内容识别

SDK 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】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

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。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。
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默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

SDK 文档
SDK 概览
COS Android SDK
快速入门
快速体验
存储桶操作
内容识别
服务和队列配置
开通AI内容识别服务
查询AI内容识别服务
关闭AI内容识别服务
查询AI内容识别队列
更新AI内容识别队列
图片上色
图像超分
图像增强
图像智能裁剪
图像修复
图片二维码上传时识别
图片二维码识别
图片二维码生成
图片标签
图片质量评分
图片文字识别(OCR)
实时文字翻译
人脸识别
人脸检测
人脸特效
人体识别
宠物识别
车辆车牌识别
卡证识别
游戏场景识别
商品抠图
人脸核身
身份证识别
活体人脸核身
获取数字验证码

版权所有：腾讯云计算（北京）有限责任公司

获取动作顺序
内容翻译
词性分析
视频目标检测
创建视频目标检测模板
更新视频目标检测模板
提交视频目标检测任务
视频人像分割
提交视频人像抠图任务
C SDK
快速入门
内容识别
二维码识别
C++ SDK
快速入门
.NET(C#) SDK
快速入门
内容识别
图片二维码
Go SDK
快速入门
开通服务
AI 内容识别
图片上色
图像增强
图像超分
图像智能裁剪
图像修复
图片二维码
图片标签
图片质量评估
图片文字识别（OCR）
实时文字翻译
人脸识别
人体识别
宠物识别
汽车识别
卡证识别
游戏场景识别
人脸核身
商品抠图

通用抠图
人像抠图
视频目标检测
创建任务
创建模板
更新模板
COS iOS SDK
快速入门
快速体验
存储桶操作
内容识别
图片标签
图片二维码
语音识别
任务接口
语音 bucket 接口
队列接口
Java SDK
快速入门
常见问题
AI 内容识别
AI通用接口示例
图片二维码
图片标签
汽车识别
回调信息处理
异常处理
JavaScript SDK
快速入门
内容识别
服务和队列配置
开通AI内容识别服务
查询 AI 内容识别服务
关闭AI内容识别服务
查询AI内容识别队列
更新AI内容识别队列
图片上色
图像超分
图像增强
图像智能裁剪
图像修复

版权所有：腾讯云计算（北京）有限责任公司

图片二维码
图片标签
图片质量评分
图片文字识别（OCR）
实时文字翻译
人脸识别
人脸检测
人脸特效
人体识别
宠物识别
汽车识别
卡证识别
游戏场景识别
人脸核身
身份证识别
活体人脸核身
获取数字验证码
获取动作顺序
商品抠图
内容翻译
提交任务
词性分析
提交任务
视频目标检测
提交任务
创建模板
更新模板
视频人像分割
提交任务
Node.js SDK
快速入门
内容识别
服务和队列配置
开通 AI 内容识别服务
查询 AI 内容识别服务
关闭 AI 内容识别服务
查询 AI 内容识别队列
更新 AI 内容识别队列
图片上色
图像超分
图像增强

图像智能裁剪
图像修复
图片二维码
图片标签
图片质量评分
图片文字识别（OCR）
实时文字翻译
人脸识别
人脸检测
人脸特效
人体识别
宠物识别
汽车识别
卡证识别
游戏场景识别
人脸核身
身份证识别
活体人脸核身
获取数字验证码
获取动作顺序
商品抠图
内容翻译
提交任务
视频目标检测
提交任务
更新模板
创建模板
视频人像分割
提交任务
PHP SDK
快速入门
内容识别
人脸特效
图片标签
图片二维码
图片文字识别（OCR）
图片质量评分
异步服务和队列配置
Python SDK
快速入门
内容识别

语音识别
任务接口
语音 bucket 接口
队列接口
服务和队列配置
开通AI内容识别服务
查询AI内容识别服务
关闭AI内容识别服务
查询AI内容识别队列
更新AI内容识别队列
图片二维码
图片主体检测
图片上色
图片标签
图像增强
图像智能裁剪
图像修复
智能抠图
通用抠图
商品抠图
人像抠图
人脸核身
身份证识别
获取动作顺序
获取数字验证码
活体人脸核身
人体识别
人脸识别
人脸检测
人脸特效
宠物识别
游戏场景识别
卡证识别
实时文字翻译
小程序 SDK
快速入门
内容识别
服务和队列配置
开通 AI 内容识别服务
查询 AI 内容识别服务
关闭 AI 内容识别服务

版权所有：腾讯云计算（北京）有限责任公司

查询 AI 内容识别队列
更新 AI 内容识别队列
图片上色
图像超分
图像增强
图像智能裁剪
图像修复
图片二维码
图片标签
图片质量评分
图片文字识别（OCR）
实时文字翻译
人脸识别
人脸检测
人脸特效
人体识别
宠物识别
汽车识别
卡证识别
游戏场景识别
人脸核身
身份证识别
活体人脸核身
获取数字验证码
获取动作顺序
商品抠图
内容翻译
提交任务
词性分析
提交任务
视频目标检测
提交任务
创建模板
更新模板
视频人像分割
提交任务

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：2024-10-16 09:22:52

内容识别服务由腾讯云 数据万象（Cloud Infinite，CI） 提供。为了开发者更方便、更快速地使用数据万象的功能，以及 CDN 的云闪图片分发功能，我们提供了以下 SDK，开发者可根据具体需求进行选择，详情请参见对应的快速入门文档。对象存储（Cloud Object Storage，COS）的 SDK 也集成了数据万象的数据处理功能，若您需要使用其他语言的 SDK，例如 C++ 、JavaScript 等，请参见 COS SDK 概览 。

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

说明

- CDN 云闪图片分发介绍请参见 云闪图片分发 。
- 关于 TPG 的更多介绍，请参见 TPG 功能介绍 。

COS Android SDK 快速入门

最近更新时间：2024-12-27 16:37:03

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。
> 如果您仍在使用 JSON 版本 SDK，请 升级到 XML Android SDK 。

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.+'
}
```

> **说明：**
> 如果使用的是精简版 SDK，请将下面实例代码以及其他 Android 文档中的 CosXmlService 改为 CosXmlSimpleService。

关闭 beacon 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
- qcloud-cos-android-base：COS 基础协议实现
- qcloud-track：日志封装相关
- 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。

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。**由于该方式存在泄露密钥的风险，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 cosXmlSelfService = 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：
高级上传时二进制数据和输入流仅支持普通上传，不支持分块上传。

```
// 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
TransferConfig transferConfig = new TransferConfig.Builder()
        // 设置启用分块上传的最小对象大小 默认为2M
        .setDivisionForUpload(2097152)
        // 设置分块上传时的分块大小 默认为1M
        .setSliceSizeForUpload(1048576)
        // 设置是否强制使用简单上传，禁止分块上传
        .setForceSimpleUpload(false)
        .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://console.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject.txt"; //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，用于获取uploadId (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，即：
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为
CosXmlServiceException?
    @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
    }
});
```

授权说明

在您进行 授权策略 时，action 请参考如下示例。

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
                //head操作
                "name/cos:HeadObject",
                //简单上传操作
                "name/cos:PutObject",
                //分块上传：初始化分块操作
                "name/cos:InitiateMultipartUpload",
                //分块上传：List 进行中的分块上传
                "name/cos:ListMultipartUploads",
                //分块上传：List 已上传分块操作
                "name/cos:ListParts",
                //分块上传：上传分块操作
                "name/cos:UploadPart",
                //分块上传：完成所有分块上传操作
                "name/cos:CompleteMultipartUpload",
                //取消分块上传操作
                "name/cos:AbortMultipartUpload"
      ],
      "effect": "allow",
      "resource": [
```

```
            "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
        ]
      }
    ]
}
```

对象存储的更多 action，请参见 支持CAM的业务接口 。

> **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。 https://console.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject.txt"; //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.txt";

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，即：
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为
CosXmlServiceException?
    @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
    }
});
```

授权说明

在您进行 授权策略 时，action 请参考如下示例。

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
                //head操作
                "name/cos:HeadObject",
                //下载操作
                "name/cos:GetObject",
```

```
        ],
        "effect": "allow",
        "resource": [
          "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
        ]
      }
    ]
}
```

对象存储的更多 action，请参见 支持CAM的业务接口 。

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。

快速体验

最近更新时间：2024-12-27 16:37:03

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，如下图所示。



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 ，如下图所示。



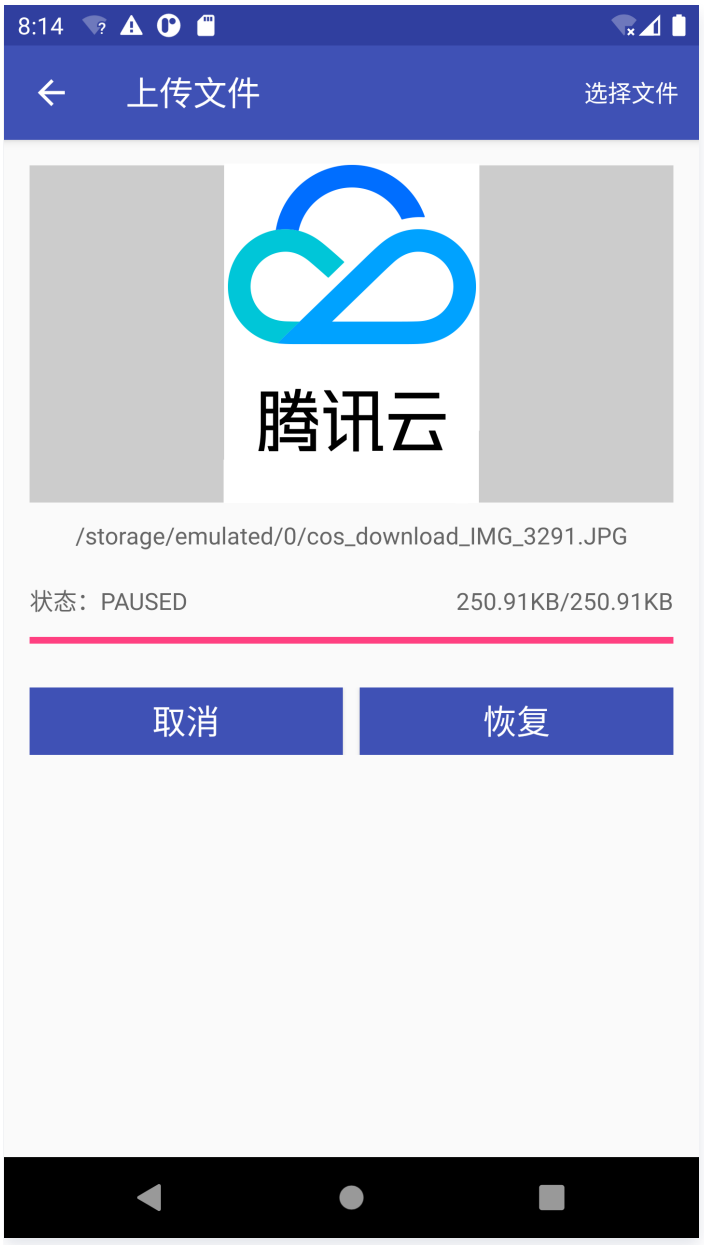
查询对象列表

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，如下图所示。



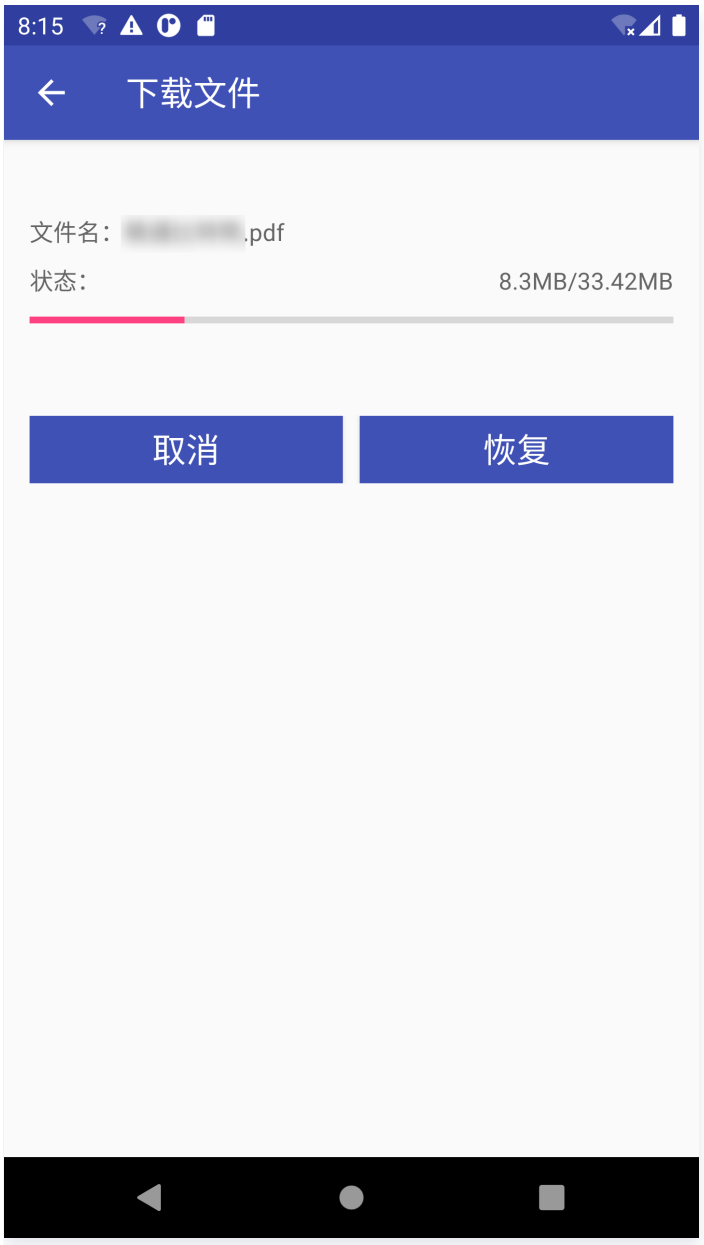
上传文件

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，如下图所示。



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，如下图所示。


8:15
下载文件
文件名：[redacted].pdf
状态：
8.3MB/33.42MB
取消
恢复

存储桶操作

最近更新时间：2024-12-27 16:37:03

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，请参见 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，即:
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为
CosXmlServiceException?
    @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://console.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
PutBucketRequest putBucketRequest = new PutBucketRequest(bucket);
// 指定存储桶是否为 多AZ 配置
putBucketRequest.enableMAZ(false);
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，即：
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为 CosXmlServiceException?
    @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://console.cloud.tencent.com/cos5/bucket
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，即：
    // clientException 的类型为 CosXmlClientException?，serviceException 的类型为 CosXmlServiceException?
    @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://console.cloud.tencent.com/cos5/bucket
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。

内容识别
服务和队列配置
开通AI内容识别服务

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于开通AI内容识别服务的相关 API 概览以及 SDK 示例代码。

API	操作描述
开通AI内容识别服务	本接口用于开通AI 内容识别（异步）服务并生成队列

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

开通AI内容识别服务

功能说明

本接口用于开通AI 内容识别（异步）服务并生成队列。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
OpenBucketAiRequest request = new OpenBucketAiRequest(bucket);

ciService.openBucketAiAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 开通AI内容识别服务的结果
        // 详细字段请查看api文档或者SDK源码
        OpenBucketAiResult result = (OpenBucketAiResult) cosResult;

    }
    @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。

查询AI内容识别服务

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于查询AI内容识别服务的相关 API 概览以及 SDK 示例代码。

API	操作描述
查询AI内容识别服务	本接口用于查询已经开通AI 内容识别（异步）服务的存储桶

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AI内容识别服务

功能说明

本接口用于查询已经开通AI 内容识别（异步）服务的存储桶。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
GetAIBucketRequest request = new GetAIBucketRequest();
request.regions = "ap-shanghai,ap-beijing";// 设置地域信息，例如 ap-shanghai、ap-beijing，若查询多个地域以“,”分隔字符串，详情请参见 地域与域名
request.bucketNames = "examplebucket-1250000000";// 设置存储桶名称，以“,”分隔，支持多个存储桶，精确搜索
request.bucketName = "examplebucket-1250000000";// 设置存储桶名称前缀，前缀搜索
request.pageNumber = 1;// 设置第几页
request.pageSize = 10;// 设置每页个数，大于0且小于等于100的整数

ciService.getAIBucketAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 查询AI内容识别服务的结果
        // 详细字段请查看api文档或者SDK源码
        GetAIBucketResult result = (GetAIBucketResult) cosResult;

    }
    @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。

关闭AI内容识别服务

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于关闭AI内容识别服务的相关 API 概览以及 SDK 示例代码。

API	操作描述
关闭AI内容识别服务	本接口用于关闭AI 内容识别（异步）服务并删除队列

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

关闭AI内容识别服务

功能说明

本接口用于关闭AI 内容识别（异步）服务并删除队列

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
CloseAIBucketRequest request = new CloseAIBucketRequest(bucket);

ciService.closeAIBucketAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 关闭AI内容识别服务的结果
        // 详细字段请查看api文档或者SDK源码
        CloseAIBucketResult result = (CloseAIBucketResult) cosResult;

    }
    @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。

查询AI内容识别队列

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于查询AI内容识别队列的相关 API 概览以及 SDK 示例代码。

API	操作描述
查询AI内容识别队列	本接口用于查询AI 内容识别（异步）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AI内容识别队列

功能说明

本接口用于查询AI 内容识别（异步）队列

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
GetAIQueueRequest request = new GetAIQueueRequest(bucket);
request.queueIds = "p4a17eeea29334bf499b7e20e2fbfd99d";// 设置队列 ID，以“,”符号分割字符串
request.state = "Active";// 设置Active 表示队列内的作业会被调度执行Paused 表示队列暂停，作业不再会被调度执行，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
request.pageNumber = 1;// 设置第几页，默认值1
request.pageSize = 10;// 设置每页个数，默认值10

ciService.getAIQueueAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 查询AI内容识别队列的结果
        // 详细字段请查看api文档或者SDK源码
        GetAIQueueResult result = (GetAIQueueResult) cosResult;

    }
    @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。

更新AI内容识别队列

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于更新AI内容识别队列的相关 API 概览以及 SDK 示例代码。

