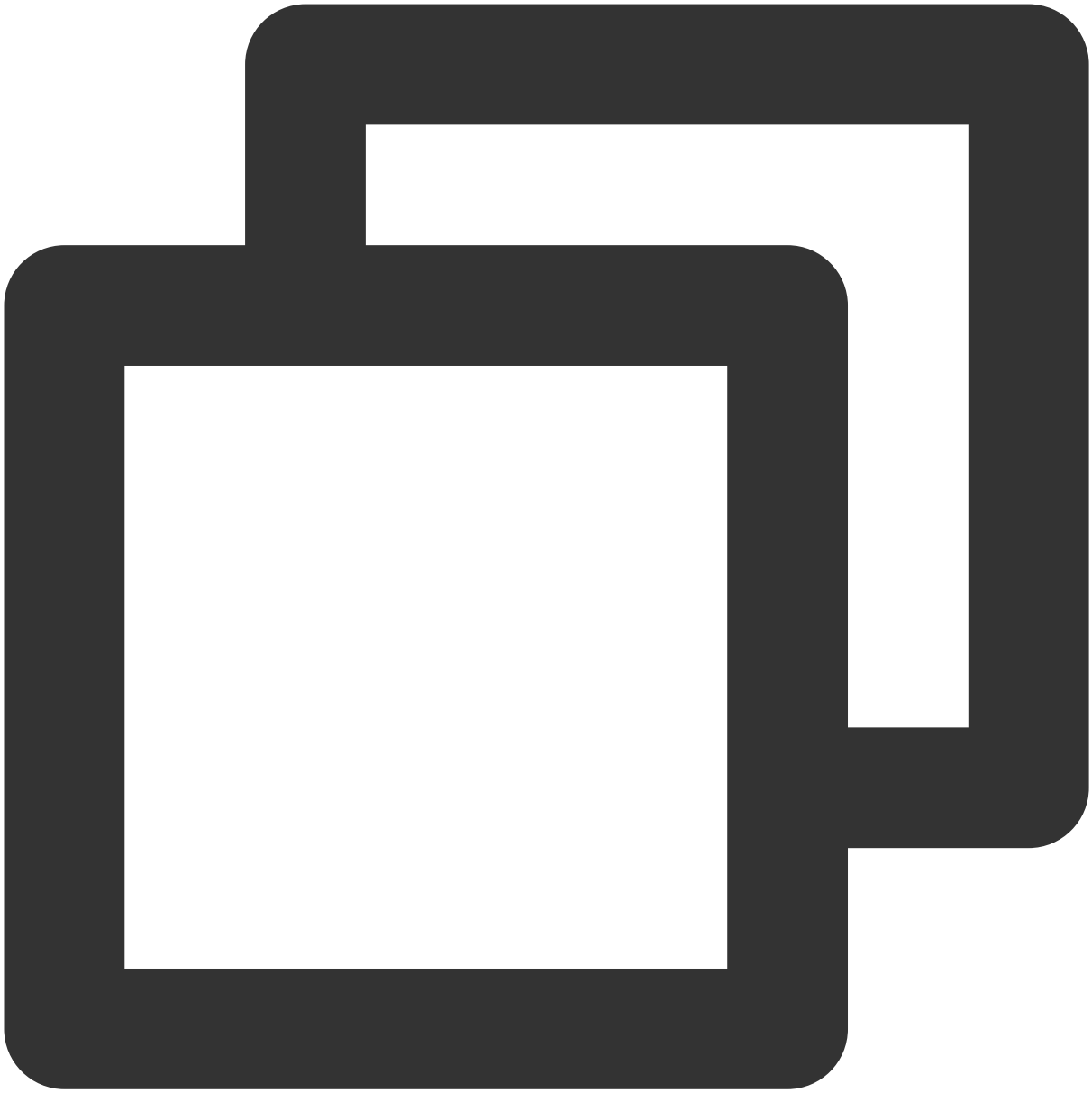
功能说明

用于在对象上传和下载的时候对对象数据做 CRC64 一致性校验。

请求示例

这里只用简单上传举例，其它接口也是同样使用方式。

版权所有：腾讯云计算（北京）有限责任公司



```
cos.putObject({
    Bucket: 'examplebucket-1250000000', /* 必须 */
    Region: 'COS_REGION',     /* 存储桶所在地域，必须字段 */
    Key: 'exampleobject',              /* 必须 */
    StorageClass: 'STANDARD',
    Body: fileObject, // 上传文件对象
    onProgress: function(progressData) {
        console.log(JSON.stringify(progressData));
    }
}, function(err, data) {
    if (err) {
```

```
        console.log(err);
    } else {
        // 需要给Expose-Headers添加字段x-cos-hash-crc64ecma字段才可正确返回
        // 参考文档：https://cloud.tencent.com/document/product/436/13318
        var crc64 = data.headers['x-cos-hash-crc64ecma'];
        console.log(crc64);
    }
});
```

版权所有：腾讯云计算（北京）有限责任公司

内容审核
图片审核

最近更新时间：2022-12-09 16:37:37

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于图片审核的 API 概览和 SDK 示例代码。

> 注意：
> COS 小程序 SDK 版本需要大于等于 v1.1.1。

API	操作描述
图片同步审核	对对象存储（Cloud Object Storage，COS）存量数据进行涉黄、违法违规以及广告引导类图片的扫描
图片批量审核	对多个图片进行批量审核

图片同步审核

功能说明

图片审核的存量扫描功能通过借助数据万象的持久化处理接口，实现对 COS 存量数据的涉黄、违法违规以及广告引导类图片的扫描。

版权所有：腾讯云计算（北京）有限责任公司

使用示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function getImageAuditing() {
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'GET',
Key: '1.png', /* 存储桶内的图片文件，必须字段 */
Query: {
'ci-process': 'sensitive-content-recognition', /** 固定值，必须 */
'biz-type': '', /** 审核类型，非必须 */
'detect-type': 'porn,ads', /** 审核策略，不填写则使用默认策略，非必须 */
'detect-url': '', /** 审核任意公网可访问的图片链接，非必须 */
'interval': 5, /** 审核 GIF 动图时，每隔interval帧截取一帧，非必须 */
'max-frames': 5, /** 审核 GIF 动图时，最大截帧数，非必须 */
'large-image-detect': '0', /** 是否需要压缩图片后再审核，非必须 */
'dataid': '123', /** 自定义图片标识，非必须 */
},
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

Query 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
biz-type	Query	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置公共审核策略。	String	否
detect-type	Query	审核类型审核类型，porn（涉黄识别）、ads（广告识别）。可选择多种识别类型用 `,` 连接。	String	否
object-key	Query	图片文件在 bucket 中的位置。	String	是

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
detect-url	Query	您可以通过填写 `detect-url` 审核任意公网可访问的图片链接。 • 不填写 `detect-url` 时，后台会默认审核 `ObjectKey` 。 • 填写 `detect-url` 时，后台会审核 `detect-url` 链接，无需再填写 `ObjectKey` 、 `detect-url` 。示例：http://www.example.com/abc.jpg。	String	否
interval	Query	审核 GIF 动图时，可使用该参数进行截帧配置，代表截帧的间隔。例如值设为5，则表示从第1帧开始截取，每隔5帧截取一帧，默认值5。	Int	否
max-frames	Query	针对 GIF 动图审核的最大截帧数量，需大于0。例如值设为5，则表示最大截取5帧，默认值为5。	Int	否
large-image-detect	Query	对于超过大小限制的图片是否进行压缩后再审核，取值为：0（不压缩），1（压缩）。默认为0。 注：压缩最大支持32M的图片，且会收取压缩费用。	String	否
dataid	Query	图片标识，该字段在结果中返回原始内容，长度限制为512字节。	String	否

返回结果说明

详情请参见 图片同步审核。

图片批量审核

功能说明

图片批量审核接口为同步请求方式，您可以通过本接口对多个图片文件进行内容审核。该接口属于 POST 请求。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function postImagesAuditing() {
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
```

版权所有：腾讯云计算（北京）有限责任公司

```
var url = 'https://' + host + '/image/auditing';
var body = COS.util.json2xml({
Request: {
Input: [{
Object: '1.png', /* 需要审核的图片，存储桶里的路径 */
}, {
Object: 'a/6.png', /* 需要审核的图片，存储桶里的路径 */
}],
Conf: {
BizType: '', /* 不填写代表默认策略 */
DetectType: 'Porn' /* 只审核是否涉黄 */
}
}
});
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'POST',
Url: url,
Key: '/image/auditing', /** 固定值，必须 */
ContentType: 'application/xml', /** 固定值，必须 */
Body: body
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

request的数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Request	无	图片批量审核的具体配置项。	Container	是