API	操作描述
更新AI内容识别队列	本接口用于更新AI 内容识别（异步）的队列

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

更新AI内容识别队列

功能说明

本接口用于更新AI 内容识别（异步）的队列。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
UpdateAIQueueRequest request = new UpdateAIQueueRequest(bucket, "queueId");
UpdateAIQueue updateAIQueue = new UpdateAIQueue();// 更新AI内容识别队列请求体
request.setUpdateAIQueue(updateAIQueue);// 设置请求
// 设置队列名称，仅支持中文、英文、数字、_、-和*，长度不超过 128;是否必传：是
updateAIQueue.name = "My-Queue-ai";
// 设置Active 表示队列内的作业会被调度执行Paused 表示队列暂停，作业不再会被调度执行，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;是否必传：是
updateAIQueue.state = "Active";
NotifyConfig notifyConfig = new NotifyConfig();
updateAIQueue.notifyConfig = notifyConfig;
// 设置回调开关OffOn;是否必传：否
notifyConfig.state = "On";
// 设置回调事件TaskFinish：任务完成WorkflowFinish：工作流完成;是否必传：否
notifyConfig.event = "TaskFinish";
// 设置回调格式XMLJSON;是否必传：否
notifyConfig.resultFormat = "JSON";
// 设置回调类型UrlTDMQ;是否必传：否
```

```
notifyConfig.type = "Url";
// 设置回调地址，不能为内网地址。;是否必传：否
notifyConfig.url = "http://callback.demo.com";

ciService.updateAIQueueAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 更新AI内容识别队列的结果
        // 详细字段请查看api文档或者SDK源码
        UpdateAIQueueResult result = (UpdateAIQueueResult) cosResult;

    }
    @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。

图片上色

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于图片上色的相关 API 概览以及 SDK 示例代码。

API	操作描述
图片上色	腾讯云数据万象通过 AIImageColoring 接口对黑白图像进行上色

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

图片上色

功能说明

腾讯云数据万象通过 AIImageColoring 接口对黑白图像进行上色

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AIImageColoringRequest request = new AIImageColoringRequest(bucket);
// 存储桶中的位置标识符，即对象键
request.objectKey = "folder/document.jpg";
// 设置待上色图片url，与objectKey二选其一，如果同时存在，则默认以objectKey为准
request.detectUrl = "http://www.example.com/abc.jpg";
// 设置结果文件保存路径
request.saveLocalPath = "保存到本地的路径";

cosXmlService.aiImageColoringAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 图片上色的结果
        // 详细字段请查看api文档或者SDK源码
        GetObjectResult result = (GetObjectResult) cosResult;

    }
    @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。

图像超分

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于图像超分的相关 API 概览以及 SDK 示例代码。

API	操作描述
图像超分	腾讯云数据万象通过 AISuperResolution 接口对图像进行超分辨率处理，当前默认超分为宽高的2倍。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

图像超分

功能说明

腾讯云数据万象通过 AISuperResolution 接口对图像进行超分辨率处理，当前默认超分为宽高的2倍。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AISuperResolutionRequest request = new AISuperResolutionRequest(bucket);
// 存储桶中的位置标识符，即对象键
request.objectKey = "folder/document.jpg";
// 设置您可以通过填写 detectUrl 处理任意公网可访问的图片链接。不填写 detectUrl 时，后台会
默认处理 objectKey ，填写了 detectUrl 时，后台会处理 detectUrl 链接，无需再填写
objectKey，detectUrl 示例：http://www.example.com/abc.jpg。
request.detectUrl = "http://www.example.com/abc.jpg";
request.magnify = 2;// 设置目标放大倍数，默认为2，范围[2,4]，仅支持偶数
// 设置结果文件保存路径
request.saveLocalPath = "保存到本地的路径";

cosXmlService.aiSuperResolutionAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 图像超分的结果
        // 详细字段请查看api文档或者SDK源码
```

```
        GetObjectResult result = (GetObjectResult) cosResult;

    }
    @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。

图像增强

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于图像增强的相关 API 概览以及 SDK 示例代码。

API	操作描述
图像增强	腾讯云数据万象通过 AIEnhanceImage 接口对图像进行增强处理

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

图像增强

功能说明

腾讯云数据万象通过 AIEnhanceImage 接口对图像进行增强处理

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AIEnhanceImageRequest request = new AIEnhanceImageRequest(bucket);
request.objectKey = "folder/document.jpg";
request.detectUrl = "http://www.example.com/abc.jpg";// 设置您可以通过填写
detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理
ObjectKey ，填写了detect-url 时，后台会处理 detect-url链接，无需再填写 ObjectKey ，
detect-url 示例：http://www.example.com/abc.jpg
request.denoise = 3;// 设置去噪强度值，取值范围为 0 - 5 之间的整数，值为 0 时不进行去噪
操作，默认值为3。
request.sharpen = 3;// 设置锐化强度值，取值范围为 0 - 5 之间的整数，值为 0 时不进行锐化
操作，默认值为3。
request.ignoreError = 1;// 设置
// 设置结果文件保存路径
request.saveLocalPath = "保存到本地的路径";

cosXmlService.aiEnhanceImageAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
```

```
            // result 图像增强的结果
            // 详细字段请查看api文档或者SDK源码
            GetObjectResult result = (GetObjectResult) cosResult;

        }
        @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。

图像智能裁剪

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于图像智能裁剪的相关 API 概览以及 SDK 示例代码。

API	操作描述
图像智能裁剪	腾讯云数据万象通过 AIImageCrop 接口对图像进行智能裁剪，支持持久化、云上处理及下载时处理

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

图像智能裁剪

功能说明

腾讯云数据万象通过 AIImageCrop 接口对图像进行智能裁剪，支持持久化、云上处理及下载时处理。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AIImageCropRequest request = new AIImageCropRequest(bucket);
request.objectKey = "folder/document.jpg";
request.detectUrl = "http://www.example.com/abc.jpg";// 设置您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey detect-url 示例：http://www.example.com/abc.jpg
request.width = 10;// 设置需要裁剪区域的宽度，与height共同组成所需裁剪的图片宽高比例；输入数字请大于0、小于图片宽度的像素值
request.height = 6;// 设置需要裁剪区域的高度，与width共同组成所需裁剪的图片宽高比例；输入数字请大于0、小于图片高度的像素值；width : height建议取值在[1, 2.5]之间，超过这个范围可能会影响效果
request.fixed = 0;// 设置是否严格按照 width 和 height 的值进行输出。取值为0时，宽高比例（width : height）会简化为最简分数，即如果width输入10、height输入20，会简化为1: 2；取值为1时，输出图片的宽度等于width，高度等于height；默认值为0
request.ignoreError = 1;// 设置当此参数为1时，针对文件过大等导致处理失败的场景，会直接返回原图而不报错
```

```
// 设置结果文件保存路径
request.saveLocalPath = "保存到本地的路径";

cosXmlService.aiImageCropAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 图像智能裁剪的结果
        // 详细字段请查看api文档或者SDK源码
        GetObjectResult result = (GetObjectResult) cosResult;

    }
    @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。

图像修复

最近更新时间：2025-06-06 18:13:23

简介

本文档提供关于图像修复的相关 API 概览以及 SDK 示例代码。

API	操作描述
图像修复	腾讯云数据万象通过 ImageRepair 接口能检测并擦除图片中常见的标志,并对擦除部分进行智能修复，此功能需携带签名。

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

腾讯云数据万象通过 ImageRepair 接口能检测并擦除图片中常见的标志,并对擦除部分进行智能修复，此功能需携带签名。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
ImageRepairRequest request = new ImageRepairRequest(bucket,
"folder/document.jpg");
// 设置遮罩 例如： [[[608, 794], [1024, 794], [1024, 842], [608, 842]],[[1295,
62], [1295, 30], [1597, 32],[1597,64]]] ，顺时针输□多边形的每个点的坐标,每个多边形:
[[x1, y1], [x2, y2]...] ，形式为三维矩阵（多个多边形： [多边形1,多边形2] ）或□维矩阵
（单个多边形），且需要经过 URL 安全的 Base64 编码
request.maskPoly =
"W1tbNjA4LCA3OTRdLCBbMTAyNCwgNzk0XSwgWzEwMjQsIDg0Ml0sIFs2MDgsIDg0Ml1dLFtbMTI5N
SwgNjdLCBbMTI5NSwgMzBdLCBbMTU5NywgMzJdLFsxNTk3LDY0XV1d";
// 设置结果文件保存路径
request.saveLocalPath = "保存到本地的路径";

cosXmlService.imageRepairAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 图像修复的结果
        // 详细字段请查看 api 文档或者 SDK 源码
```

```
        GetObjectResult result = (GetObjectResult) cosResult;

    }
    @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。

图片二维码上传时识别

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于图片二维码上传时识别相关 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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

识别二维码

功能说明

支持在上传时添加二维码识别。

示例代码：上传时识别二维码

```
// 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.cloud.tencent.com/cos5/bucket
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject.pdf"; //文档位于存储桶中的位置标识符，即对象键
String localPath = "localdownloadpath"; // 二维码图片本地路径
final PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, localPath);

List<PicOperationRule> rules = new LinkedList<>();
rules.add(new PicOperationRule("/test.png", "QRcode/cover/0"));
PicOperations picOperations = new PicOperations(false, rules);
putObjectRequest.setPicOperations(picOperations);

cosXmlService.putObjectAsync(putObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutObjectResult putObjectResult = (PutObjectResult) result;
        PicUploadResult picUploadResult = putObjectResult.picUploadResult; //图片处理结局
    }

    @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。

图片二维码识别

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于图片二维码识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
图片二维码识别	数据万象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。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

图片二维码识别

功能说明

数据万象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。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
RecognitionQRcodeRequest request = new RecognitionQRcodeRequest(bucket,
"folder/sample.jpg");
request.cover = 0;// 设置二维码覆盖功能，将对识别出的二维码覆盖上马赛克。取值为0或1。0表
示不开启二维码覆盖，1表示开启二维码覆盖，默认值0

cosXmlService.recognitionQRcodeAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 图片二维码识别的结果
        // 详细字段请查看api文档或者SDK源码
        RecognitionQRcodeResult result = (RecognitionQRcodeResult) cosResult;

    }
    @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。

图片二维码生成

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于图片二维码生成的相关 API 概览以及 SDK 示例代码。

API	操作描述
图片二维码生成	数据万象二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

图片二维码生成

功能说明

数据万象二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
CreateCRcodeRequest request = new CreateCRcodeRequest(bucket);
request.qrcodeContent = "可识别的二维码文本信息";// 设置可识别的二维码文本信息
request.mode = 0;// 设置生成的二维码类型，可选值：0或1。0为二维码，1为条形码，默认值为0
request.width = "10";// 设置指定生成的二维码或条形码的宽度，高度会进行等比压缩

cosXmlService.createCRcodeAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 图片二维码生成的结果
        // 详细字段请查看api文档或者SDK源码
        CreateCRcodeResult result = (CreateCRcodeResult) cosResult;

    }
    @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。

图片标签

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于图片标签的相关 API 概览以及 SDK 示例代码。

API	操作描述
图片标签	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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

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。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
DetectLabelRequest request = new DetectLabelRequest(bucket);
// 存储桶中的位置标识符，即对象键
request.objectKey = "folder/document.jpg";
// 设置您可以通过填写 detectUrl 处理任意公网可访问的图片链接。不填写 detectUrl 时，后台会默认处理 objectKey ，填写了 detectUrl 时，后台会处理 detectUrl 链接，无需再填写 objectKey，detectUrl 示例：http://www.example.com/abc.jpg。
request.detectUrl = "http://www.example.com/abc.jpg";
request.scenes = "web";// 设置本次调用支持的识别场景，可选值如下：web，针对网络图片优化；camera，针对手机摄像头拍摄图片优化；album，针对手机相册、网盘产品优化；news，针对新闻、资讯、广电等行业优化；如果不传此参数，则默认为camera。支持多场景（scenes）一起检测，以，分隔。例如，使用 scenes=web，camera 即对一张图片使用两个模型同时检测，输出两套识别结果。

cosXmlService.detectLabelAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
```

```
        // result 图片标签的结果
        // 详细字段请查看api文档或者SDK源码
        DetectLabelResult result = (DetectLabelResult) cosResult;

    }
    @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。

图片质量评分

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于图片质量评分的相关 API 概览以及 SDK 示例代码。

API	操作描述
图片质量评分	图片质量评分功能为同步请求方式，您可以通过本接口对图片文件进行检测，从多方面评估，最终给出综合可观的清晰度评分和主观的美观度评分。该接口属于 GET 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 。

图片质量评分

功能说明

图片质量评分功能为同步请求方式，您可以通过本接口对图片文件进行检测，从多方面评估，最终给出综合可观的清晰度评分和主观的美观度评分。该接口属于 GET 请求

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AssessQualityRequest request = new AssessQualityRequest(bucket,
"folder/document.jpg");

cosXmlService.assessQualityAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 图片质量评分的结果
        // 详细字段请查看api文档或者SDK源码
        AssessQualityResult result = (AssessQualityResult) cosResult;

    }
    @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。

图片文字识别(OCR)

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于图片文字识别(OCR)的相关 API 概览以及 SDK 示例代码。

API	操作描述
图片文字识别(OCR)	通用文字识别功能（Optical Character Recognition，OCR）基于行业前沿的深度学习技术，将图片上的文字内容，智能识别为可编辑的文本，可应用于随手拍扫描、纸质文档电子化、电商广告审核等多种场景，大幅提升信息处理效率。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

图片文字识别(OCR)

功能说明

通用文字识别功能（Optical Character Recognition，OCR）基于行业前沿的深度学习技术，将图片上的文字内容，智能识别为可编辑的文本，可应用于随手拍扫描、纸质文档电子化、电商广告审核等多种场景，大幅提升信息处理效率。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
COSOCRRequest request = new COSOCRRequest(bucket, "folder/document.jpg");
request.detectUrl = "http://www.example.com/abc.jpg";// 设置您可以通过填写
detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey
，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey detect-url
示例：http://www.example.com/abc.jpg
request.type = "general";// 设置ocr的识别类型，有效值为general，accurate，
efficient，fast，handwriting。general表示通用印刷体识别；accurate表示印刷体高精度版；
efficient表示印刷体精简版；fast表示印刷体高速版；handwriting表示手写体识别。默认值为
general。
request.languageType = "auto";// 设置type值为general时有效，表示识别语言类型。支持自
动识别语言类型，同时支持自选语言种类，默认中英文混合(zh)，各种语言均支持与英文混合的文字识
别。可选值：zh：中英混合zh_rare：支持英文、数字、中文生僻字、繁体字，特殊符号等auto：自动
mix：混合语种jap：日语kor：韩语spa：西班牙语fre：法语ger：德语por：葡萄牙语vie：越语
```

```
may：马来语rus：俄语ita：意大利语hol：荷兰语swe：瑞典语fin：芬兰语dan：丹麦语nor：挪威语
hun：匈牙利语tha：泰语hi：印地语ara：阿拉伯语
request.ispdf = false;// 设置type值为general，fast时有效，表示是否开启PDF识别，有效值
为true和false，默认值为false，开启后可同时支持图片和PDF的识别。
request.pdfPagenumber = 1;// 设置type值为general，fast时有效，表示需要识别的PDF页面
的对应页码，仅支持PDF单页识别，当上传文件为PDF且ispdf参数值为true时有效，默认值为1。
request.isword = false;// 设置type值为general，accurate时有效，表示识别后是否需要返回
单字信息，有效值为true和false，默认为false
request.enableWordPolygon = false;// 设置type值为handwriting时有效，表示是否开启单
字的四点定位坐标输出，有效值为true和false，默认值为false。

cosXmlService.cOSOCRAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 图片文字识别(OCR)的结果
        // 详细字段请查看api文档或者SDK源码
        COSOCRResult result = (COSOCRResult) cosResult;

    }
    @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。

实时文字翻译

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于实时文字翻译的相关 API 概览以及 SDK 示例代码。

API	操作描述
实时文字翻译	腾讯云数据万象通过 AutoTranslationBlock 接口对文字块进行翻译，请求时需要携带签名

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

实时文字翻译

功能说明

腾讯云数据万象通过 AutoTranslationBlock 接口对文字块进行翻译，请求时需要携带签名。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AutoTranslationBlockRequest request = new AutoTranslationBlockRequest(bucket);
request.inputText = "待翻译的文本";// 设置待翻译的文本
request.sourceLang = "zh";// 设置输入语言，如 "zh"
request.targetLang = "en";// 设置输出语言，如 "en"
request.textDomain = "general";// 设置文本所属业务领域，如: "ecommerce", //缺省值为
general
request.textStyle = "sentence";// 设置文本类型，如: "title", //缺省值为 sentence

cosXmlService.autoTranslationBlockAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 实时文字翻译的结果
        // 详细字段请查看api文档或者SDK源码
        AutoTranslationBlockResult result = (AutoTranslationBlockResult)
cosResult;

    }
```

```
    @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。

人脸识别
人脸检测

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于人脸检测的相关 API 概览以及 SDK 示例代码。

API	操作描述
人脸检测	人脸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人脸位置。该接口属于 GET 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 。

人脸检测

功能说明

人脸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人脸位置。该接口属于 GET 请求。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AIDetectFaceRequest request = new AIDetectFaceRequest(bucket,
"folder/document.jpg");
request.maxFaceNum = 1;// 设置最多处理的人脸数目。默认值为1（仅检测图片中面积最大的那张
人脸），最大值为120。此参数用于控制处理待检测图片中的人脸个数，值越小，处理速度越快。

cosXmlService.aiDetectFaceAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 人脸检测的结果
        // 详细字段请查看api文档或者SDK源码
        AIDetectFaceResult result = (AIDetectFaceResult) cosResult;

    }
    @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。

人脸特效

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于人脸特效的相关 API 概览以及 SDK 示例代码。

API	操作描述
人脸特效	人脸特效，提供人脸美颜、人像变换、人像分割功能

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

人脸特效

功能说明

人脸特效，提供人脸美颜、人像变换、人像分割功能。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AIFaceEffectRequest request = new AIFaceEffectRequest(bucket);
request.objectKey = "folder/image.jpg";
// 设置您可以通过填写 detectUrl 处理任意公网可访问的图片链接。不填写 detectUrl 时，后台会默认处理 objectKey ，填写了 detectUrl 时，后台会处理 detectUrl 链接，无需再填写 objectKey，detectUrl 示例：http://www.example.com/abc.jpg。
request.detectUrl = "http://www.example.com/abc.jpg";
request.type = "face-segmentation";// 设置人脸特效类型，人脸美颜：face-beautify；人脸性别转换：face-gender-transformation；人脸年龄变化：face-age-transformation；人像分割：face-segmentation
request.whitening = 30;// 设置type为face-beautify时生效，美白程度，取值范围[0,100]。0不美白，100代表最高程度。默认值30
request.smoothing = 10;// 设置type为face-beautify时生效，磨皮程度，取值范围[0,100]。0不磨皮，100代表最高程度。默认值10
request.faceLifting = 70;// 设置type为face-beautify时生效，瘦脸程度，取值范围[0,100]。0不瘦脸，100代表最高程度。默认值70
request.eyeEnlarging = 70;// 设置type为face-beautify时生效，大眼程度，取值范围[0,100]。0不大眼，100代表最高程度。默认值70
```

```
request.gender = 1;// 设置type为face-gender-transformation时生效，选择转换方向，0：
男变女，1：女变男。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。
request.age = 18;// 设置type为face-age-transformation时生效，变化到的人脸年龄，
[10,80]。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。

cosXmlService.aiFaceEffectAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 人脸特效的结果
        // 详细字段请查看api文档或者SDK源码
        AIFaceEffectResult result = (AIFaceEffectResult) cosResult;

    }
    @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。

人体识别

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于人体识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
人体识别	腾讯云数据万象通过 AIBodyRecognition 接口识别并输出画面中人体，输出其位置（矩形框）和置信度。图片人体识别请求包属于 GET 请求，请求时需要携带签名

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

人体识别

功能说明

腾讯云数据万象通过 AIBodyRecognition 接口识别并输出画面中人体，输出其位置（矩形框）和置信度。图片人体识别请求包属于 GET 请求，请求时需要携带签名。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AIBodyRecognitionRequest request = new AIBodyRecognitionRequest(bucket);
// 存储桶中的位置标识符，即对象键
request.objectKey = "folder/document.jpg";
// 设置您可以通过填写 detectUrl 处理任意公网可访问的图片链接。不填写 detectUrl 时，后台会默认处理 objectKey ，填写了 detectUrl 时，后台会处理 detectUrl 链接，无需再填写 objectKey，detectUrl 示例：http://www.example.com/abc.jpg。
request.detectUrl = "http://www.example.com/abc.jpg";

cosXmlService.aiBodyRecognitionAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 人体识别的结果
        // 详细字段请查看api文档或者SDK源码
        AIBodyRecognitionResult result = (AIBodyRecognitionResult) cosResult;
```

```
    }
    @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。

宠物识别

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于宠物识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
宠物识别	腾讯云数据万象通过 detect-pet 接口识别并输出画面中宠物，输出其位置（矩形框）和置信度。图片宠物识别请求包属于 GET 请求，请求时需要携带签名

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

宠物识别

功能说明

腾讯云数据万象通过 detect-pet 接口识别并输出画面中宠物，输出其位置（矩形框）和置信度。图片宠物识别请求包属于 GET 请求，请求时需要携带签名。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AIDetectPetRequest request = new AIDetectPetRequest(bucket,
"folder/document.jpg");

cosXmlService.aiDetectPetAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 宠物识别的结果
        // 详细字段请查看api文档或者SDK源码
        AIDetectPetResult result = (AIDetectPetResult) cosResult;

    }
    @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。

车辆车牌识别

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于车辆车牌识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
车辆车牌识别	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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

车辆车牌识别

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。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AIDetectCarRequest request = new AIDetectCarRequest(bucket,
"folder/document.jpg");

cosXmlService.aiDetectCarAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 车辆车牌识别的结果
        // 详细字段请查看api文档或者SDK源码
        AIDetectCarResult result = (AIDetectCarResult) cosResult;

    }
    @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。

卡证识别

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于卡证识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
卡证识别	本接口支持中国大陆居民二代身份证正面（暂不支持背面）、驾驶证主页（暂不支持副页）所有字段的自动定位，暂不支持文本识别，用于对特定字段的抹除、屏蔽，以及进一步的文本识别

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

卡证识别

功能说明

本接口支持中国大陆居民二代身份证正面（暂不支持背面）、驾驶证主页（暂不支持副页）所有字段的自动定位，暂不支持文本识别，用于对特定字段的抹除、屏蔽，以及进一步的文本识别。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AILicenseRecRequest request = new AILicenseRecRequest(bucket,
"folder/document.jpg");
request.detectUrl = "http://www.example.com/abc.jpg";// 设置您可以通过填写
detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey
，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey detect-url
示例：http://www.example.com/abc.jpg
request.cardType = "IDCard";// 设置卡证识别类型，有效值为IDCard，DriverLicense。
<br>IDCard表示身份证；DriverLicense表示驾驶证，默认：DriverLicense

cosXmlService.aiLicenseRecAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 卡证识别的结果
        // 详细字段请查看api文档或者SDK源码
        AILicenseRecResult result = (AILicenseRecResult) cosResult;
```

```
    }
    @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。

游戏场景识别

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于游戏场景识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
游戏场景识别	游戏标签功能实现游戏图片场景的识别，返回图片中置信度较高的游戏类别标签。游戏标签识别请求包属于 GET 请求，请求时需要携带签名

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

游戏场景识别

功能说明

游戏标签功能实现游戏图片场景的识别，返回图片中置信度较高的游戏类别标签。游戏标签识别请求包属于 GET 请求，请求时需要携带签名。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AIGameRecRequest request = new AIGameRecRequest(bucket);
// 存储桶中的位置标识符，即对象键
request.objectKey = "folder/document.jpg";
// 设置您可以通过填写 detectUrl 处理任意公网可访问的图片链接。不填写 detectUrl 时，后台会默认处理 objectKey ，填写了 detectUrl 时，后台会处理 detectUrl 链接，无需再填写 objectKey，detectUrl 示例：http://www.example.com/abc.jpg。
request.detectUrl = "http://www.example.com/abc.jpg";

cosXmlService.aiGameRecAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 游戏场景识别的结果
        // 详细字段请查看api文档或者SDK源码
        AIGameRecResult result = (AIGameRecResult) cosResult;
```

```
    }
    @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。

商品抠图

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于商品抠图的相关 API 概览以及 SDK 示例代码。

API	操作描述
商品抠图	腾讯云数据万象通过 GoodsMatting 接口检测图片中的商品信息，生成只包含商品信息的图片，支持持久化、云上处理及下载时处理。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

商品抠图

功能说明

腾讯云数据万象通过 GoodsMatting 接口检测图片中的商品信息，生成只包含商品信息的图片，支持持久化、云上处理及下载时处理。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
GoodsMattingRequest request = new GoodsMattingRequest(bucket,
"folder/document.jpg");
// 设置结果文件保存路径
request.saveLocalPath = "保存到本地的路径";

cosXmlService.goodsMattingAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 商品抠图的结果
        // 详细字段请查看api文档或者SDK源码
        GetObjectResult result = (GetObjectResult) cosResult;

    }
    @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。

人脸核身
身份证识别

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于身份证识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
身份证识别	支持中国大陆居民二代身份证正反面所有字段的识别，包括姓名、性别、民族、出生日期、住址、公民身份证号、签发机关、有效期限；具备身份证照片、人像照片的裁剪功能和翻拍、PS、复印件告警功能，以及边框和框内遮挡告警、临时身份证告警和身份证有效期不合法告警等扩展功能。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

身份证识别

功能说明

支持中国大陆居民二代身份证正反面所有字段的识别，包括姓名、性别、民族、出生日期、住址、公民身份证号、签发机关、有效期限；具备身份证照片、人像照片的裁剪功能和翻拍、PS、复印件告警功能，以及边框和框内遮挡告警、临时身份证告警和身份证有效期不合法告警等扩展功能。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
AIIDCardOCRRequest request = new AIIDCardOCRRequest(bucket,
"folder/document.jpg");
request.cardSide = "FRONT";// 设置FRONT：身份证有照片的一面（人像面）BACK：身份证有国徽的一面（国徽面）该参数如果不填，将为您自动判断身份证正反面
request.config = "{\"CropIdCard\":true,\"CropPortrait\":true}";// 设置以下可选字段均为 bool 类型，默认 false：CropIdCard，身份证照片裁剪（去掉证件外多余的边缘、自动矫正拍摄角度）CropPortrait，人像照片裁剪（自动抠取身份证头像区域）CopyWarn，复印件告警BorderCheckWarn，边框和框内遮挡告警ReshootWarn，翻拍告警DetectPsWarn，PS 检测告警TempIdWarn，临时身份证告警InvalidDateWarn，身份证有效日期不合法告警Quality，图片质量分数（评价图片的模糊程度）MultiCardDetect，是否开启多卡证检测参数设置方式参考：Config = {"CropIdCard":true,"CropPortrait":true}
```

```
cosXmlService.aiIDCardOCRAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 身份证识别的结果
        // 详细字段请查看api文档或者SDK源码
        AIIDCardOCRResult result = (AIIDCardOCRResult) cosResult;

    }
    @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。

活体人脸核身

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于活体人脸核身的相关 API 概览以及 SDK 示例代码。

API	操作描述
活体人脸核身	集成了活体检测和跟权威库进行比对的能力，传入一段视频和姓名、身份证号信息即可进行验证。对录制的自拍视频进行活体检测，从而确认当前用户为真人，可防止照片、视频、静态3D建模等各种不同类型的攻击。检测为真人后，再判断该视频中的人与权威库的证件照是否属于同一个人，实现用户身份信息核实。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

活体人脸核身

功能说明

集成了活体检测和跟权威库进行比对的能力，传入一段视频和姓名、身份证号信息即可进行验证。对录制的自拍视频进行活体检测，从而确认当前用户为真人，可防止照片、视频、静态3D建模等各种不同类型的攻击。检测为真人后，再判断该视频中的人与权威库的证件照是否属于同一个人，实现用户身份信息核实。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
LivenessRecognitionRequest request = new LivenessRecognitionRequest(bucket,
"folder/document.mp4");
request.idCard = "身份证号";// 设置身份证号
request.name = "姓名";// 设置姓名
request.livenessType = "ACTION";// 设置活体检测类型，取值：LIP/ACTION/SILENTLIP 为
数字模式，ACTION 为动作模式，SILENT 为静默模式，三种模式选择一种传入
request.validateData = "1,2";// 设置数字模式传参：数字验证码（1234），需先调用接口获取
数字验证码动作模式传参：传动作顺序（2，1 or 1，2），需先调用接口获取动作顺序静默模式传参：空
request.bestFrameNum = 1;// 设置需要返回多张最佳截图，取值范围1 - 10，不设置默认返回一
张最佳截图

cosXmlService.livenessRecognitionAsync(request, new CosXmlResultListener() {
```

```
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 活体人脸核身的结果
        // 详细字段请查看api文档或者SDK源码
        LivenessRecognitionResult result = (LivenessRecognitionResult)
cosResult;

    }
    @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。

获取数字验证码

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于获取数字验证码的相关 API 概览以及 SDK 示例代码。

API	操作描述
获取数字验证码	使用数字活体检测模式前，需调用本接口获取数字验证码

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

获取数字验证码

功能说明

使用数字活体检测模式前，需调用本接口获取数字验证码。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
GetLiveCodeRequest request = new GetLiveCodeRequest(bucket);

cosXmlService.getLiveCodeAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 获取数字验证码的结果
        // 详细字段请查看api文档或者SDK源码
        GetLiveCodeResult result = (GetLiveCodeResult) cosResult;

    }
    @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。

获取动作顺序

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于获取动作顺序的相关 API 概览以及 SDK 示例代码。

API	操作描述
获取动作顺序	使用动作活体检测模式前，需调用本接口获取动作顺序

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

获取动作顺序

功能说明

使用动作活体检测模式前，需调用本接口获取动作顺序。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
GetActionSequenceRequest request = new GetActionSequenceRequest(bucket);

cosXmlService.getActionSequenceAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 获取动作顺序的结果
        // 详细字段请查看api文档或者SDK源码
        GetActionSequenceResult result = (GetActionSequenceResult) cosResult;

    }
    @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。

内容翻译

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于内容翻译的相关 API 概览以及 SDK 示例代码。

API	操作描述
提交任务	提交一个翻译任务

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

提交任务

功能说明

提交一个翻译任务。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
PostTranslationRequest request = new PostTranslationRequest(bucket);
PostTranslation postTranslation = new PostTranslation();// 提交任务请求体
request.setPostTranslation(postTranslation);// 设置请求
PostTranslation.PostTranslationInput postTranslationInput = new
PostTranslation.PostTranslationInput();
postTranslation.input = postTranslationInput;
// 设置源文档文件名单文件（docx/xlsx/html/markdown/txt）：不超过800万字符有页数的
（pdf/pptx）：不超过300页文本文件（txt）：不超过10MB二进制文件（pddocx/pptx/xlsx）：不
超过60MB图片文件（jpg/jpeg/png/webp）：不超过10MB;是否必传：是
postTranslationInput.object = "en.pdf";
// 设置文档语言类型zh：简体中文zh-hk：繁体中文zh-tw：繁体中文zh-tr：繁体中文en：英语ar：
阿拉伯语de：德语es：西班牙语fr：法语id：印尼语it：意大利语ja语pt：葡萄牙语ru：俄语ko：韩语
km：高棉语lo：老挝语;是否必传：是
postTranslationInput.lang = "en";
// 设置文档类型pdfdocxpptxxlsxtxtxmlhtml：只能翻译 HTML 里的文本节点，需要通过 JS 动态
加载的不进行翻译markdownjpgjpegpngwebp;是否必传：是
postTranslationInput.type = "pdf";
```

```
// 设置原始文档类型仅在 Type=pdf/jpg/jpeg/png/webp 时使用，当值为pdf时，仅支持 docx、
pptx当值为jpg/jpeg/png/webp时，仅支持txt;是否必传：否
postTranslationInput.basicType = "pptx";
PostTranslation.PostTranslationOperation postTranslationOperation = new
PostTranslation.PostTranslationOperation();
postTranslation.operation = postTranslationOperation;
PostTranslation.PostTranslationTranslation postTranslationTranslation = new
PostTranslation.PostTranslationTranslation();
postTranslationOperation.translation = postTranslationTranslation;
// 设置目标语言类型源语言类型为 zh/zh-hk/zh-tw/zh-tr 时支持：en、ar、de、es、fr、id、
it、ja、it、ru、ko、km、lo、pt源语言类型为 en 时支持：zzh-hk、zh-tw、zh-tr、ar、de、
es、fr、id、it、ja、it、ru、ko、km、lo、pt其他类型时支持：zh、zh-hk、zh-tw、zh-tr、
en;是否必传：是
postTranslationTranslation.lang = "zh";
// 设置文档类型，源文件类型与目标文件类型映射关系如下：docx：docxpptx：pptxxlsx：
xlsxtxt：txtxml：xmlhtml：htmlmarkdown：markdownpdf：pddocxpng：txtjpg：txtjpeg：
txtwebp：txt;是否必传：是
postTranslationTranslation.type = "pdf";
PostTranslation.PostTranslationOutput postTranslationOutput = new
PostTranslation.PostTranslationOutput();
postTranslationOperation.output = postTranslationOutput;
// 设置存储桶的地域;是否必传：是
postTranslationOutput.region = "ap-chongqing";
// 设置存储结果的存储桶;是否必传：是
postTranslationOutput.bucket = "test-123456789";
// 设置输出结果的文件名;是否必传：是
postTranslationOutput.object = "output/zh.pdf";
// 设置透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
postTranslationOperation.userData = "This is my data.";
// 设置任务优先级，级别限制：0 、1 、2 。级别越大任务优先级越高，默认为0;是否必传：否
postTranslationOperation.jobLevel = "0";
// 设置任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
postTranslation.callBackFormat = "JSON";
// 设置任务回调地址，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;是
否必传：否
postTranslation.callBack = "http://callback.demo.com";

ciService.postTranslationAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 提交任务的结果
        // 详细字段请查看api文档或者SDK源码
        PostTranslationResult result = (PostTranslationResult) cosResult;

    }
    @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。

词性分析

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于词性分析的相关 API 概览以及 SDK 示例代码。

API	操作描述
提交任务	提交一个分词任务

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

提交任务

功能说明

提交一个分词任务。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
CreateWordsGeneralizeJobRequest request = new
CreateWordsGeneralizeJobRequest(bucket);
// 设置文件路径;是否必传：是
request.setInputObject("text.txt");
// 设置任务所在的队列 ID
request.setQueueId("queueId");
// 设置任务回调地址，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;是
否必传：否
request.setCallBack("http://callback.demo.com");
// 设置任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
request.setCallBackFormat("JSON");
// 设置透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
request.setUserData("This is my data");
// 设置任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0;是否必传：否
request.setJobLevel(0);
// 设置ner 方式，支持 NerBasic 和 DL，默认值 DL;是否必传：否
request.setNerMethod("DL");
```

```
// 设置分词粒度，支持 SegBasic 和 MIX，默认值 MIX;是否必传：否
request.setSegMethod("MIX");

ciService.createWordsGeneralizeJobAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 提交任务的结果
        // 详细字段请查看api文档或者SDK源码
        CreateWordsGeneralizeJobResult result =
(CreateWordsGeneralizeJobResult) cosResult;

    }
    @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。

视频目标检测
创建视频目标检测模板

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于创建视频目标检测模板的相关 API 概览以及 SDK 示例代码。

API	操作描述
创建模板	创建视频目标检测模板

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

创建模板

功能说明

创建视频目标检测模板。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
PostVideoTargetTempleteRequest request = new
PostVideoTargetTempleteRequest(bucket);
PostVideoTargetTemplete postVideoTargetTemplete = new
PostVideoTargetTemplete();// 创建模板请求体
request.setPostVideoTargetTemplete(postVideoTargetTemplete);// 设置请求
// 设置模板名称，仅支持中文、英文、数字、_、-和*，长度不超过 64;是否必传：是
postVideoTargetTemplete.name = "TemplateName";
VideoTargetRec videoTargetRec = new VideoTargetRec();
postVideoTargetTemplete.videoTargetRec = videoTargetRec;
// 设置是否开启人体检测，取值 true/false;是否必传：否
videoTargetRec.body = "true";
// 设置是否开启宠物检测，取值 true/false;是否必传：否
videoTargetRec.pet = "false";
// 设置是否开启车辆检测，取值 true/false;是否必传：否
videoTargetRec.car = "true";
```

```
ciService.postVideoTargetTempleteAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 创建模板的结果
        // 详细字段请查看api文档或者SDK源码
        PostVideoTargetTempleteResult result = (PostVideoTargetTempleteResult)
cosResult;

    }
    @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。

更新视频目标检测模板

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于更新视频目标检测模板的相关 API 概览以及 SDK 示例代码。

API	操作描述
更新模板	更新视频目标检测模板

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

更新模板

功能说明

更新视频目标检测模板。

> **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
UpdateVideoTargetTempleteRequest request = new
UpdateVideoTargetTempleteRequest(bucket, "TemplateId");
UpdateVideoTargetTemplete updateVideoTargetTemplete = new
UpdateVideoTargetTemplete();// 更新模板请求体
request.setUpdateVideoTargetTemplete(updateVideoTargetTemplete);// 设置请求
// 设置模板名称，仅支持中文、英文、数字、_、-和*，长度不超过 64;是否必传：是
updateVideoTargetTemplete.name = "TemplateName";
VideoTargetRec videoTargetRec = new VideoTargetRec();
updateVideoTargetTemplete.videoTargetRec = videoTargetRec;
// 设置是否开启人体检测，取值 true/false;是否必传：否
videoTargetRec.body = "true";
// 设置是否开启宠物检测，取值 true/false;是否必传：否
videoTargetRec.pet = "false";
// 设置是否开启车辆检测，取值 true/false;是否必传：否
videoTargetRec.car = "true";

ciService.updateVideoTargetTempleteAsync(request, new CosXmlResultListener() {
```

```
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 更新模板的结果
        // 详细字段请查看api文档或者SDK源码
        UpdateVideoTargetTempleteResult result =
(UpdateVideoTargetTempleteResult) cosResult;

    }
    @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。

提交视频目标检测任务

最近更新时间：2024-12-27 16:37:03

简介

本文档提供关于提交一个视频目标检测任务的相关 API 概览以及 SDK 示例代码。

API	操作描述
提交任务	提交一个视频目标检测任务

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

提交任务

功能说明

提交一个视频目标检测任务。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
PostVideoTargetRecRequest request = new PostVideoTargetRecRequest(bucket);
PostVideoTargetRec postVideoTargetRec = new PostVideoTargetRec();// 提交任务请求
体
request.setPostVideoTargetRec(postVideoTargetRec);// 设置请求
PostVideoTargetRec.PostVideoTargetRecOperation postVideoTargetRecOperation =
new PostVideoTargetRec.PostVideoTargetRecOperation();
postVideoTargetRec.operation = postVideoTargetRecOperation;
// 设置视频目标检测模板 ID;是否必传：否
postVideoTargetRecOperation.templateId = "t1460606b9752148c4ab182f55163ba7cd";
// 设置透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
postVideoTargetRecOperation.userData = "This is my data.";
// 设置任务优先级，级别限制：0 、1 、2 。级别越大任务优先级越高，默认为0;是否必传：否
postVideoTargetRecOperation.jobLevel = "0";
PostVideoTargetRec.PostVideoTargetRecInput postVideoTargetRecInput = new
PostVideoTargetRec.PostVideoTargetRecInput();
postVideoTargetRec.input = postVideoTargetRecInput;
// 设置媒体文件名;是否必传：否
```

```
postVideoTargetRecInput.object = "input/demo.mp4";
// 设置任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
postVideoTargetRec.callBackFormat = "JSON";
// 设置任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
postVideoTargetRec.callBackType = "Url";
// 设置任务回调地址，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;是
否必传：否
postVideoTargetRec.callBack = "http://callback.demo.com";

ciService.postVideoTargetRecAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 提交任务的结果
        // 详细字段请查看api文档或者SDK源码
        PostVideoTargetRecResult result = (PostVideoTargetRecResult)
cosResult;

    }
    @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。

视频人像分割
提交视频人像抠图任务

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于视频人像分割的相关 API 概览以及 SDK 示例代码。

API	操作描述
提交任务	提交一个视频人像抠图任务

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 。

提交任务

功能说明

提交一个视频人像抠图任务。

> ⚠ **注意：**
> COS Android SDK 版本需要大于等于 v5.9.32。

示例代码