Container 类型 Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的内容，如有多个图片，请传入多个 Input 结构。	Container Array	是
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下，使用其中一种：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	存储在 COS 存储桶中的图片文件名称，例如在目录 test 中的文件 image.jpg，则文件名称为 test/image.jpg。Object 和 Url 只能选择其中一种。	String	否
Url	Request.Input	图片文件的链接地址，例如 http://a-1250000.cos.ap-shanghai.myqcloud.com/image.jpg。Object 和 Url 只能选择其中一种。	String	否
Interval	Request.Input	截帧频率，GIF 图检测专用，默认值为5，表示从第一帧（包含）开始每隔5帧截取一帧。	Int	否
MaxFrames	Request.Input	最大截帧数量，GIF 图检测专用，默认值为5，表示只截取 GIF 的5帧图片进行审核，必须大于0。	Int	否
DataId	Request.Input	图片标识，该字段在结果中返回原始内容，长度限制为512字节。	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以 `,` 分隔，例如：Porn,Ads。	String	是
BizType	Request.Conf	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置公共审核策略。	String	否

注意：

- 通过 Object 进行审核为内网操作，不会产生额外的外网流量。
- 通过 Url 进行审核，会产生图片所在源站对应的外网流量。

返回结果说明

详情请参见 图片批量审核。

视频审核

最近更新时间：2022-12-09 16:37:37

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于视频审核的 API 概览和 SDK 示例代码。

> 注意：
> COS 小程序 SDK 版本需要大于等于 v1.1.1。

API	操作描述
提交视频审核任务	用于提交一个视频审核任务
查询视频审核任务结果	用于查询指定的视频审核任务

提交视频审核任务

功能说明

用于提交一个视频审核任务。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
```

```
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function postVideoAuditing() {
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/video/auditing';
var body = COS.util.json2xml({
Request: {
Input: {
Object: '1.mp4', /* 需要审核的视频文件，存储桶里的路径 */
},
Conf: {
BizType: '',
DetectType: 'Porn',
Snapshot: {
Count: 1000, // 视频截帧数量
},
DetectContent: 1, // 是否审核视频声音，0-只审核视频不审核声音；1-审核视频+声音
}
}
});
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'POST',
Url: url,
Key: '/video/auditing', /** 固定值，必须 */
ContentType: 'application/xml', /** 固定值，必须 */
Body: body
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的视频。	Container	是
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下：

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	当前 COS 存储桶中的视频文件名称，例如在目录 test 中的文件 video.mp4，则文件名称为 test/video.mp4。	String	是
Url	Request.Input	视频文件的链接地址，例如 http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/test.mp4。Object 和 Url 只能选择其中一种。	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。	String	否
Snapshot	Request.Conf	视频画面的审核通过视频截帧能力截取出一定量的截图，通过对截图逐一审核而实现的，该参数用于指定视频截帧的配置。	Container	是
Callback	Request.Conf	回调地址，以 `http://` 或者 `https://` 开头的地址。	String	否
CallbackVersion	Request.Conf	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。	String	否
BizType	Request.Conf	审核策略，不带审核策略时使用默认策略。	String	否
DetectContent	Request.Conf	用于指定是否审核视频声音，当值为0时：表示只审核视频画面截图；值为1时：表示同时审核视频画面截图和视频声音。默认值为0。	Integer	否

Container 类型 Snapshot 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Mode	Request.Conf.Snapshot	截帧模式。Interval 表示间隔模式；Average 表示平均模式；Fps 表示固定帧率模式。 • Interval 模式：TimeInterval，Count 参数生效。当设置 Count，未设置 TimeInterval 时，表示截取所有帧，共 Count 张图片。 • Average 模式：Count 参数生效。表示整个视频，按平均间隔截取共 Count 张图片。 • Fps 模式：TimeInterval 表示每秒截取多少帧，Count 表示共截取多少帧。	String	否
Count	Request.Conf.Snapshot	视频截帧数量，范围为(0, 10000]。	String	否
TimeInterval	Request.Conf.Snapshot	视频截帧频率，范围为(0, 60]，单位为秒，支持 float 格式，执行精度精确到毫秒。	Float	否

返回结果说明

详情请参见 提交视频审核任务。

查询视频审核任务结果

功能说明

用于查询视频审核任务执行情况和结果。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function getVideoAuditingResult() {
var jobId = 'av14d9ca15af3a11eca0d6525400d88xxx'; // jobId可以通过提交视频审核任务返回
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/video/auditing/' + jobId;
cos.request({
```

```
Bucket: config.Bucket,
Region: config.Region,
Method: 'GET',
Key: '/video/auditing/' + jobId,
Url: url,
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID。	String	是