```
// 存储桶名称，格式为 BucketName-APPID
String bucket = "examplebucket-1250000000";
PostSegmentVideoBodyRequest request = new PostSegmentVideoBodyRequest(bucket);
PostSegmentVideoBody postSegmentVideoBody = new PostSegmentVideoBody();// 提交
任务请求体
request.setPostSegmentVideoBody(postSegmentVideoBody);// 设置请求
PostSegmentVideoBody.PostSegmentVideoBodyInput postSegmentVideoBodyInput = new
PostSegmentVideoBody.PostSegmentVideoBodyInput();
postSegmentVideoBody.input = postSegmentVideoBodyInput;
// 设置文件路径;是否必传：是
postSegmentVideoBodyInput.object = "input/test.mp4";
PostSegmentVideoBody.PostSegmentVideoBodyOperation
postSegmentVideoBodyOperation = new
PostSegmentVideoBody.PostSegmentVideoBodyOperation();
postSegmentVideoBody.operation = postSegmentVideoBodyOperation;
PostSegmentVideoBody.PostSegmentVideoBodySegmentVideoBody
postSegmentVideoBodySegmentVideoBody = new
```

```
PostSegmentVideoBody.PostSegmentVideoBodySegmentVideoBody();
postSegmentVideoBodyOperation.segmentVideoBody =
postSegmentVideoBodySegmentVideoBody;
// 设置抠图模式 Mask：输出alpha通道结果Foreground：输出前景视频Combination：输出抠图后的前景与自定义背景合成后的视频默认值：Mask;是否必传：否
postSegmentVideoBodySegmentVideoBody.mode = "Mask";
// 设置抠图类型HumanSeg：人像抠图GreenScreenSeg：绿幕抠图SolidColorSeg：纯色背景抠图默认值：HumanSeg;是否必传：否
postSegmentVideoBodySegmentVideoBody.segmentType = "HumanSeg";
// 设置mode为 Foreground 时参数生效，背景颜色为红色，取值范围 [0, 255]， 默认值为 0;是否必传：否
postSegmentVideoBodySegmentVideoBody.backgroundRed = "0";
// 设置mode为 Foreground 时参数生效，背景颜色为绿色，取值范围 [0, 255]，默认值为 0;是否必传：否
postSegmentVideoBodySegmentVideoBody.backgroundGreen = "0";
// 设置mode为 Foreground 时参数生效，背景颜色为蓝色，取值范围 [0, 255]，默认值为 0;是否必传：否
postSegmentVideoBodySegmentVideoBody.backgroundBlue = "0";
// 设置调整抠图的边缘位置，取值范围为[0, 255]，默认值为 0;是否必传：否
postSegmentVideoBodySegmentVideoBody.binaryThreshold = "0";
// 设置纯色背景抠图的背景色（红），当 SegmentType 为 SolidColorSeg 生效，取值范围为[0, 255]，默认值为 0;是否必传：否
postSegmentVideoBodySegmentVideoBody.removeRed = "0";
// 设置纯色背景抠图的背景色（绿），当 SegmentType 为 SolidColorSeg 生效，取值范围为[0, 255]，默认值为 0;是否必传：否
postSegmentVideoBodySegmentVideoBody.removeGreen = "0";
// 设置纯色背景抠图的背景色（蓝），当 SegmentType 为 SolidColorSeg 生效，取[illegible]范围为[0, 255]，默认值为 0;是否必传：否
postSegmentVideoBodySegmentVideoBody.removeBlue = "0";
PostSegmentVideoBody.PostSegmentVideoBodyOutput postSegmentVideoBodyOutput =
new PostSegmentVideoBody.PostSegmentVideoBodyOutput();
postSegmentVideoBodyOperation.output = postSegmentVideoBodyOutput;
// 设置存储桶的地域;是否必传：是
postSegmentVideoBodyOutput.region = "ap-chongqing";
// 设置存储结果的存储桶;是否必传：是
postSegmentVideoBodyOutput.bucket = "test-123456789";
// 设置结果文件名;是否必传：是
postSegmentVideoBodyOutput.object = "output/out.mp4";
// 设置透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
postSegmentVideoBodyOperation.userData = "This is my data.";
// 设置任务优先级，级别限制：0 、1 、2。级别越大任务优先级越高，默认为0;是否必传：否
postSegmentVideoBodyOperation.jobLevel = "0";
// 设置任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
postSegmentVideoBody.callBackFormat = "JSON";
// 设置任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
postSegmentVideoBody.callBackType = "Url";
```

```
// 设置任务回调地址，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;是否必传：否
postSegmentVideoBody.callBack = "http://callback.demo.com";

ciService.postSegmentVideoBodyAsync(request, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult cosResult) {
        // result 提交任务的结果
        // 详细字段请查看api文档或者SDK源码
        PostSegmentVideoBodyResult result = (PostSegmentVideoBodyResult) cosResult;

    }
    @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。

C SDK
快速入门

最近更新时间：2024-12-27 16:37:04

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：

```
./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。

```
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，涉及网络，内存等部分 */
    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 */
    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_cname, curl参数等全局配置信息
 * 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，不再需要单独释放内存
 */
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 依据用户所在地域的 COS 服务域名填写
cos_str_set(&options->config->access_key_id, "<用户的SecretId>");          //用户注册 COS 服务后所获得的 SecretId
cos_str_set(&options->config->access_key_secret, "<用户的SecretKey>");     //用户注册 COS 服务后所获得的 SecretKey
cos_str_set(&options->config->sts_token, "<用户的StsToken>");              //临时密钥的 Token，临时密钥生成和使用指引参见 https://cloud.tencent.com/document/product/436/14048
cos_str_set(&options->config->appid, "<用户的AppId>");                     //用户注册 COS 服务后所获得的 AppId
```

```
/* 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_secret 均需要设置为对应临时密钥所配套的 SecretId 和 SecretKey */
//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 Content-MD5 头部，enable 为 COS_TRUE 时上传请求将自动添加Content-MD5 头部，如
果不设置此项则默认将添加 Content-MD5 头部 */
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_cname，curl参数等全局配置信息
 * 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
再需要单独释放内存
 */
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_cname, curl参数等全局配置信息
 * 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
需要单独释放内存
 */
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_cname，curl参数等全局配置信息
 * 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，不再需要单独释放内存
 */
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_cname, curl参数等全局配置信息
```

```
 * 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
单独释放内存
 */
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_headers);
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_cname, curl 参数等全局配置信息
 * 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
需要单独释放内存
 */
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);
```

内容识别
二维码识别

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于图片二维码相关的 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

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)
```

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。

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://cloud.tencent.com/document/product/436/6224 文档
```

```
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 数据万象的访问域名，详情请参见 https://cloud.tencent.com/document/product/460/31066 文档
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.cloud.tencent.com/cam/capi 页面获取
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.cloud.tencent.com/cam/capi 页面获取
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666，可在 https://console.cloud.tencent.com/cos5/bucket 查看
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_qrcode()
{
    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 *headers = NULL;
    ci_operation_result_t *results = NULL;
    ci_qrcode_info_t *content = 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\":
[{\"fileid\":\"test.png\",\"rule\":\"QRcode/cover/1\"}]}");
    // 上传时识别
    cos_str_set(&file, "test.jpg");
    cos_str_set(&object, "test.jpg");
    s = ci_put_object_from_file(options, &bucket, &object, &file, headers,
&resp_headers, &results);
    log_status(s);
    if (!cos_status_is_ok(s)) {
        printf("put object failed\n");
    }
    printf("CodeStatus: %d\n", results->object.code_status);
    cos_list_for_each_entry(ci_qrcode_info_t, content, &results-
>object.qrcode_info, node) {
        printf("CodeUrl: %s\n", content->code_url.data);
        printf("Point: %s\n", content->point[0].data);
        printf("Point: %s\n", content->point[1].data);
        printf("Point: %s\n", content->point[2].data);
        printf("Point: %s\n", content->point[3].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_ci_image_qrcode();

    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

下载时识别

功能说明

下载文件时进行二维码识别。

方法原型

```
cos_status_t *ci_get_qrcode(const cos_request_options_t *options,
                            const cos_string_t *bucket,
                            const cos_string_t *object,
                            int cover,
                            cos_table_t *headers,
                            cos_table_t *query_params,
                            cos_table_t **resp_headers,
                            ci_qrcode_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://cloud.tencent.com/document/product/436/6224 文档
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 数据万象的访问域名，详情请参见
https://cloud.tencent.com/document/product/460/31066 文档
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.cloud.tencent.com/cam/capi 页
面获取
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.cloud.tencent.com/cam/capi 页面获取
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-
1253666666，可在 https://console.cloud.tencent.com/cos5/bucket 查看
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_get_image_qrcode()
{
    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;
    ci_qrcode_info_t *content = NULL;
    ci_qrcode_result_t *result2 = 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");

    // 下载时识别
    s = ci_get_qrcode(options, &bucket, &object, 1, NULL, NULL, &resp_headers,
&result2);
    log_status(s);
    if (!cos_status_is_ok(s)) {
        printf("get object failed\n");
    }
    printf("CodeStatus: %d\n", result2->code_status);
    cos_list_for_each_entry(ci_qrcode_info_t, content, &result2->qrcode_info,
node) {
        printf("CodeUrl: %s\n", content->code_url.data);
        printf("Point: %s\n", content->point[0].data);
        printf("Point: %s\n", content->point[1].data);
        printf("Point: %s\n", content->point[2].data);
        printf("Point: %s\n", content->point[3].data);
    }
```

```
    printf("ImageResult: %s\n", result2->result_image.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_ci_get_image_qrcode();

    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
cover	二维码覆盖功能，对识别出的二维码覆盖上马赛克。取值为0或1。0表示不开启二维码覆盖，1表示开启二维码覆盖	int
headers	COS 请求附加头域	Struct

query_params	COS 请求操作参数	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct
results	二维码识别结果	Struct

C++ SDK
快速入门

最近更新时间：2024-12-27 16:37:04

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。
> 如果您仍在使用 JSON 版本 SDK，请 升级到 XML C++ SDK 。

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。

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。

常见编译问题

1. 编译可执行程序的时候提示错误：

```
PocoCrypto.so.64: undefined reference to
`PEM_write_bio_PrivateKey@libcrypto.so.10'
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。

```
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
```

2. 编译 poco 库的时候无法编译出 PocoNetSSL 库，一般是因为机器没装 openssl-devel 库。

```
yum install -y openssl-devel
```

3. 编译可执行程序的时候提示错误：

```
undefined reference to
`qcloud_cos::CosConfig::CosConfig(std::__cxx11::basic_string<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。

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。

配置文件各字段介绍：

```
{
"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
"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
"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，参考
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”
}
```

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[]) {
```

```
    qcloud_cos::CosConfig config("./config.json");
    // 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://cloud.tencent.com/document/product/436/14048
    config.SetTmpToken("xxx");
    qcloud_cos::CosAPI cos(config);
}
```

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.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_display_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，比如
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
    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，比如 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.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
    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。
    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.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;
    }
}
```

.NET(C#) SDK 快速入门

最近更新时间：2025-04-29 10:20:42

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。

步骤1：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 4.5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 。

步骤2：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：永久密钥、持续更新的临时密钥、不变的临时密钥。

方式1：永久密钥

```
string secretId = "SECRET_ID"; //"云 API 密钥 SecretId";
string secretKey = "SECRET_KEY"; //"云 API 密钥 SecretKey";
long durationSecond = 600;  //每次请求签名有效时长，单位为秒
QCloudCredentialProvider cosCredentialProvider = new
DefaultQCloudCredentialProvider(
  secretId, secretKey, durationSecond);
```

方式2：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 :
DefaultSessionQCloudCredentialProvider
{

  // 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";
    string tmpSecretKey = "SECRET_KEY"; //"临时密钥 SecretKey";
    string tmpToken = "COS_TOKEN"; //"临时密钥 token";
    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3：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
DefaultSessionQCloudCredentialProvider(
  tmpSecretId, tmpSecretKey, tmpExpireTime, tmpToken);
```

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);
```

步骤3：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
/ 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);
}
```

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
/ total));
};

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());
}
```

内容识别
图片二维码

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于图片二维码相关的 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

二维码识别功能支持在文件上传时进行识别。

示例代码

```
PutObjectRequest request = new PutObjectRequest(bucket, key, srcPath);

JObject o = new JObject();
// 不返回原图
o["is_pic_info"] = 1;
JArray rules = new JArray();
JObject rule = new JObject();
rule["bucket"] = bucket;
rule["fileid"] = "qrcode.jpg";
//处理参数，规则参见：https://cloud.tencent.com/document/product/436/54070
rule["rule"] = "QRcode/cover/<mode>";
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。

下载时识别二维码

功能说明

二维码识别功能可支持在下载时进行识别。

示例代码

```
//二维码覆盖功能，将对识别出的二维码覆盖上马赛克。取值为0或1。0表示不开启二维码覆盖，1表示开启二维码覆盖，默认值0
QRCodeRecognitionRequest request = new QRCodeRecognitionRequest(bucket, key, 0);

QRCodeRecognitionResult result = cosXml.QRCodeRecognition(request);
```

> **说明**
> 更多完整示例，请前往 GitHub 查看。

Go SDK
快速入门

最近更新时间：2024-12-27 16:37:04

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 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.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://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:  "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.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.cloud.tencent.com/cam/capi
    },
})
```

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.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://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://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://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://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://cloud.tencent.com/document/product/436/14048
        SecretID:     "SECRETID",
        SecretKey:    "SECRETKEY",
        SessionToken: "SECRETTOKEN",
    },
})
// 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.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://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:  "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.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.cloud.tencent.com/cam/capi
        },
    })

    _, 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:  "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.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.cloud.tencent.com/cam/capi
        },
    })


    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.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://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:  "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.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.cloud.tencent.com/cam/capi
        },
    })
    // 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.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://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:  "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.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.cloud.tencent.com/cam/capi
        },
    })


    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.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://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:  "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.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.cloud.tencent.com/cam/capi
        },
```

```
    })
    // 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

```
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.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://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:  "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.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.cloud.tencent.com/cam/capi
        },
    })
    name := "test/objectPut.go"
    _, err := c.Object.Delete(context.Background(), name)
    if err != nil {
        panic(err)
    }
}
```

开通服务

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于数据万象服务开通、查询、解绑的 API 概览以及 SDK 示例代码。

API	操作描述
绑定数据万象服务	用于开通数据万象服务，需要绑定对象存储（Cloud Object Storage，COS）上已有的存储桶。
查询数据万象服务	用于查询该 Bucket 是否已开通数据万象服务。
解绑数据万象服务	解绑开通数据万象服务的 Bucket（Bucket 及桶中文件会保留在对象存储 COS）。

绑定数据万象服务

功能说明

用于开通数据万象服务。

方法原型