返回结果说明

详情请参见 查询视频审核任务结果。

音频审核

最近更新时间：2022-12-09 16:37:37

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于音频审核的 API 概览和 SDK 示例代码。

> 注意：
> COS 小程序 SDK 版本需要大于等于 v1.1.1。

API	操作描述
提交音频审核任务	用于提交一个音频审核任务
查询音频审核任务结果	用于查询指定的音频审核任务

提交音频审核任务

功能说明

用于提交一个音频审核任务。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
```

```
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function postAudioAuditing() {
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/audio/auditing';
var body = COS.util.json2xml({
Request: {
Input: {
Object: '1.mp3', /* 需要审核的音频文件，存储桶里的路径 */
},
Conf: {
BizType: '',
DetectType: 'Porn',
}
}
});
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'POST',
Url: url,
Key: '/audio/auditing', /** 固定值，必须 */
ContentType: 'application/xml', /** 固定值，必须 */
Body: body
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的内容。	Container	是
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	存储在 COS 存储桶中的音频文件名称，例如在目录 test 中的文件 audio.mp3，则文件名称为 test/audio.mp3。Object 和 Url 只能选择其中一种。	String	否
Url	Request.Input	音频文件的链接地址，例如 http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/audio.mp3。Object 和 Url 只能选择其中一种。	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告）、Illegal（违法）、Abuse（谩骂），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。	String	否
Callback	Request.Conf	审核结果以回调形式发送至您的回调地址，支持以 `http://` 或者 `https://` 开头的地址，例如：`http://www.callback.com` 。	String	否
CallbackVersion	Request.Conf	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。	string	否
BizType	Request.Conf	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置公共审核策略。	String	否

返回结果说明

详情请参见 提交音频审核任务。

查询音频审核任务

功能说明

用于查询音频审核任务执行情况和结果。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function getAudioAuditingResult() {
var jobId = 'sa0c28d41daff411ecb23352540078cxxx'; // jobId可以通过提交音频审核任务返回
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/audio/auditing/' + jobId;
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'GET',
Key: '/audio/auditing/' + jobId,
Url: url,
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID。	String	是

返回结果说明

详情请参见 查询音频审核任务结果。

文本审核

最近更新时间：2022-12-09 16:37:37

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于文本审核的 API 概览和 SDK 示例代码。

> 注意：
> COS 小程序 SDK 版本需要大于等于 v1.1.1。

API	操作描述
提交文本审核任务	用于提交一个文本审核任务
查询文本审核任务结果	用于查询指定的文本审核任务

提交文本审核任务

功能说明

用于提交一个文本审核任务。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
```

```
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function postTextAuditing() {
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/text/auditing';
var body = COS.util.json2xml({
Request: {
Input: {
Object: 'hello.txt', /* 需要审核的文本文件，存储桶里的路径 */
},
Conf: {
BizType: '',
DetectType: 'Porn',
}
}
});
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'POST',
Url: url,
Key: '/text/auditing', /** 固定值，必须 */
ContentType: 'application/xml', /** 固定值，必须 */
Body: body
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Request	无	文本审核的具体配置项。	Container	是

Container 类型 Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的内容。	Container	是
Conf	Request	审核规则配置。	Container	是

版权所有：腾讯云计算（北京）有限责任公司

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Object	Request.Input	当前 COS 存储桶中的文本文件名称，例如在目录 test 中的文件 img.jpg，则文件名称为 test/img.jpg，文本文件仅支持UTF8编码和 GBK 编码的内容，且文件大小不得超过1MB。	String	否
Content	Request.Input	当传入的内容为纯文本信息，需要先经过 base64 编码，文本编码前的原文长度不能超过10000个 utf8 编码字符。若超出长度限制，接口将会报错。	String	否

注意：

- 不可同时输入 Object 和 Content。
- 当选择 Object 时，审核结果为异步返回，可通过 查询文本审核任务结果 API 接口获取返回结果。
- 当选择 Content 时，审核结果为同步返回，可通过查看响应体 TextAuditingResponse 获取返回结果。
- 目前仅支持中文、英文、阿拉伯数字的检测与审核。

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告）、Illegal（违法）、Abuse（谩骂），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。	String	否
Callback	Request.Conf	审核结果可以回调形式发送至您的回调地址，支持以 `http://` 或者 `https://` 开头的地址，例如：`http://www.callback.com` 。	String	否
BizType	Request.Conf	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置公共审核策略。	String	否

返回结果说明

详情请参见 提交文本审核任务。

版权所有：腾讯云计算（北京）有限责任公司

查询文本审核任务

功能说明

用于查询文本审核任务执行情况和结果。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function getTextAuditingResult() {
var jobId = 'st8d88c664aff511ecb23352540078cxxx'; // jobId可以通过提交文本审核任务返回（Input传入Object）
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/text/auditing/' + jobId;
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'GET',
Key: '/text/auditing/' + jobId,
Url: url,
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID。	String	是

返回结果说明

详情请参见 查询文本审核任务结果。

文档审核

最近更新时间：2022-12-09 16:37:37

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于文档审核的 API 概览和 SDK 示例代码。

> 注意：
> COS 小程序 SDK 版本需要大于等于 v1.1.1。

API	操作描述
提交文档审核任务	用于提交一个文档审核任务
查询文档审核任务结果	用于查询指定的文档审核任务

提交文档审核任务

功能说明

用于提交一个文档审核任务。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
```

```
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function postDocumentAuditing() {
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/document/auditing';
var body = COS.util.json2xml({
Request: {
Input: {
Object: 'test.xlsx', /* 需要审核的文档文件，存储桶里的路径 */
},
Conf: {
BizType: '',
DetectType: 'Porn',
}
}
});
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'POST',
Url: url,
Key: '/document/auditing', /** 固定值，必须 */
ContentType: 'application/xml', /** 固定值，必须 */
Body: body
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的内容。	Container	是
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Url	Request.Input	文档文件的链接地址，例如 http://www.example.com/doctest.doc。	String	是
Type	Request.Input	指定文档文件的类型，如未指定则默认以文件的后缀为类型。 如果文件没有后缀，该字段必须指定，否则会审核失败。例如：doc、docx、ppt、pptx 等。	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。	String	否
Callback	Request.Conf	审核结果可以回调形式发送至您的回调地址，支持以 `http://` 或者 `https://` 开头的地址，例如：`http://www.callback.com`。	String	否
BizType	Request.Conf	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置审核策略。	String	否

返回结果说明

详情请参见 提交文档审核任务。

查询文档审核任务

功能说明

用于查询文档审核任务执行情况和结果。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
```

```
function getDocumentAuditingResult() {
var jobId = 'sd7815c21caff611eca12f525400d88xxx'; // jobId可以通过提交文档审核任务返回
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/document/auditing/' + jobId;
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'GET',
Key: '/document/auditing/' + jobId,
Url: url,
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID。	String	是

返回结果说明

详情请参见 查询文档审核任务结果。

网页审核

最近更新时间：2022-12-09 16:37:37

简介

内容审核功能是由 数据万象（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

> 说明：
> 使用内容审核服务需拥有数据万象使用权限：

- 主账号请 单击此处 进行角色授权。
- 子账号请参见 授权子账号接入数据万象服务 文档。

本文档提供关于网页审核的 API 概览和 SDK 示例代码。

> 注意：
> COS 小程序 SDK 版本需要大于等于 v1.1.1。

API	操作描述
提交网页审核任务	用于提交一个网页审核任务
查询网页审核任务结果	用于查询指定的网页审核任务

提交网页审核任务

功能说明

用于提交一个网页审核任务。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
```

```
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function postWebpageAuditing() {
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/webpage/auditing';
var body = COS.util.json2xml({
Request: {
Input: {
Url: 'https://cloud.tencent.com/', // 存在cos里的资源，审核结果异步返回，可以调用查询文本审核结果api查询
},
Conf: {
BizType: '',
DetectType: 'Porn,Ads',
}
}
});
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'POST',
Url: url,
Key: '/webpage/auditing', /** 固定值，必须 */
ContentType: 'application/xml', /** 固定值，必须 */
Body: body
},
function(err, data){
console.log(err || data);
});
}
```

参数说明

Request 中的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Request	无	网页审核的具体配置项。	Container	是

Container 类型 Request 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Input	Request	需要审核的网页信息。	Container	是

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
Url	Request.Input	网页文件的链接地址，例如 `http://www.test.com` 。	String	是
DataId	Request.Input	该字段在审核结果中会返回原始内容，长度限制为512字节。您可以使用该字段对待审核的数据进行唯一业务标识。	String	否
UserInfo	Request.Input	用户业务字段。	Container	否

Container 节点 UserInfo 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
TokenId	Request.Input.UserInfo	用户业务 TokenId，长度不超过128字节。	String	否
Nickname	Request.Input.UserInfo	用户业务 Nickname，长度不超过128字节。	String	否
DeviceId	Request.Input.UserInfo	用户业务 DeviceId，长度不超过128字节。	String	否
AppId	Request.Input.UserInfo	用户业务 AppId，长度不超过128字节。	String	否
Room	Request.Input.UserInfo	用户业务 Room，长度不超过128字节。	String	否
IP	Request.Input.UserInfo	用户业务 IP，长度不超过128字节。	String	否
Type	Request.Input.UserInfo	用户业务 Type，长度不超过128字节。	String	否

Container 类型 Conf 的具体数据描述如下：

版权所有：腾讯云计算（北京）有限责任公司

节点名称（关键字）	父节点	描述	类型	是否必选
DetectType	Request.Conf	审核的场景类型，有效值：Porn（涉黄）、Ads（广告），可以传入多种类型，不同类型以逗号分隔，例如：Porn,Ads。	String	是
Callback	Request.Conf	回调地址，以 `http://` 或者 `https://` 开头的地址。	String	否
ReturnHighlightHtml	Request.Conf	指定是否需要高亮展示网页内的违规文本，查询及回调结果时会根据此参数决定是否返回高亮展示的 html 内容。取值为 true 或者 false，默认为 false。	Boolean	否

返回结果说明

详情请参见 提交网页审核任务。

查询网页审核任务结果

功能说明

用于查询网页审核任务执行情况和结果。

请求示例