```
func (s *CIService) OpenCIService(ctx context.Context) (*Response, error)
```

请求示例

```
_, err := c.CI.OpenCIService(context.Background())
```

查询数据万象服务

功能说明

用于查询该 Bucket 是否已开通数据万象服务。

方法原型

```
func (s *CIService) GetCIService(ctx context.Context) (*CIServiceResult,
*Response, error)
```

请求示例

```
res, _, err := c.CI.GetCIService(context.Background())
```

参数说明

参数名称	参数描述
CIServiceResult	查询数据万象的结果

解绑数据万象服务

功能说明

解绑开通数据万象服务的 Bucket。

方法原型

```
func (s *CIService) CloseCIService(ctx context.Context) (*Response, error)
```

请求示例

```
_, err := c.CI.CloseCIService(context.Background())
```

AI 内容识别
图片上色

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于图片上色的相关 API 概览以及 SDK 示例代码。

API	操作描述
图片上色	腾讯云数据万象通过 AIImageColoring 接口对黑白图像进行上色。

图片上色

腾讯云数据万象通过 AIImageColoring 接口对黑白图像进行上色。

方法原型

```
func (s *CIService) GetAIImageColoringV2(ctx context.Context, name string, opt
*AIImageColoringOptions) (*Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        }
})
obj := "pic/heibai.jpeg"
localPath := "test.jpeg"
opt := &cos.AIImageColoringOptions{}
resp, err := c.CI.GetAIImageColoringV2(context.Background(), obj, opt)
log_status(err)
if err == nil {
    fd, _ := os.OpenFile(localPath, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0660)
    io.Copy(fd, resp.Body)
    fd.Close()
}
```

参数说明

```
type AIImageColoringOptions struct {
    DetectUrl string `url:"detect-url,omitempty"`
}
```

参数名称	参数描述	类型	是否必填
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 name，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 name。	string	否

结果说明

处理后的图片直接返回至响应体。

图像增强

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于图像增强的相关 API 概览以及 SDK 示例代码。

API	操作描述
图像增强	腾讯云数据万象通过 AIEnhanceImage 接口对图像进行增强处理。

图像增强

腾讯云数据万象通过 AIEnhanceImage 接口对图像进行增强处理。

方法原型

```
func (s *CIService) GetAIEnhanceImageV2(ctx context.Context, name string, opt
*AIEnhanceImageOptions) (*Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        }
})
obj := "pic/heibai.jpeg"
localPath := "test.jpeg"
opt := &cos.AIEnhanceImageOptions{}
resp, err := c.CI.GetAIEnhanceImageV2(context.Background(), obj, opt)
log_status(err)
if err == nil {
    fd, _ := os.OpenFile(localPath, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0660)
    io.Copy(fd, resp.Body)
    fd.Close()
}
```

参数说明

```
type AIEnhanceImageOptions struct {
    DetectUrl    string `url:"detect-url,omitempty"`
    Denoise      int    `url:"denoise,omitempty"`
    Sharpen      int    `url:"sharpen,omitempty"`
    IgnoreError int    `url:"ignore-error,omitempty"`
}
```

参数名称	参数描述	类型	是否必填
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 name，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 name。	string	否
Denoise	去噪强度值，取值范围为 0 - 5 之间的整数，值为 0 时不进行去噪操作，默认值为 3。	int	否
Sharpen	锐化强度值，取值范围为 0 - 5 之间的整数，值为 0 时不进行锐化操作，默认值为 3。	int	否
IgnoreError	当此参数为1时，针对文件过大等导致处理失败的场景，会直接返回原图而不报错。	int	否

结果说明

处理后的图片直接返回至响应体。

图像超分

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于图像超分的相关 API 概览以及 SDK 示例代码。

API	操作描述
图像超分	腾讯云数据万象通过 AISuperResolution 接口对图像进行超分辨率处理，当前默认超分为宽高的2倍。

图像超分

腾讯云数据万象通过 AISuperResolution 接口对图像进行超分辨率处理，当前默认超分为宽高的2倍。

方法原型

```
func (s *CIService) GetAISuperResolutionV2(ctx context.Context, name string,
opt *AISuperResolutionOptions) (*Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        }
})
obj := "pic/heibai.jpeg"
localPath := "test.jpeg"
opt := &cos.AISuperResolutionOptions{}
resp, err := c.CI.GetAISuperResolutionV2(context.Background(), obj, opt)
log_status(err)
if err == nil {
    fd, _ := os.OpenFile(localPath, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0660)
    io.Copy(fd, resp.Body)
    fd.Close()
}
```

参数说明

```
type AISuperResolutionOptions struct {
    DetectUrl string `url:"detect-url,omitempty"`
}
```

参数名称	参数描述	类型	是否必填
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 name，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 name。	string	否

结果说明

处理后的图片直接返回至响应体。

图像智能裁剪

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于图像智能裁剪的相关 API 概览以及 SDK 示例代码。

API	操作描述
图像智能裁剪	腾讯云数据万象通过 AIImageCrop 接口对图像进行智能裁剪，支持持久化、云上处理及下载时处理。

图像智能裁剪

腾讯云数据万象通过 AIImageCrop 接口对图像进行智能裁剪，支持持久化、云上处理及下载时处理。

方法原型

```
func (s *CIService) GetAIImageCrop(ctx context.Context, name string, opt
*AIImageCropOptions) (*Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        }
})
obj := "pic/walk.jpeg"
localPath := "test.jpeg"
opt := &cos.AIImageCropOptions{
    Width:  128,
    Height: 96,
}
resp, err := c.CI.GetAIImageCrop(context.Background(), obj, opt)
log_status(err)
if err == nil {
    fd, _ := os.OpenFile(localPath, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0660)
    io.Copy(fd, resp.Body)
    fd.Close()
```

```
}
```

参数说明

```
type AIImageCropOptions struct {
    DetectUrl   string `url:"detect-url,omitempty"`
    Width       int    `url:"width,omitempty"`
    Height      int    `url:"height,omitempty"`
    Fixed       int    `url:"fixed,omitempty"`
    IgnoreError int    `url:"ignore-error,omitempty"`
}
```

参数名称	参数描述	类型	是否必填
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 name ，填写了 detect-url 时，后台会处理 detect-url 链接。	string	否
Width	需要裁剪区域的宽度，与 height 共同组成所需裁剪的图片宽高比例；输入数字请大于0、小于图片宽度的像素值	int	是
Height	需要裁剪区域的高度，与 width 共同组成所需裁剪的图片宽高比例；输入数字请大于0、小于图片高度的像素值；width : height 建议取值在[1, 2.5]之间，超过这个范围可能会影响效果	int	是
Fixed	是否严格按照 width 和 height 的值进行输出。取值为0时，宽高比例（width : height）会简化为最简分数，即如果 width 输入10、height 输入20，会简化为1：2；取值为1时，输出图片的宽度等于 width，高度等于 height；默认值为0	int	否
IgnoreError	当此参数为1时，针对文件过大等导致处理失败的场景，会直接返回原图而不报错	int	否

结果说明

处理后的图片直接返回至响应体。

图像修复

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于图像修复的相关 API 概览以及 SDK 示例代码。

API	操作描述
图像修复	腾讯云数据万象通过 ImageRepair 接　能检测并擦除图片中常见的标志,并对擦除部分进行智能修复，此功能需携带签名。

图像修复

腾讯云数据万象通过 ImageRepair 接　能检测并擦除图片中常见的标志,并对擦除部分进行智能修复，此功能需携带签名。

方法原型

```
func (s *CIService) GetImageRepair(ctx context.Context, name string, opt
*ImageRepairOptions)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        }
})
obj := "pic/watermark.png"
localPath := "test.jpeg"
opt := &cos.ImageRepairOptions{
    MaskPic: "xxx",
}
resp, err := c.CI.GetImageRepair(context.Background(), obj, opt)
log_status(err)
if err == nil {
    fd, _ := os.OpenFile(localPath, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0660)
    io.Copy(fd, resp.Body)
    fd.Close()
```

```
}
```

参数说明

```
type ImageRepairOptions struct {
    DetectUrl string `url:"detect-url,omitempty"`
    MaskPic   string `url:"MaskPic,omitempty"`
    MaskPoly  string `url:"MaskPoly,omitempty"`
}
```

参数名称	参数描述	类型	是否必填
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 name ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 name。	string	否
MaskPic	遮罩（白色区域为需要去除的水印位置）图片地址，私有图片需携带签名，需要经过 URL 安全的 Base64 编码 。例如，遮罩图片为 `http://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/shuiyin_2.png` ，则该处编码后的字符串为 `aHR0cDovL2V4YW1wbGVidWNrZXQtMTI1MDAwMDAwMC5jb3MuYXAtZ3Vhbmd6aG91Lm15cWNsb3VkLmNvbS9zaHVpeWluXzIucG5n`	string	是
MaskPoly	例如： [[[608, 794], [1024, 794], [1024, 842], [608, 842]],[[1295, 62], [1295, 30], [1597, 32],[1597,64]]] ，顺时针输　多边形的每个点的坐标,每个多边形: [[x1, y1], [x2, y2]...] , 形式为三维矩阵（多个多边形： [多边形1,多边形2] ）或　维矩阵（单个多边形），且需要经过 URL 安全的 Base64 编码。MaskPoly 同时与 MaskPic 填写时，优先采　MaskPic 的值。	string	否

结果说明

该请求 body 内容为待修复图片原图。

图片二维码

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于图片二维码相关的 API 概览以及 SDK 示例代码。

API	说明
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
二维码生成	二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码

二维码识别

功能说明

数据万象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。

方法原型

```
func (s *CIService) GetQRcodeV2(ctx context.Context, name string, cover int,
opt *ObjectGetOptions, id ...string) (*GetQRcodeResultV2, *Response, error)
```

请求示例1：上传时识别

```
opt := &cos.ObjectPutOptions{
    nil,
    &cos.ObjectPutHeaderOptions{
        XOptionHeader: &http.Header{},
    },
}
pic := &cos.PicOperations{
    IsPicInfo: 1,
    Rules: []cos.PicOperationsRules{
        {
            FileId: "format.jpg",
            Rule:   "QRcode/cover/1",
        },
    },
}
opt.XOptionHeader.Add("Pic-Operations", cos.EncodePicOperations(pic))
name := "test.jpg"
```

```
filepath := "./QRcode.jpg"
res, _, err := c.CI.PutFromFile(context.Background(), name, filepath, opt)
```

请求示例2：下载时识别

```
name := "test.jpg"
res, _, err := c.CI.GetQRcodeV2(context.Background(), name, 1, nil)
```

参数说明

参数名称	参数描述	类型	必填
cover	二维码覆盖功能，将对识别出的二维码覆盖上马赛克。取值为0或1。0表示不开启二维码覆盖，1表示开启二维码覆盖，默认值0	int	是

结果说明

下载时识别的结果返回如下

```
type GetQRcodeResultV2 struct {
    CodeStatus  int
    QRcodeInfo  []QRcodeInfo
    ResultImage string
}
type QRcodeInfo struct {
    CodeUrl      string
    CodeLocation *CodeLocation
}
type CodeLocation struct {
    Point []string
}
```

参数名称	参数描述	类型
CodeStatus	二维码识别结果。0表示未识别到二维码，1表示识别到二维码	int
QRcodeInfo	二维码识别结果，可能有多个	Struct
ResultImage	处理后的图片 base64数据，请求参数 cover 为1时返回	string
CodeUrl	二维码的内容。可能识别不到内容	string
CodeLocation	图中识别到的二维码位置坐标	Struct
Point	二维码坐标点（X坐标,Y坐标）	Array

二维码生成

功能说明

数据万象二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码。

方法原型

```
func (s *CIService) GenerateQRcode(ctx context.Context, opt
*GenerateQRcodeOptions) (*GenerateQRcodeResult, *Response, error)

func (s *CIService) GenerateQRcodeToFile(ctx context.Context, filePath string,
opt *GenerateQRcodeOptions) (*GenerateQRcodeResult, *Response, error)
```

请求示例

```
opt := &cos.GenerateQRcodeOptions{
    QRcodeContent: "<https://www.example.com>"
    Mode:          0,
    Width:         200,
}
// Case1 生成二维码
res, _, err := c.CI.GenerateQRcode(context.Background(), opt)

// Case2 生成二维码，并保存成文件
filepath := "example.jpg"
res, _, err := c.CI.GenerateQRcodeToFile(context.Background(), filepath, opt)
```

参数说明

```
type GenerateQRcodeOptions struct {
    QRcodeContent string
    Mode          int
    Width         int
}
```

参数名称	参数描述	类型	是否必填
QRcodeContent	可识别的二维码文本信息	string	是
Mode	生成的二维码类型，可选值：0或1。0为二维码，1为条形码，默认值为0	int	否

Width	指定生成的二维码或条形码的宽度，高度会进行等比压缩	int	是

结果说明

```
type GenerateQRcodeResult struct {
    ResultImage string
}
```

参数名称	参数描述	类型
ResultImage	二维码图片 base64 数据	string

图片标签

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于图片标签的 API 概览以及 SDK 示例代码。

API	说明
图片标签	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

功能说明

获取图片中置信度较高的主题标签。

方法原型

```
func (s *CIService) PicTag(ctx context.Context, key string) (*PicTagResult,
*Response, error)
```

请求示例

```
key := "pic/demo.jpg"
res, _, err := c.CI.PicTag(context.Background(), key)
```

参数说明

参数名称	参数描述
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/pic/pic.jpg` 中，对象键为 pic/pic.jpg

结果说明

```
type PicTagResult struct {
    XMLName xml.Name `xml:"RecognitionResult"`
    Labels  []PicTag `xml:"Labels,omitempty"`
}
```

```
type PicTag struct {
    Confidence int    `xml:"Confidence,omitempty"`
    Name       string `xml:"Name,omitempty"`
}
```

参数名称	参数描述
Labels	标签信息
Confidence	该标签的置信度分数，分数越高则该标签准确度越高
Name	识别出的图片标签

图片质量评估

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于图片质量评估的 API 概览以及 SDK 示例代码。

API	说明
图片质量评估	图片质量评估功能为同步请求方式，您可以通过本接口对图片文件进行检测，从多方面评估，最终给出综合可观的清晰度评分和主观的美观度评分。该接口属于 GET 请求。

图片质量评估

功能说明

评估图片的清晰度和主观的美观度。

方法原型

```
func (s *CIService) ImageQuality(ctx context.Context, obj string)
(*ImageQualityResult, *Response, error)
```

请求示例

```
key := "pic/car.png"
res, _, err := c.CI.ImageQuality(context.Background(), key)
```

参数说明

参数名称	参数描述
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/pic/pic.jpg` 中，对象键为 pic/pic.jpg。

结果说明

```
type ImageQualityResult struct {
    XMLName         xml.Name `xml:"Response"`
    LongImage       bool     `xml:"LongImage,omitempty"`
    BlackAndWhite   bool     `xml:"BlackAndWhite,omitempty"`
    SmallImage      bool     `xml:"SmallImage,omitempty"`
```

```
    BigImage        bool      `xml:"BigImage,omitempty"`
    PureImage       bool      `xml:"PureImage,omitempty"`
    ClarityScore    int       `xml:"ClarityScore,omitempty"`
    AestheticScore  int       `xml:"AestheticScore,omitempty"`
    RequestId       string    `xml:"RequestId,omitempty"`
}
```

参数名称	参数描述
LongImage	取值为 TRUE 或 FALSE，TRUE 为长图，FALSE 为正常图，长图定义为长宽比大于等于3或小于等于1/3的图片。
BlackAndWhite	取值为 TRUE 或 FALSE，TRUE 为黑白图，FALSE 为否。黑白图即灰度图，指红绿蓝三个通道都是以灰度色阶显示的图片，并非视觉上的“黑白图片”。
SmallImage	取值为 TRUE 或 FALSE，TRUE 为小图，FALSE 为否，小图定义为最长边小于179像素的图片。当一张图片被判断为小图时，不建议做推荐和展示，其他字段统一输出为0或 FALSE。
BigImage	取值为 TRUE 或 FALSE，TRUE 为大图，FALSE 为否，定义为最短边大于1000像素的图片。
PureImage	取值为 TRUE 或 FALSE，TRUE 为纯色图或纯文字图，即没有内容或只有简单内容的图片，FALSE 为正常图片。
ClarityScore	综合评分。图像清晰度的得分，对图片的噪声、曝光、模糊、压缩等因素进行综合评估，取值为[0, 100]，值越大，越清晰。一般大于50为较清晰图片，标准可以自行把握。
AestheticScore	综合评分。图像美观度得分，从构图、色彩等多个艺术性维度评价图片，取值为[0, 100]，值越大，越美观。一般大于50为较美观图片，标准可以自行把握。
RequestId	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

图片文字识别（OCR）

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于通用文字识别的 API 概览以及 SDK 示例代码。

API	说明
通用文字识别	通用文字识别功能（Optical Character Recognition，OCR）基于行业前沿的深度学习技术，将图片上的文字内容，智能识别为可编辑的文本，可应用于随手拍扫描、纸质文档电子化、电商广告审核等多种场景，大幅提升信息处理效率。

通用文字识别

功能说明

通用文字识别。

方法原型

```
func (s *CIService) OcrRecognition(ctx context.Context, key string, opt
*OcrRecognitionOptions) (*OcrRecognitionResult, *Response, error)
```

请求示例

```
obj := "pic/ocr.png"
opt := &cos.OcrRecognitionOptions{
    Type:               "general",
    LanguageType:       "zh",
    Isword:             true,
    EnableWordPolygon: true,
}
res, _, err := c.CI.OcrRecognition(context.Background(), obj, opt)
```

参数说明