```
var config = {
// 需要替换成您自己的存储桶信息
Bucket: 'examplebucket-1250000000', /* 存储桶，必须 */
Region: 'COS_REGION', /* 存储桶所在地域，必须字段 */
};
function getWebpageAuditingResult() {
var jobId = 'shce868019aff611ecb1155254009a4xxx'; // jobId可以通过提交网页审核任务返回
var host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
var url = 'https://' + host + '/webpage/auditing/' + jobId;
cos.request({
Bucket: config.Bucket,
Region: config.Region,
Method: 'GET',
Key: '/webpage/auditing/' + jobId,
Url: url,
},
function(err, data){
```

```
console.log(err || data);
});
}
```

参数说明

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID。	String	是

返回结果说明

详情请参见 查询网页审核任务结果。

设置访问域名（CDN/全球加速）

最近更新时间：2022-01-06 16:17:22

简介

本文档提供关于如何使用非默认域名请求 COS 服务。

相关参数说明

通过初始化参数，来控制请求域名，相关的参数说明如下，更多初始化参数说明请查看 配置项。

参数名	参数描述	类型	是否必填
Domain	调用操作存储桶和对象的 API 时自定义请求域名。可以使用模版，如 `"{Bucket}.cos.{Region}.myqcloud.com"`，即在调用 API 时会使用参数中传入的 Bucket 和 Region 进行替换。	String	否
Protocol	发请求时用的协议，可选项 `https:`、`http:`，默认判断当前页面是 `http:` 时使用 `http:`，否则使用 `https:`	String	否

默认 CDN 加速域名

关于如何开启默认加速域名请参考 开启默认 CDN 加速域名。

以下代码展示了如何使用默认加速域名访问 COS 服务。

```
var cos = new COS({
Domain: '{Bucket}.file.myqcloud.com', // 自定义加速域名，Domain 参数支持模板，这个例子里 {Bucket} 会在请求时按传入的 Bucket 自动替换
Protocol: 'https:', // 请求协议： 'https:' 或 'http:'
});
```

自定义 CDN 加速域名

关于如何开启 CDN 自定义加速域名请参考 开启自定义 CDN 加速域名。

以下代码展示了如何使用自定义加速域名访问 COS 服务。

```
var cos = new COS({
Domain: 'example-cdn-domain.com', // 自定义加速域名
Protocol: 'https:', // 请求协议： 'https:' 或 'http:'
});
```

版权所有：腾讯云计算（北京）有限责任公司

自定义源站域名

关于如何设置自定义源站域名请参考 自定义源站域名。

以下代码展示了如何使用自定义源站域名访问 COS 服务。

```
var cos = new COS({
Domain: 'example-cos-domain.com', // 自定义源站域名
Protocol: 'https:', // 请求协议： 'https:' 或 'http:'
});
```

全球加速域名

关于全球加速功能请参考 全球加速功能概述。

以下代码展示了如何使用全球加速域名访问 COS 服务。