```
type OcrRecognitionOptions struct {
    Type              string `url:"type,omitempty"`
    LanguageType      string `url:"language-type,omitempty"`
    Ispdf             bool   `url:"ispdf,omitempty"`
    PdfPageNumber     int    `url:"pdf-pagenumber,omitempty"`
    Isword            bool   `url:"isword,omitempty"`
    EnableWordPolygon bool   `url:"enable-word-polygon,omitempty"`
```

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	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/pic/pic.jpg` 中，对象键为 pic/pic.jpg	是
Type	OCR 的识别类型，有效值为 general，accurate，efficient，fast，handwriting。general 表示通用印刷体识别；accurate 表示印刷体高精度版；efficient 表示印刷体精简版；fast 表示印刷体高速版；handwriting 表示手写体识别。默认值为 general。	否
LanguageType	type 值为 general 时有效，表示识别语言类型。支持自动识别语言类型，同时支持自选语言种类，默认中英文混合(zh)，各种语言均支持与英文混合的文字识别。可选值请参见 可识别的语言类型 。	否
Ispdf	type 值为 general、fast 时有效，表示是否开启 PDF 识别，有效值为 true 和 false，默认值为 false，开启后可同时支持图片和 PDF 的识别。	否
PdfPageNumber	type 值为 general、fast 时有效，表示需要识别的 PDF 页面的对应页码，仅支持 PDF 单页识别，当上传文件为 PDF 且 ispdf 参数值为 true 时有效，默认值为1。	否
Isword	type 值为 general、accurate 时有效，表示识别后是否需要返回单字信息，有效值为 true 和 false，默认为 false。	否
EnableWordPolygon	type 值为 handwriting 时有效，表示是否开启单字的四点定位坐标输出，有效值为 true 和 false，默认值为 false。	否

结果说明

```
type OcrRecognitionResult struct {
    XMLName         xml.Name          `xml:"Response"`
    TextDetections []TextDetections `xml:"TextDetections,omitempty"`
    Language        string            `xml:"Language,omitempty"`
    Angel           float64           `xml:"Angel,omitempty"`
    PdfPageSize     int               `xml:"PdfPageSize,omitempty"`
    RequestId       string            `xml:"RequestId,omitempty"`
}

type TextDetections struct {
    DetectedText string         `xml:"DetectedText,omitempty"`
    Confidence   int            `xml:"Confidence,omitempty"`
    Polygon      []Polygon      `xml:"Polygon,omitempty"`
    ItemPolygon  []ItemPolygon `xml:"ItemPolygon,omitempty"`
```

```
    Words         []Words       `xml:"Words,omitempty"`
    WordPolygon   []WordPolygon `xml:"WordPolygon,omitempty"`
}

type Polygon struct {
    X int `xml:"X,omitempty"`
    Y int `xml:"Y,omitempty"`
}

type ItemPolygon struct {
    X      int `xml:"X,omitempty"`
    Y      int `xml:"Y,omitempty"`
    Width  int `xml:"Width,omitempty"`
    Height int `xml:"Height,omitempty"`
}

type Words struct {
    Confidence     int             `xml:"Confidence,omitempty"`
    Character      string          `xml:"Character,omitempty"`
    WordCoordPoint *WordCoordPoint `xml:"WordCoordPoint,omitempty"`
}

type WordCoordPoint struct {
    WordCoordinate []Polygon `xml:"WordCoordinate,omitempty"`
}

type WordPolygon struct {
    LeftTop     *Polygon `xml:"LeftTop,omitempty"`
    RightTop    *Polygon `xml:"RightTop,omitempty"`
    RightBottom *Polygon `xml:"RightBottom,omitempty"`
    LeftBottom  *Polygon `xml:"LeftBottom,omitempty"`
}
```

参数名称	参数描述
TextDetections	检测到的文本信息，包括文本行内容、置信度、文本行坐标以及文本行旋转纠正后的坐标。
Language	检测到的语言类型，目前支持的语言类型可参考入参 language-type 说明。
Angel	图片旋转角度（角度制），文本的水平方向为0°；顺时针为正，逆时针为负。
PdfPageSize	图片为 PDF 时，返回 PDF 的总页数，默认为0。
RequestId	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。
DetectedText	识别出的文本行内容。

Confidence	置信度 0 – 100。
Polygon	文本行坐标，以顶点坐标表示 注意：此字段可能返回 null，表示取不到有效值。
ItemPolygon	文本行在旋转纠正之后的图像中的像素坐标，表示为（左上角 x, 左上角 y，宽 width，高 height）。
Words	识别出来的单字信息包括单字（包括单字 Character 和单字置信度 confidence），支持识别的接口：general、accurate。
WordPolygon	字的坐标数组，以四个顶点坐标表示。 注意：此字段可能返回 null，表示取不到有效值。支持识别的类型：handwriting。
X	横坐标。
Y	纵坐标。
Width	宽 width。
Height	高 height。
Confidence	置信度 0 – 100。
Character	识别出来的单词信息。
WordCoordPoint	单字在原图中的四点坐标，支持识别的接口：general、accurate。
WordCoordinate	单字在原图中的坐标，以四个顶点坐标表示，以左上角为起点，顺时针返回。
LeftTop	左上顶点坐标。
RightTop	右上顶点坐标。
RightBottom	右下顶点坐标。
LeftBottom	左下顶点坐标。

版权所有：腾讯云计算（北京）有限责任公司

实时文字翻译

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于实时文字翻译的相关 API 概览以及 SDK 示例代码。

API	操作描述
实时文字翻译	腾讯云数据万象通过 AutoTranslationBlock 接口对文字块进行翻译，请求时需要携带签名。

实时文字翻译

腾讯云数据万象通过 AutoTranslationBlock 接口对文字块进行翻译，请求时需要携带签名。

方法原型

```
func (s *CIService) GetAutoTranslationBlock(ctx context.Context, opt
*AutoTranslationBlockOptions) (*AutoTranslationBlockResults, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        }
})
opt := &cos.AutoTranslationBlockOptions{
    InputText:  "您好，很高兴认识您！",
    SourceLang: "zh",
    TargetLang: "en",
}
res, _, err := c.CI.GetAutoTranslationBlock(context.Background(), opt)
log_status(err)
fmt.Printf("%+v\n", res.TranslationResult)
```

参数说明

```
type AutoTranslationBlockOptions struct {
```

```
    InputText  string `url:"InputText,omitempty"`
    SourceLang string `url:"SourceLang,omitempty"`
    TargetLang string `url:"TargetLang,omitempty"`
    TextDomain string `url:"TextDomain,omitempty"`
    TextStyle  string `url:"TextStyle,omitempty"`
}
```

参数名称	参数描述	类型	是否必填
InputText	待翻译的文本	string	是
SourceLang	输入语言，如 "zh"	string	是
TargetLang	输出语言，如 "en"	string	是
TextDomain	文本所属业务领域，如: "ecommerce", 缺省值为 general	string	否
TextStyle	文本类型，如: "title", 缺省值为 sentence	string	否

结果说明

调用 GetAutoTranslationBlock 函数，会解析 api 返回的 xml 内容到 AutoTranslationBlockResults 结构，具体返回参数可查看 实时文字翻译 文档。

人脸识别

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于人脸识别的 API 概览以及 SDK 示例代码。

API	操作描述
人脸检测	人脸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人脸位置。
人脸特效	人脸特效支持人脸美颜、人脸性别变换、人脸年龄变化、人像分割的特效功能，适用于社交娱乐、广告营销、互动传播等场景。

人脸检测

功能说明

检测图片中的人脸位置。

方法原型

```
func (s *CIService) DetectFace(ctx context.Context, key string, opt
*DetectFaceOptions) (*DetectFaceResult, *Response, error)
```

请求示例

```
obj := "pic/face.jpeg"
opt := &cos.DetectFaceOptions{
    MaxFaceNum: 2,
}
res, _, err := c.CI.DetectFace(context.Background(), obj, opt)
```

参数说明

```
type DetectFaceOptions struct {
    MaxFaceNum int `url:"max-face-num,omitempty"`
}
```

参数名称	参数描述	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/pic/pic.jpg` 中，对象键为 pic/pic.jpg	是
MaxFaceNum	最多处理的人脸数目。默认值为1（仅检测图片中面积最大的那张人脸），最大值为120。此参数用于控制处理待检测图片中的人脸个数，值越小，处理速度越快。	否

结果说明

```
type DetectFaceResult struct {
    XMLName          xml.Name     `xml:"Response"`
    ImageWidth       int          `xml:"ImageWidth,omitempty"`
    ImageHeight      int          `xml:"ImageHeight,omitempty"`
    FaceModelVersion string       `xml:"FaceModelVersion,omitempty"`
    RequestId        string       `xml:"RequestId,omitempty"`
    FaceInfos        []FaceInfos `xml:"FaceInfos,omitempty"`
}

type FaceInfos struct {
    X      int `xml:"X,omitempty"`
    Y      int `xml:"Y,omitempty"`
    Width  int `xml:"Width,omitempty"`
    Height int `xml:"Height,omitempty"`
}
```

参数名称	参数描述
ImageWidth	请求的图片宽度。
ImageHeight	请求的图片高度。
FaceModelVersion	人脸识别所用的算法模型版本。
RequestId	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。
FaceInfos	人脸信息列表。包含人脸坐标信息。
X	人脸框左上角横坐标。人脸框包含人脸五官位置并在此基础上进行一定的扩展，若人脸框超出图片范围，会导致坐标负值。
Y	人脸框左上角纵坐标。人脸框包含人脸五官位置并在此基础上进行一定的扩展，若人脸框超出图片范围，会导致坐标负值。

Width	人脸框宽度。
Height	人脸框高度。

人脸特效

功能说明

人脸特效支持人脸美颜、人脸性别变换、人脸年龄变化、人像分割的特效功能。

方法原型

```
func (s *CIService) FaceEffect(ctx context.Context, obj string, opt
*FaceEffectOptions) (*FaceEffectResult, *Response, error)
```

请求示例

```
obj := "pic/self.jpeg"
opt := &cos.FaceEffectOptions{
    Type:      "face-segmentation",
    Whitening: 50,
}
res, _, err := c.CI.FaceEffect(context.Background(), obj, opt)
```

参数说明

```
type FaceEffectOptions struct {
    Type         string `url:"type,omitempty"`
    Whitening    int    `url:"whitening,omitempty"`
    Smoothing    int    `url:"smoothing,omitempty"`
    FaceLifting  int    `url:"faceLifting,omitempty"`
    EyeEnlarging int    `url:"eyeEnlarging,omitempty"`
    Gender       int    `url:"gender,omitempty"`
    Age          int    `url:"age,omitempty"`
}
```

参数名称	参数描述	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/pic/pic.jpg`	是

	中，对象键为 pic/pic.jpg	
Type	人脸特效类型。人脸美颜：face-beautify；人脸性别转换：face-gender-transformation；人脸年龄变化：face-age-transformation；人像分割：face-segmentation	是
Whitening	type 为 face-beautify 时生效，美白程度，取值范围[0,100]。0表示不美白，100表示最高程度。默认值30	否
Smoothing	type 为 face-beautify 时生效，磨皮程度，取值范围[0,100]。0表示不磨皮，100表示最高程度。默认值10	否
FaceLifting	type 为 face-beautify 时生效，瘦脸程度，取值范围[0,100]。0表示不瘦脸，100表示最高程度。默认值70	否
EyeEnlarging	type 为 face-beautify 时生效，大眼程度，取值范围[0,100]。0表示不大眼，100表示最高程度。默认值70	否
Gender	type 为 face-gender-transformation 时生效，选择转换方向，0表示男变女，1表示女变男。无默认值。注意：仅对图片中面积最大的人脸进行转换	否
Age	type 为 face-age-transformation 时生效，变化到的人脸年龄，范围为[10,80]，无默认值。注意：仅对图片中面积最大的人脸进行转换	否

结果说明

```
type FaceEffectResult struct {
    XMLName     xml.Name `xml:"Response"`
    ResultImage string   `xml:"ResultImage,omitempty"`
    ResultMask  string   `xml:"ResultMask,omitempty"`
}
```

参数名称	参数描述
ResultImage	处理后的图片 base64 数据
ResultMask	人像分割输出参数，一个通过 Base64 编码的文件，解码后文件由 Float 型浮点数组成。这些浮点数代表原图从左上角开始的每一行的每一个像素点，每一个浮点数的值是原图相应像素点位于人体轮廓内的置信度（0-1）转化的灰度值（0-255）

人体识别

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于人体识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
人体识别	腾讯云数据万象通过 AIBodyRecognition 接口识别并输出画面中人体，输出其位置（矩形框）和置信度。图片人体识别请求包属于 GET 请求，请求时需要携带签名。

人体识别

腾讯云数据万象通过 AIBodyRecognition 接口识别并输出画面中人体，输出其位置（矩形框）和置信度。图片人体识别请求包属于 GET 请求，请求时需要携带签名。

方法原型

```
func (s *CIService) AIBodyRecognition(ctx context.Context, key string, opt
*AIBodyRecognitionOptions) (*AIBodyRecognitionResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        }
})
obj := "pic/body.jpeg"
opt := &cosAIBodyRecognitionOptions{}
res, _, err := client.CI.AIBodyRecognition(context.Background(), obj, opt)
log_status(err)
fmt.Printf("%+v\n", res)
```

参数说明

```
type AIBodyRecognitionOptions struct {
    DetectUrl string `url:"detect-url,omitempty"`
```

```
}
```

参数名称	参数描述	类型	是否必填
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 key ，填写了 detect-url 时，后台会处理 detect-url 链接。	string	否

结果说明

调用 AIBodyRecognition 函数，会解析 api 返回的 xml 内容到 AIBodyRecognitionResult 结构，具体返回参数可查看 人体识别 文档。

宠物识别

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于宠物识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
宠物识别	腾讯云数据万象通过 detect-pet 接口识别并输出画面中宠物，输出其位置（矩形框）和置信度。图片宠物识别请求包属于 GET 请求，请求时需要携带签名。

宠物识别

腾讯云数据万象通过 detect-pet 接口识别并输出画面中宠物，输出其位置（矩形框）和置信度。图片宠物识别请求包属于 GET 请求，请求时需要携带签名。

方法原型

```
func (s *CIService) DetectPet(ctx context.Context, obj string, opt
*PetDetectOption) (*PetDetectResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        }
})
obj := "pic/pet.jpeg"
opt := &cos.PetDetectOption{}
res, _, err := c.CI.DetectPet(context.Background(), obj, opt)
log_status(err)
fmt.Printf("%+v\n", res)
```

参数说明

```
type PetDetectOption struct {
    DetectUrl string `url:"detect-url,omitempty"`
```

```
}
```

参数名称	参数描述	类型	是否必填
DetectUrl	您可以通过填写 `detect-url` 处理任意公网可访问的图片链接。不填写 `detect-url` 时，后台会默认处理 obj ，填写了 `detect-url` 时，后台会处理 `detect-url` 链接，无需再填写 obj。	string	否

结果说明

调用 DetectPet 函数，会解析 api 返回的 xml 内容到 PetDetectResult 结构，具体返回参数可查看 宠物识别 文档。

汽车识别

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于汽车识别的 API 概览以及 SDK 示例代码。

API	说明
车辆车牌检测	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。

车辆车牌检测

功能说明

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。

方法原型

```
func (s *CIService) DetectCar(ctx context.Context, key string)
(*DetectCarResult, *Response, error)
```

请求示例

```
key := "pic/car.png"
res, _, err := c.CI.DetectCar(context.Background(), key)
```

参数说明

参数名称	参数描述
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/pic/pic.jpg` 中，对象键为 pic/pic.jpg。

结果说明

```
type DetectCarResult struct {
    XMLName    xml.Name  `xml:"Response"`
    RequestId string    `xml:"RequestId,omitempty"`
    CarTags    []CarTags `xml:"CarTags,omitempty"`
}
```

```
type CarTags struct {
    Serial       string          `xml:"Serial,omitempty"`
    Brand        string          `xml:"Brand,omitempty"`
    Type         string          `xml:"Type,omitempty"`
    Color        string          `xml:"Color,omitempty"`
    Confidence   int             `xml:"Confidence,omitempty"`
    Year         int             `xml:"Year,omitempty"`
    CarLocation  []CarLocation   `xml:"CarLocation,omitempty"`
    PlateContent []PlateContent `xml:"PlateContent,omitempty"`
}

type CarLocation struct {
    X int `xml:"X,omitempty"`
    Y int `xml:"Y,omitempty"`
}

type PlateContent struct {
    Plate         string         `xml:"Plate,omitempty"`
    Color         string         `xml:"Color,omitempty"`
    Type          string         `xml:"Type,omitempty"`
    PlateLocation *PlateLocation `xml:"PlateLocation,omitempty"`
}

type PlateLocation struct {
    X int `xml:"X,omitempty"`
    Y int `xml:"Y,omitempty"`
}
```

参数名称	参数描述
RequestId	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId
CarTags	车辆属性识别的结果数组，支持返回多个车辆信息
Serial	车系
Brand	车辆品牌
Type	车辆类型
Color	车辆颜色
Confidence	置信度，0 - 100
Year	年份，识别不出年份时返回0

CarLocation	车辆在图片中的坐标信息，可能返回多个坐标点的值
PlateContent	车牌信息，包含车牌号、车牌颜色、车牌位置。支持返回多个车牌
Plate	车牌号信息
Color	车牌的颜色
Type	车牌的种类，例如普通蓝牌
PlateLocation	车牌的位置
X	定位出的车牌左上角、右上角、左下角、右下角的 X 坐标
Y	定位出的车牌左上角、右上角、左下角、右下角的 Y 坐标

版权所有：腾讯云计算（北京）有限责任公司

卡证识别

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于卡证识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
卡证识别	本接口支持中国大陆居民二代身份证正面（暂不支持背面）、驾驶证主页（暂不支持副页）所有字段的自动定位，暂不支持文本识别，用于对特定字段的抹除、屏蔽，以及进一步的文本识别。

卡证识别

本接口支持中国大陆居民二代身份证正面（暂不支持背面）、驾驶证主页（暂不支持副页）所有字段的自动定位，暂不支持文本识别，用于对特定字段的抹除、屏蔽，以及进一步的文本识别。

方法原型

```
func (s *CIService) AILicenseRec(ctx context.Context, obj string, opt
*AILicenseRecOptions) (*AILicenseRecResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        }
})
obj := "pic/carCard.jpeg"
opt := &cos.AILicenseRecOptions{
    CardType: "DriverLicense",
}
res, _, err := c.CI.AILicenseRec(context.Background(), obj, opt)
log_status(err)
fmt.Printf("%+v\n", res)
```

参数说明

```
type AILicenseRecOptions struct {
    DetectUrl string `url:"detect-url,omitempty"`
    CardType  string `url:"CardType,omitempty"`
}
```

参数名称	参数描述	类型	是否必填
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 obj，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 obj。	string	否
CardType	卡证识别类型，默认为 DriverLicense。有效值有： • IDCard：表示身份证。 • DriverLicense：表示驾驶证。	string	是

结果说明

调用 AILicenseRec 函数，会解析 api 返回的 xml 内容到 AILicenseRecResult 结构，具体返回参数可查看 卡证识别 文档。

游戏场景识别

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于游戏场景识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
游戏场景识别	游戏标签功能实现游戏图片场景的识别，返回图片中置信度较高的游戏类别标签。游戏标签识别请求包属于 GET 请求，请求时需要携带签名。

游戏场景识别

游戏标签功能实现游戏图片场景的识别，返回图片中置信度较高的游戏类别标签。游戏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