```
var cos = new COS({
UseAccelerate: true, // 指定 true，使用全球加速域名请求
Protocol: 'https:', // 请求协议： 'https:' 或 'http:'
});
```

版权所有：腾讯云计算（北京）有限责任公司

图片处理
基础图片处理

最近更新时间：2022-10-09 16:17:24

简介

本文档提供关于基础图片处理的相关的 API 概览以及 SDK 示例代码。

服务	功能	说明
基础图片处理服务	缩放	等比缩放、设定目标宽高缩放等多种方式
	裁剪	普通裁剪、缩放裁剪、内切圆、人脸智能裁剪
	旋转	自适应旋转、普通旋转
	格式转换	格式转换、GIF 格式优化、渐进显示
	质量变换	针对 JPG 和 WEBP 图片进行质量变换
	高斯模糊	对图片进行模糊处理
	锐化	对图片进行锐化处理
	添加水印	图片水印、文字水印
	获取图片信息	基本信息、EXIF 信息、主色调
	去除元信息	包括 EXIF 信息
	快速缩略模板	快速实现图片格式转换、缩略、剪裁等功能，生成缩略图

上传时使用图片处理

下面示例展示了如何在上传图片时自动实现图片处理。

图片上传完成后，COS 会存储原始图片和已处理过的图片。后续用户可以通过普通的下载请求获取处理结果。

示例代码

版权所有：腾讯云计算（北京）有限责任公司

```
<view>
<button type="primary" bindtap="button">上传时使用图片处理</button>
</view>
```

```
Page({
button: function () {
wx.chooseMessageFile({
count: 10,
type: 'all',
success: function (res) {
var file = res.tempFiles[0];
wxfs.readFile({
filePath: file.path,
success: function (res) {
cos.putObject(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: file.name,
Body: res.data,
Headers: {
// 通过 imageMogr2 接口使用图片缩放功能：指定图片宽度为 200，宽度等比压缩
'Pic-Operations':
'{"is_pic_info": 1, "rules": [{"fileid": "desample_photo.jpg", "rule": "imageMogr
2/thumbnail/200x/"}]}',
},
},
(err, data) => {
console.log(err || data);
},
);
},
fail: (err) => console.error(err),
});
},
fail: (err) => console.error(err),
});
},
});
```

对云上数据进行图片处理

下面示例展示了如何对已存储在 COS 的图片进行相应处理操作，并将结果存入到 COS。

版权所有：腾讯云计算（北京）有限责任公司

示例代码

```
<view>
<button type="primary" bindtap="button">对云上数据进行图片处理</button>
</view>
```

```
Page({
button: function () {
cos.request(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleImage',
Method: 'POST',
Action: 'image_process',
Headers: {
// 通过 imageMogr2 接口使用图片缩放功能：指定图片宽度为 200，宽度等比压缩
'Pic-Operations':
'{"is_pic_info": 1, "rules": [{"fileid": "desample_photo.jpg", "rule": "imageMogr
2/thumbnail/200x/"}]}',
},
},
(err, data) => {
console.log(err || data);
},
);
},
});
```

下载时使用图片处理

下面示例展示了如何在下载图片时实现图片处理。

示例代码

```
<view>
<button type="primary" bindtap="button">下载时使用图片处理</button>
</view>
```

```
Page({
button: function () {
cos.getObject(
{
```

```
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleImage',
QueryString: `imageMogr2/thumbnail/200x/`,
},
(err, data) => {
console.log(err || data);
},
);
},
});
```

生成带图片处理参数的签名 URL

示例代码

```
<view>
<button type="primary" bindtap="sign">生成带图片处理参数的 URL，带签名</button>
<button type="primary" bindtap="unsign">生成带图片处理参数的 URL，不带签名</button>
</view>
```

```
Page({
sign: function () {
// 生成带图片处理参数的文件签名 URL，过期时间设置为 30 分钟。
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleobject',
Query: {
`imageMogr2/thumbnail/200x/`: ''
},
Expires: 1800,
Sign: true,
},
(err, data) => {
if (data) {
console.log(data.URL);
}
},
);
},
unsign: function () {
// 生成带图片处理参数的文件 URL，不带签名。
```

```
cos.getObjectUrl(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleobject',
Query: {
`imageMogr2/thumbnail/200x/`: ''
},
Sign: false,
},
(err, data) => {
if (data) {
console.log(data.URL);
}
},
);
},
});
```

盲水印

最近更新时间：2022-10-09 16:17:25

简介

本文档提供关于盲水印的相关的 API 概览以及 SDK 示例代码。

关于盲水印 API 文档请参见盲水印。

添加盲水印

功能说明

盲水印支持在上传时添加和下载时添加。

示例代码一：上传时添加盲水印

```
<view>
<button type="primary" bindtap="button">上传时添加盲水印</button>
</view>
```

```
Page({
button: function () {
wx.chooseMessageFile({
count: 10,
type: 'all',
success: function (res) {
var file = res.tempFiles[0];
wxfs.readFile({
filePath: file.path,
success: function (res) {
cos.putObject(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: file.name,
Body: res.data,
Headers: {
// 万象持久化接口，上传时持久化
'Pic-Operations': '{"is_pic_info": 1, "rules": [{"fileid": "desample_photo.jpg",
"rule": "watermark/3/type/1/image/aHR0cDovL2V4YW1wbGVzLTEyNTEwMDAwMDQucGljc2gubXl
xY2xvdWQuY29tL3NodWl5aW4uanBn"}]}',
```

```
},
},
(err, data) => {
console.log(err || data);
},
);
},
fail: (err) => console.error(err),
});
},
fail: (err) => console.error(err),
});
},
});
```

示例代码二：下载时添加盲水印

```
<view>
<button type="primary" bindtap="button">下载时添加盲水印</button>
</view>
```

```
Page({
button: function () {
cos.getObject(
{
Bucket: 'examplebucket-1250000000',
Region: 'COS_REGION',
Key: 'exampleImage',
QueryString: `watermark/3/type/1/image/aHR0cDovL2V4YW1wbGVzLTEyNTEwMDAwMDQucGljc2
gubXlxY2xvdWQuY29tL3NodWl5aW4uanBn`,
},
(err, data) => {
console.log(err || data);
},
);
},
});
```

异常处理

最近更新时间：2022-01-05 14:31:57

简介

调用 SDK 接口请求 COS 服务失败时，将在回调函数里返回错误信息。

错误处理示例

```
cos.headBucket({
Bucket: 'examplebucket-1250000000',
Region: 'ap-beijing',
}, function(err, data) {
if (err) {
console.log(err.error);
}
});
```

客户端异常

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误，如果请求成功则为空，更多详情请参见 错误码 文档	Object
- error	请求报错信息	Object/String

服务端异常

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功则为空，更多详情请参见 错误码 文档	Object

版权所有：腾讯云计算（北京）有限责任公司

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，如200、403、404等	Number
- headers	请求返回的头部信息	Object
- error	请求报错信息	Object/String
- - Code	请求失败时 body 返回的 Error Code，更多详情请参见 错误码 文档	String
- - Message	请求失败时 body 返回的 Error Message，更多详情请参见 错误码 文档	String
- - RequestId	服务器请求日志里的唯一 ID，可用于遇到问题提工单排查错误原因	String

使用自助诊断工具

针对请求可能遇到不同的报错情况，我们为您提供了 COS 自助诊断工具，帮助您快速定位问题，调试报错代码。

使用步骤

1. 复制异常处理返回的 RequestId（请求 ID）。
2. 单击 COS 自助诊断工具，进入自助诊断页面。
   COS 自助诊断工具
   点击自助诊断
   输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。
3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击**开始诊断**。
4. 稍侯片刻，便能看到相应的智能诊断结果。

常见问题

若您在使用小程序 SDK 过程中，有相关的疑问，请参见 常见问题 文档小程序部分。

版权所有：腾讯云计算（北京）有限责任公司

错误码

最近更新时间：2019-08-02 16:37:17

概述

在使用 SDK 提供的 COS API 服务过程中，出现失败可以分为客户端错误和服务端返回的错误。

- 服务端返回的错误，指服务端处理一些不符合要求的客户端请求所返回的错误。例如访问不存在的文件，没有访问文件的权限等。更多服务端返回的错误码详细信息，请参阅 API 错误码。
- 客户端错误，主要是指网络异常、文件读写 IO 异常、参数校验失败等。

此文将为您介绍客户端定义的错误码信息。

错误码列表

目前支持的 SDK 有：Android、iOS。

错误码	错误信息	错误描述
10000	InvalidArgument	参数校验失败，如必填参数为空。
10001	InvalidCredentials	密钥信息校验失败，如密钥为空。
10002	BadRequest	SDK配置错误，如 APPID，region 配置出错。
10003	SinkSourceNotFound	输入源或者输出源错误，如上传的文件不存在。
10004	UnsupportOperation	无法支持的操作。
20000	InternalError	内部错误，如 xml 格式数据解析失败。
20001	ServerError	服务错误，如返回了非 xml 格式数据。
20002	IOError	流读写 IO 异常，如文件读写 IO 异常。
20003	NetworkError	网络出现异常，如网络不可用，DNS 解析失败等。
20004	DataIntegrityError	数据完整性校验失败。
30000	UserCancelled	用户已取消了请求。
30001	AlreadyFinished	已执行过请求。

版权所有：腾讯云计算（北京）有限责任公司