```
func (s *CIService) AIGameRec(ctx context.Context, obj string, opt
*AIGameRecOptions) (*AIGameRecResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        }
})
obj := "pic/game2.jpeg"
opt := &cos.AIGameRecOptions{}
res, _, err := c.CI.AIGameRec(context.Background(), obj, opt)
log_status(err)
fmt.Printf("%+v\n", res)
```

参数说明

```
type AIGameRecOptions struct {
    DetectUrl string `url:"detect-url,omitempty"`
```

```
}
```

参数名称	参数描述	类型	是否必填
DetectUrl	您可以通过填写 detect-url 对任意公网可访问的图片进行游戏场景识别。不填写 detect-url 时，后台会默认处理 obj；填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 obj。	string	否

结果说明

调用 AIGameRec 函数，会解析 api 返回的 xml 内容到 AIGameRecResult 结构，具体返回参数可查看 游戏场景识别 文档。

人脸核身

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于人脸核身的 API 概览以及 SDK 示例代码。

API	操作描述
身份证识别	支持中国大陆居民二代身份证正反面所有字段的识别，包括姓名、性别、民族、出生日期、住址、公民身份证号、签发机关、有效期限； 具备身份证照片、人像照片的裁剪功能和翻拍、PS、复印件告警功能，以及边框和框内遮挡告警、临时身份证告警和身份证有效期不合法告警等扩展功能。
获取动作顺序	使用动作活体检测模式前，需调用本接口获取动作顺序。
获取数字验证码	使用数字活体检测模式前，需调用本接口获取数字验证码。
活体人脸核身	集成了活体检测和跟权威库进行比对的能力，传入一段视频和姓名、身份证号信息即可进行验证。对录制的自拍视频进行活体检测，从而确认当前用户为真人，可防止照片、视频、静态3D建模等各种不同类型的攻击。检测为真人后，再判断该视频中的人与权威库的证件照是否属于同一个人，实现用户身份信息核实。

身份证识别

功能说明

中国大陆居民二代身份证正反面所有字段的识别。

方法原型

```
func (s *CIService) IdCardOCRWhenCloud(ctx context.Context, key string, query
*IdCardOCROptions) (*IdCardOCRResult, *Response, error) // 云上数据处理
func (s *CIService) IdCardOCRWhenUpload(ctx context.Context, key, filePath
string, query *IdCardOCROptions, header *ObjectPutOptions) (*IdCardOCRResult,
*Response, error) // 上传时处理
```

请求示例1：云上数据处理

```
key := "pic/idcard_1.png"
query := &cos.IdCardOCROptions{
    Config: &cos.IdCardOCROptionsConfig{
        CropPortrait:    true,
        CropIdCard:      true,
```

```
        CopyWarn:        true,
        BorderCheckWarn: true,
        ReshootWarn:     true,
        DetectPsWarn:    true,
        TempIdWarn:      true,
        InvalidDateWarn: true,
        Quality:         true,
        MultiCardDetect: true,
    },
}
res, _, err := c.CI.IdCardOCRWhenCloud(context.Background(), key, query)
```

参数说明

```
type IdCardOCROptions struct {
    CardSide string                  `url:"CardSide,omitempty"`
    Config   *IdCardOCROptionsConfig `url:"Config,omitempty"`
}

type IdCardOCROptionsConfig struct {
    CropIdCard      bool `json:"CropIdCard,omitempty"`
    CropPortrait    bool `json:"CropPortrait,omitempty"`
    CopyWarn        bool `json:"CopyWarn,omitempty"`
    BorderCheckWarn bool `json:"BorderCheckWarn,omitempty"`
    ReshootWarn     bool `json:"ReshootWarn,omitempty"`
    DetectPsWarn    bool `json:"DetectPsWarn,omitempty"`
    TempIdWarn      bool `json:"TempIdWarn,omitempty"`
    InvalidDateWarn bool `json:"InvalidDateWarn,omitempty"`
    Quality         bool `json:"Quality,omitempty"`
    MultiCardDetect bool `json:"MultiCardDetect,omitempty"`
}
```

参数名称	参数描述	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/pic/pic.jpg` 中，对象键为 pic/pic.jpg	是
CardSide	• FRONT：身份证有照片的一面（人像面） • BACK：身份证有国徽的一面（国徽面） 该参数如果不填，将为您自动判断身份证正反面	否
CropIdCard	身份证照片裁剪（去掉证件外多余的边缘、自动矫正拍摄角度）	否

CropPortrait	人像照片裁剪（自动抠取身份证头像区域	否
CopyWarn	复印件告警	否
BorderCheckWarn	边框和框内遮挡告警	否
ReshootWarn	翻拍告警	否
DetectPsWarn	PS 检测告警	否
TempIdWarn	临时身份证告警	否
InvalidDateWarn	身份证有效日期不合法告警	否
Quality	图片质量分数（评价图片的模糊程度）	否
MultiCardDetect	是否开启多卡证检测	否

请求示例2：上传时处理

```
key := "pic/idcard_0.png"
filePath := "./idcard_0.png"
query := &cos.IdCardOCROptions{
    Config: &cos.IdCardOCROptionsConfig{
        CropPortrait:    true,
        CropIdCard:      true,
        CopyWarn:        true,
        BorderCheckWarn: true,
        ReshootWarn:     true,
        DetectPsWarn:    true,
        TempIdWarn:      true,
        InvalidDateWarn: true,
        Quality:         true,
        MultiCardDetect: true,
    },
}

res, _, err := c.CI.IdCardOCRWhenUpload(context.Background(), key, filePath,
query, nil)
```

参数说明

参数名称	参数描述
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/pic/pic.jpg` 中，对象键为 pic/pic.jpg
filePath	本地文件路径
query	身份证识别相关选项
header	对象上传参数，详见 上传对象

结果说明

```
type IdCardOCRResult struct {
    XMLName       xml.Name            `xml:"Response"`
    IdInfo        *IdCardInfo         `xml:"IdInfo,omitempty"`
    AdvancedInfo *IdCardAdvancedInfo `xml:"AdvancedInfo,omitempty"`
}

type IdCardInfo struct {
    Name      string `xml:"Name,omitempty"`
    Sex       string `xml:"Sex,omitempty"`
    Nation    string `xml:"Nation,omitempty"`
    Birth     string `xml:"Birth,omitempty"`
    Address   string `xml:"Address,omitempty"`
    IdNum     string `xml:"IdNum,omitempty"`
    Authority string `xml:"Authority,omitempty"`
    ValidDate string `xml:"ValidDate,omitempty"`
}

type IdCardAdvancedInfo struct {
    IdCard          string   `xml:"IdCard,omitempty"`
    Portrait        string   `xml:"Portrait,omitempty"`
    Quality         string   `xml:"Quality,omitempty"`
    BorderCodeValue string   `xml:"BorderCodeValue,omitempty"`
    WarnInfos       []string `xml:"WarnInfos,omitempty"`
}
```

参数名称	参数描述	
Name	姓名（人像面）	

Sex	性别（人像面）	
Nation	民族（人像面）	
Birth	出生日期（人像面）	
Address	地址（人像面）	
IdNum	身份证号（人像面）	
Authority	发证机关（国徽面）	
ValidDate	证件有效期（国徽面）	
IdCard	裁剪后身份证照片的 Base64 编码，设置 Config.CropIdCard 为 true 时返回	
Portrait	身份证头像照片的 Base64 编码，设置 Config.CropPortrait 为 true 时返回	
Quality	图片质量分数，设置 Config.Quality 为 true 时返回（取值范围：0~100，分数越低越模糊，建议阈值≥50）	
BorderCodeValue	身份证边框不完整告警阈值分数，设置 Config.BorderCheckWarn 为 true 时返回（取值范围：0~100，分数越低边框遮挡可能性越低，建议阈值≥50）	
WarnInfos	告警信息，Code 告警码列表和释义： • 9100 身份证有效日期不合法告警 • 9101 身份证边框不完整告警 • 9102 身份证复印件告警 • 9103 身份证翻拍告警 • 9104 临时身份证告警 • 9105 身份证框内遮挡告警 • 9106 身份证 PS 告警 可能存在多个 WarnInfos	

获取动作顺序

功能说明

获取动作顺序。

方法原型

```
func (s *CIService) GetActionSequence(ctx context.Context)
(*GetActionSequenceResult, *Response, error)
```

请求示例

```
res, _, err := c.CI.GetActionSequence(context.Background())
```

参数说明

无

结果说明

```
type GetActionSequenceResult struct {
    XMLName        xml.Name `xml:"Response"`
    ActionSequence string   `xml:"ActionSequence,omitempty"`
}
```

参数名称	参数描述
ActionSequence	动作顺序（2，1 or 1，2）。1代表张嘴，2代表闭眼

获取数字验证码

功能说明

获取数字验证码。

方法原型

```
func (s *CIService) GetLiveCode(ctx context.Context) (*GetLiveCodeResult,
*Response, error)
```

请求示例

```
res, _, err := c.CI.GetLiveCode(context.Background())
```

参数说明

无

结果说明

```
type GetLiveCodeResult struct {
    XMLName  xml.Name `xml:"Response"`
    LiveCode string   `xml:"LiveCode,omitempty"`
}
```

参数名称	参数描述
LiveCode	数字验证码，如：1234

活体人脸核身

功能说明

对录制的自拍视频进行活体检测。

方法原型

```
func (s *CIService) LivenessRecognitionWhenCloud(ctx context.Context, key
string, query *LivenessRecognitionOptions) (*LivenessRecognitionResult,
*Response, error) // 云上数据处理
func (s *CIService) LivenessRecognitionWhenUpload(ctx context.Context, key,
filePath string, query *LivenessRecognitionOptions, header *ObjectPutOptions)
(*LivenessRecognitionResult, *Response, error) // 上传时处理
```

请求示例1：云上数据处理

```
key := "pic/self.mp4"
query := &cos.LivenessRecognitionOptions{
    IdCard:       "111222xxxxxxxxxxxx",
    Name:         "张三",
    LivenessType: "SILENT",
    BestFrameNum: 2,
}
res, _, err := c.CI.LivenessRecognitionWhenCloud(context.Background(), key,
query)
```

参数说明

```
type LivenessRecognitionOptions struct {
    IdCard       string `url:"IdCard,omitempty"`
    Name         string `url:"Name,omitempty"`
    LivenessType string `url:"LivenessType,omitempty"`
    ValidateData string `url:"ValidateData,omitempty"`
    BestFrameNum int    `url:"BestFrameNum,omitempty"`
}
```

参数名称	参数描述	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/pic/pic.jpg` 中，对象键为 pic/pic.jpg	是
IdCard	身份证号	是
Name	姓名。中文请使用 UTF-8编码	是
LivenessType	• 活体检测类型，取值：LIP/ACTION/SILENT • LIP 为数字模式，ACTION 为动作模式，SILENT 为静默模式，三种模式选择一种传入	是
ValidateData	• 数字模式传参：数字验证码（1234），需先调用接口获取数字验证码 • 动作模式传参：传动作顺序（2，1 or 1，2），需先调用接口获取动作顺序 • 静默模式传参：空	是。当 LivenessType 填的是 SILENT 时，该参数为可选项
BestFrameNum	需要返回多张最佳截图，取值范围1 - 10，不设置默认返回一张最佳截图	否

请求示例2：上传时处理

```
key := "pic/self.mp4"
filePath := "./self.mp4"
query := &cos.LivenessRecognitionOptions{
    IdCard:        "111222xxxxxxxxxxxx",
    Name:          "张三",
    LivenessType: "SILENT",
    BestFrameNum: 2,
}
res, _, err := c.CI.LivenessRecognitionWhenUpload(context.Background(), key,
filePath, query, nil)
```

参数说明

参数名称	参数描述
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/pic/pic.jpg` 中，对象键为 pic/pic.jpg
filePath	本地文件路径

query	活体人脸检测相关选项
header	对象上传参数，详见 上传对象

结果说明

```
type LivenessRecognitionResult struct {
    XMLName          xml.Name `xml:"Response"`
    BestFrameBase64  string   `xml:"BestFrameBase64,omitempty"`
    Sim              float64  `xml:"Sim,omitempty"`
    BestFrameList    []string `xml:"BestFrameList,omitempty"`
}
```

参数名称	参数描述
BestFrameBase64	验证通过后的视频最佳截图照片，照片为 BASE64 编码后的值，JPG 格式 注意：此字段可能返回 null，表示取不到有效值
Sim	• 相似度，取值范围 [0.00，100.00]。推荐相似度大于等于70时可判断为同一人，可根据具体场景自行调整阈值（阈值70的误通过率为千分之一，阈值80的误通过率是万分之一） • 检测成功返回
BestFrameList	最佳截图列表，仅在配置了返回多张最佳截图时返回 注意：此字段可能返回 null，表示取不到有效值

商品抠图

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于商品抠图的相关 API 概览以及 SDK 示例代码。

API	操作描述
商品抠图	腾讯云数据万象通过 GoodsMatting 接口检测图片中的商品信息，生成只包含商品信息的图片，支持持久化、云上处理及下载时处理。

方式一：下载时处理

方法原型

```
func (s *CIService) GoodsMattingWithOpt(ctx context.Context, key string, opt
*GoodsMattingptions) (*Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        },
})
key := "pic/cup.jpeg"
localPath := "cup.jpeg"
opt := &cos.GoodsMattingptions{
    CenterLayout:  "1",
    PaddingLayout: "500x500",
}
resp, err := c.CI.GoodsMattingWithOpt(context.Background(), key, opt)
log_status(err)
fd, _ := os.OpenFile(localPath, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0660)
io.Copy(fd, resp.Body)
fd.Close()
```

参数说明

```
type GoodsMattingptions struct {
    CenterLayout  string `url:"center-layout,omitempty"`
    PaddingLayout string `url:"padding-layout,omitempty"`
    DetectUrl     string `url:"detect-url,omitempty"`
}
```

参数名称	参数描述	类型	是否必填
CenterLayout	抠图主体居中显示； 值为1时居中显示，值为0时不作处理，默认为0	string	否
PaddingLayout	• 将处理后的图片四边进行留白，形式为 padding-layout=<dx>x<dy>，左右两边各进行 dx 像素的留白，上下两边各进行 dy 像素的留白，例如：padding-layout=20x10 • 默认不进行留白操作，dx、dy最大值为1000像素。	string	否
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 key。	string	否

结果说明

处理后的图片直接返回至响应体。

方式二：上传时处理

方法原型

```
func (s *CIService) PutFromFile(ctx context.Context, name string, filePath
string, opt *ObjectPutOptions) (*ImageProcessResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        },
```

```
})

opt := &cos.ObjectPutOptions{
    ACLHeaderOptions: nil,
    ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
        XOptionHeader: &http.Header{},
    },
}
pic := &cos.PicOperations{
    IsPicInfo: 1,
    Rules: []cos.PicOperationsRules{
        {
            FileId: "pic/out.jpeg",
            Rule:   "ci-process=GoodsMatting&center-layout=1&padding-
layout=500x500",
        },
    },
}
opt.XOptionHeader.Add("Pic-Operations", cos.EncodePicOperations(pic))
name := "pic/test/cup.jpeg"
local_filename := "./cup.jpeg"
res, _, err := c.CI.PutFromFile(context.Background(), name, local_filename,
opt)
log_status(err)
fmt.Printf("%+v\n", res)
fmt.Printf("%+v\n", res.OriginalInfo)
fmt.Printf("%+v\n", res.ProcessResults)
```

方式三：云上数据处理

方法原型

```
func (s *CIService) ImageProcess(ctx context.Context, name string, opt
*ImageProcessOptions) (*ImageProcessResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
```

```
        },
})

pic := &cos.PicOperations{
    IsPicInfo: 1,
    Rules: []cos.PicOperationsRules{
        {
            FileId: "pic/cup_out.jpeg",
            Rule:   "ci-process=GoodsMatting&center-layout=1&padding-
layout=500x500",
        },
    },
}

key := "pic/cup.jpeg"
res, _, err := c.CI.ImageProcess(context.Background(), key, pic)
log_status(err)
fmt.Printf("%+v\n", res)
```

通用抠图

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于通用抠图的相关 API 概览以及 SDK 示例代码。

API	操作描述
通用抠图	数据万象通用抠图 AIPicMatting 接口检测图片中的主体信息，自动分割图片背景，生成只包含主体信息的图片，支持持久化、云上处理及下载时处理。

方式一：下载时处理

方法原型

```
func (s *CIService) AIPicMatting(ctx context.Context, ObjectKey string, opt
*AIPicMattingOptions) (*Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        },
})
key := "pic/cup.jpeg"
localPath := "cup.jpeg"
opt := &cos.AIPicMattingOptions{
    CenterLayout:  1,
    PaddingLayout: "500x500",
}
resp, err := c.CI.AIPicMatting(context.Background(), key, opt)
log_status(err)
fd, _ := os.OpenFile(localPath, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0660)
io.Copy(fd, resp.Body)
fd.Close()
```

参数说明

```
// https://github.com/tencentyun/cos-go-sdk-v5/blob/master/ci.go#L2585
type AIPicMattingOptions struct {
    CenterLayout  string `url:"center-layout,omitempty"`
    PaddingLayout string `url:"padding-layout,omitempty"`
    DetectUrl     string `url:"detect-url,omitempty"`
}
```

参数名称	参数描述	类型	是否必填
CenterLayout	抠图商品居中显示； 值为1时居中显示，值为0时不作处理，默认为0	Integer	否
PaddingLayout	• 将处理后的图片四边进行留白，形式为 padding-layout=<dx>x<dy>，左右两边各进行 dx 像素的留白，上下两边各进行 dy 像素的留白，例如：padding-layout=20x10 • 默认不进行留白操作，dx、dy最大值为1000像素。	string	否
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 key。	string	否

结果说明

处理后的图片直接返回至响应体。

方式二：上传时处理

方法原型

```
func (s *CIService) PutFromFile(ctx context.Context, name string, filePath
string, opt *ObjectPutOptions) (*ImageProcessResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        },
})
```

```
opt := &cos.ObjectPutOptions{
    ACLHeaderOptions: nil,
    ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
        XOptionHeader: &http.Header{},
    },
}
pic := &cos.PicOperations{
    IsPicInfo: 1,
    Rules: []cos.PicOperationsRules{
        {
            FileId: "pic/out.jpeg",
            Rule:   "ci-process=AIPicMatting&center-layout=1&padding-
layout=500x500",
        },
    },
}
opt.XOptionHeader.Add("Pic-Operations", cos.EncodePicOperations(pic))
name := "pic/test/cup.jpeg"
local_filename := "./cup.jpeg"
res, _, err := c.CI.PutFromFile(context.Background(), name, local_filename,
opt)
log_status(err)
fmt.Printf("%+v\n", res)
fmt.Printf("%+v\n", res.OriginalInfo)
fmt.Printf("%+v\n", res.ProcessResults)
```

方式三：云上数据处理

方法原型

```
func (s *CIService) ImageProcess(ctx context.Context, name string, opt
*ImageProcessOptions) (*ImageProcessResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        },
```

```
})

pic := &cos.PicOperations{
    IsPicInfo: 1,
    Rules: []cos.PicOperationsRules{
        {
            FileId: "pic/cup_out.jpeg",
            Rule:   "ci-process=AIPicMatting&center-layout=1&padding-
layout=500x500",
        },
    },
}

key := "pic/cup.jpeg"
res, _, err := c.CI.ImageProcess(context.Background(), key, pic)
log_status(err)
fmt.Printf("%+v\n", res)
```

人像抠图

最近更新时间：2024-12-27 16:37:04

简介

本文档提供关于人像抠图的相关 API 概览以及 SDK 示例代码。

API	操作描述
人像抠图	腾讯云数据万象通过 AIPortraitMatting 接口检测图片中的人像主体信息，智能分割图像背景，生成只包含人像主体信息的图片，支持持久化、云上处理及下载时处理。

方式一：下载时处理

方法原型

```
func (s *CIService) AIPortraitMatting(ctx context.Context, ObjectKey string,
opt *AIPortraitMattingOptions) (*Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        },
})

key := "pic/cup.jpeg"
localPath := "cup.jpeg"
opt := &cos.AIPortraitMattingOptions{
    CenterLayout:  1,
    PaddingLayout: "500x500",
}
resp, err := c.CI.AIPortraitMatting(context.Background(), key, opt)
log_status(err)
fd, _ := os.OpenFile(localPath, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0660)
io.Copy(fd, resp.Body)
fd.Close()
```

参数说明

```
type AIPortraitMattingOptions struct {
    DetectUrl      string      `url:"detect-url, omitempty" json:"-"`      // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey。 detect-url 示例：http://www.example.com/abc.jpg，需要进行 UrlEncode，处理后为http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg。
    CenterLayout   int         `url:"center-layout, omitempty" json:"-"`   // 抠图主体居中显示；值为1时居中显示，值为0不做处理，默认为0
    PaddingLayout  string      `url:"padding-layout, omitempty" json:"-"` // 将处理后的图片四边进行留白，形式为 padding-layout=x，左右两边各进行 dx 像素的留白，上下两边各进行 dy 像素的留白，例如：padding-layout=20x10默认不进行留白操作，dx、dy最大值为1000像素。
    OptHeaders     *OptHeaders `header:"-, omitempty" url:"-" json:"-" xml:"-"`
}
```

参数名称	参数描述	类型	是否必填
CenterLayout	抠图主体居中显示； 值为1时居中显示，值为0时不作处理，默认为0	Integer	否
PaddingLayout	• 将处理后的图片四边进行留白，形式为 padding-layout=<dx>x<dy>，左右两边各进行 dx 像素的留白，上下两边各进行 dy 像素的留白，例如：padding-layout=20x10 • 默认不进行留白操作，dx、dy最大值为1000像素。	string	否
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 key。	string	否

结果说明

处理后的图片直接返回至响应体。

方式二：上传时处理

方法原型

```
func (s *CIService) PutFromFile(ctx context.Context, name string, filePath string, opt *ObjectPutOptions) (*ImageProcessResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
```

```
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        },
})

opt := &cos.ObjectPutOptions{
    ACLHeaderOptions: nil,
    ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
        XOptionHeader: &http.Header{},
    },
}
pic := &cos.PicOperations{
    IsPicInfo: 1,
    Rules: []cos.PicOperationsRules{
        {
            FileId: "pic/out.jpeg",
            Rule:   "ci-process=AIPortraitMatting&center-layout=1&padding-
layout=500x500",
        },
    },
}
opt.XOptionHeader.Add("Pic-Operations", cos.EncodePicOperations(pic))
name := "pic/test/cup.jpeg"
local_filename := "./cup.jpeg"
res, _, err := c.CI.PutFromFile(context.Background(), name, local_filename,
opt)
log_status(err)
fmt.Printf("%+v\n", res)
fmt.Printf("%+v\n", res.OriginalInfo)
fmt.Printf("%+v\n", res.ProcessResults)
```

方式三：云上数据处理

方法原型

```
func (s *CIService) ImageProcess(ctx context.Context, name string, opt
*ImageProcessOptions) (*ImageProcessResult, *Response, error)
```

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
bu, _ := url.Parse("https://examplebucket-
1250000000.cos.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu}
c := cos.NewClient(b, &http.Client{
        Transport: &cos.AuthorizationTransport{
            SecretID:  os.Getenv("SECRETID"),
            SecretKey: os.Getenv("SECRETKEY"),
        },
})

pic := &cos.PicOperations{
    IsPicInfo: 1,
    Rules: []cos.PicOperationsRules{
        {
            FileId: "pic/cup_out.jpeg",
            Rule:   "ci-process=AIPortraitMatting&center-layout=1&padding-
layout=500x500",
        },
    },
}

key := "pic/cup.jpeg"
res, _, err := c.CI.ImageProcess(context.Background(), key, pic)
log_status(err)
fmt.Printf("%+v\n", res)
```

视频目标检测
创建任务

最近更新时间：2024-12-27 16:37:05

简介

本文档提供关于提交视频目标检测任务的相关 API 概览以及 SDK 示例代码。

API	操作描述
提交任务	提交一个视频目标检测任务。

提交任务

腾讯云数据万象通过 CreateJob 接口创建一个视频目标检测任务。

方法原型

```
func (s *CIService) CreateJob(ctx context.Context, opt *CreateJobsOptions)
(*CreateJobsResult, *Response, error)
```

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")
cu, _ := url.Parse("https://examplebucket-
1250000000.ci.COS_REGION.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"),
        },
})

opt := &cos.CreateJobsOptions{
    Tag: "VideoTargetRec",
    Input: &cos.JobInput{
        Object: "input/test.mp4",
    },
    Operation: &cos.MediaProcessJobOperation{
        VideoTargetRec: &cos.VideoTargetRec{
            Body:         "false",
```

```
            Pet:         "false",
            Car:         "false",
            Face:        "true",
            Plate:       "true",
            ProcessType: "Mosaic",
            TransTpl: &cos.VTRTranscode{
                Container: &cos.Container{
                    Format: "mp4",
                },
                Video: &cos.VTRVideo{
                    Codec: "h.264",
                    Width: "1280",
                    Fps:   "30",
                },
            },
        },
        Output: &cos.JobOutput{
            Region: "ap-beijing",
            Bucket: "test-1234567890",
            Object: "output/out.mp4",
        },
        UserData: "This is my VideoTargetRec job",
        JobLevel: 1,
    },
}
resp, err := c.CI.CreateJob(context.Background(), opt)
log_status(err)
fmt.Printf("%+v\n", resp.JobsDetail)
```

参数说明

```
type CreateJobsOptions CreateMediaJobsOptions

type CreateMediaJobsOptions struct {
    XMLName           xml.Name                    `xml:"Request"`
    Tag               string                      `xml:"Tag,omitempty"`
    Input             *JobInput                   `xml:"Input,omitempty"`
    Operation         *MediaProcessJobOperation   `xml:"Operation,omitempty"`
    QueueId           string                      `xml:"QueueId,omitempty"`
    QueueType         string                      `xml:"QueueType,omitempty"`
    CallBackFormat    string
`xml:"CallBackFormat,omitempty"`
    CallBackType      string
`xml:"CallBackType,omitempty"`
    CallBack          string                      `xml:"CallBack,omitempty"`
```

```
    CallBackMqConfig *NotifyConfigCallBackMqConfig
`xml:"CallBackMqConfig,omitempty"`
}

type JobInput struct {
    Object     string `xml:"Object,omitempty"`
}

type MediaProcessJobOperation struct {
    TemplateId              string
`xml:"TemplateId,omitempty"`
    VideoTargetRec          *VideoTargetRec
`xml:"VideoTargetRec,omitempty"`
    UserData                string
`xml:"UserData,omitempty"`
    Output                  *JobOutput               `xml:"Output,omitempty"`
    JobLevel                int
`xml:"JobLevel,omitempty"`
    SnapshotPrefix          string
`xml:"SnapshotPrefix,omitempty"`
}

type NotifyConfigCallBackMqConfig struct {
    MqMode    string `xml:"MqMode,omitempty"`
    MqRegion string `xml:"MqRegion,omitempty"`
    MqName    string `xml:"MqName,omitempty"`
}

type VideoTargetRec struct {
    Body        string         `xml:"Body,omitempty"`
    Pet         string         `xml:"Pet,omitempty"`
    Car         string         `xml:"Car,omitempty"`
    Face        string         `xml:"Face,omitempty"`
    Plate       string         `xml:"Plate,omitempty"`
    ProcessType string         `xml:"ProcessType,omitempty"`
    TransTpl    *VTRTranscode `xml:"TransTpl,omitempty"`
}

type VTRTranscode struct {
    Container *Container `xml:"Container,omitempty"`
    Video     *VTRVideo  `xml:"Video,omitempty"`
}

type ClipConfig struct {
    Duration string `xml:"Duration"`
}
```

```
type Container struct {
    Format      string       `xml:"Format,omitempty"`
    ClipConfig *ClipConfig `xml:"ClipConfig,omitempty"`
}

type VTRVideo struct {
    Codec   string `xml:"Codec"`
    Bitrate string `xml:"Bitrate,omitempty"`
    Crf     string `xml:"Crf,omitempty"`
    Width   string `xml:"Width,omitempty"`
    Height  string `xml:"Height,omitempty"`
    Fps     string `xml:"Fps,omitempty"`
}

type JobOutput struct {
    Region          string          `xml:"Region,omitempty"`
    Bucket          string          `xml:"Bucket,omitempty"`
    Object          string          `xml:"Object,omitempty"`
}
```

具体的数据描述如下：

Container 类型 CreateMediaJobsOptions 的具体数据描述如下：

参数名称	描述	类型	是否必选
Tag	创建任务的 Tag：VideoTargetRec	String	是
Input	待操作的媒体信息	Container	是
Operation	操作规则	Container	是
QueueId	任务所在的队列 ID。获取队列 ID，请参见 队列与回调 文档	String	否
QueueType	队列类型	String	否
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	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 Input 的具体数据描述如下：

参数名称	描述	类型	是否必选
Object	媒体文件名	String	是

Container 类型 Operation 的具体数据描述如下：

参数名称	描述	类型	是否必选
TemplateId	视频目标检测模板 ID	String	否
VideoTargetRec	视频目标检测参数，同创建视频目标检测模板接口中的 Request.VideoTargetRec	Container	否
UserData	透传用户信息，可打印的 ASCII 码，长度不超过1024	String	否
Output	结果输出配置，当设置了后置处理时必填	Container	否
JobLevel	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
SnapshotPrefix	截图输出前缀。配置后，截图输出到用户桶，路径为 `"SnapshotPrefix/<jobId>/<随机的截图名>"` ，会额外产生费用。为空时不保存截图	String	否

Container 类型 Output 的具体数据描述如下：

参数名称	描述	类型	是否必选
Region	存储桶的地域	String	否
Bucket	存储结果的存储桶	String	否
Object	输出结果的文件名	String	否

返回结果说明

```
type CreateMediaJobsResult struct {
    XMLName     xml.Name                `xml:"Response"`
    JobsDetail *MediaProcessJobDetail `xml:"JobsDetail,omitempty"`
}
type MediaProcessJobDetail struct {
    Code          string                     `xml:"Code,omitempty"`
    Message       string                     `xml:"Message,omitempty"`
    JobId         string                     `xml:"JobId,omitempty"`
    Tag           string                     `xml:"Tag,omitempty"`
    Progress      string                     `xml:"Progress,omitempty"`
    State         string                     `xml:"State,omitempty"`
    CreationTime string                     `xml:"CreationTime,omitempty"`
    StartTime     string                     `xml:"StartTime,omitempty"`
    EndTime       string                     `xml:"EndTime,omitempty"`
    QueueId       string                     `xml:"QueueId,omitempty"`
    Input         *JobInput                  `xml:"Input,omitempty"`
    Operation     *MediaProcessJobOperation `xml:"Operation,omitempty"`
}
```

具体的数据描述如下：

Container 类型 CreateMediaJobsResult 的具体数据描述如下：

节点名称（关键字）	描述	类型
JobsDetail	任务的详细信息	Container

Container 类型 JobsDetail 的具体数据描述如下：

节点名称（关键字）	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：FileCompress	String
Progress	任务进度	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。获取取队列 ID，请参见 队列与回调 文档	String
Input	同请求中的 Input 节点	Container
Operation	该任务的规则	Container

Container 类型 Operation 的具体数据描述如下：

节点名称（关键字）	描述	类型
TemplateId	任务的模板 ID	String
TemplateName	任务的模板名称，当 TemplateId 存在时返回	String
VideoTargetRec	同请求中的 Request.Operation.VideoTargetRec	Container
VideoTargetRecResult	视频目标检测结果	Container
UserData	透传用户信息	String
JobLevel	任务优先级	String

Container 类型 VideoTargetRecResult 的具体数据描述如下：

节点名称（关键字）	描述	类型
Sensitive	脱敏标记，如输入视频中被检测的每一帧都不出现用户所选的检测的模型信息，则此字段值为 false，大于等于一帧出现检测的模型信息时，此字段都为 true，sensitive 为 false 时不返回后处理视频	String
BodyRecognition	人体识别结果	Container 数组
PetRecognition	宠物识别结果	Container 数组
CarRecognition	车辆识别结果	Container 数组
FaceRecognition	人脸识别结果	Container 数组
LicenseRecognitionResult	车牌识别结果	Container 数组

Container 类型 BodyRecognitionResult 的具体数据描述如下：

节点名称（关键字）	描述	类型
FusionScore	混合置信度，取值范围 [0,100]	String
ResultDetails	识别结果	Container 数组

Container 类型 ResultDetails 的具体数据描述如下：

节点名称（关键字）	描述	类型
Time	截图的时间点，单位为秒	String
Url	截图 URL	String
ResultInfo	识别结果，可能有多个	Container 数组

Container 类型 ResultInfo 的具体数据描述如下：

节点名称（关键字）	描述	类型

版权所有：腾讯云计算（北京）有限责任公司

Name	识别类型	String
Score	识别的置信度，取值范围为 [0-100]。值越高概率越大。	Int
Location	图中识别到的坐标	Container

Container 类型 Location 的具体数据描述如下：

节点名称（关键字）	描述	类型
X	X 坐标	String
Y	Y 坐标	String
Height	(X,Y)坐标距离高度	String
Width	(X,Y)坐标距离长度	String

Container 类型节点 PetRecognitionResult 、CarRecognitionResult 、FaceRecognitionResult、LicenseRecognitionResult 节点的内容与 BodyRecognitionResult 节点相同。

查询任务结果

功能说明

查询一个视频目标检测任务，根据任务 ID 查询任务详情。

方法原型

```
func (s *CIService) DescribeJob(ctx context.Context, jobid string)
(*DescribeJobsResult, *Response, error)
```

请求示例

```
jobid := "<jobid>"
res, _, err := c.CI.DescribeJob(context.Background(), jobid)
```

参数说明

参数名称	描述	类型	是否必选
jobid	ship 任务 ID	String	是

返回结果说明

```
type CreateMediaJobsResult struct {
    XMLName     xml.Name                `xml:"Response"`
    JobsDetail *MediaProcessJobDetail `xml:"JobsDetail,omitempty"`
}
type MediaProcessJobDetail struct {
    Code          string                     `xml:"Code,omitempty"`
    Message       string                     `xml:"Message,omitempty"`
    JobId         string                     `xml:"JobId,omitempty"`
    Tag           string                     `xml:"Tag,omitempty"`
    Progress      string                     `xml:"Progress,omitempty"`
    State         string                     `xml:"State,omitempty"`
    CreationTime string                     `xml:"CreationTime,omitempty"`
    StartTime     string                     `xml:"StartTime,omitempty"`
    EndTime       string                     `xml:"EndTime,omitempty"`
    QueueId       string                     `xml:"QueueId,omitempty"`
    Input         *JobInput                  `xml:"Input,omitempty"`
    Operation     *MediaProcessJobOperation `xml:"Operation,omitempty"`
}
```

返回结果具体的数据描述同创建任务返回结果。

创建模板

最近更新时间：2024-12-27 16:37:05

简介

本文档提供关于创建视频目标检测模板的相关 API 概览以及 SDK 示例代码。

API	操作描述
创建模板	创建一个视频目标检测模板。

创建模板

腾讯云数据万象通过 CreateVideoTargetRecTemplate 接口创建一个视频目标检测模板。

方法原型

```
func (s *CIService) CreateVideoTargetRecTemplate(ctx context.Context, opt
*CreateVideoTargetRecTemplateOptions) (*CreateMediaTemplateResult, *Response,
error)
```

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")
cu, _ := url.Parse("https://examplebucket-
1250000000.ci.COS_REGION.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"),
        },
})

opt := &cos.CreateVideoTargetRecTemplateOptions{
    Tag:  "VideoTargetRec",
    Name: "VideoTargetRec-test",
    VideoTargetRec: &cos.VideoTargetRec{
        Body:        "false",
        Pet:         "false",
        Car:         "false",
        Face:        "false",
        Plate:       "true",
```

```
        ProcessType: "Mosaic",
        TransTpl: &cos.VTRTranscode{
            Container: &cos.Container{
                Format: "mp4",
            },
            Video: &cos.VTRVideo{
                Codec: "h.264",
                Width: "1280",
                Fps:   "30",
            },
        },
    },
}
resp, err := c.CI.CreateVideoTargetRecTemplate(context.Background(), opt)
log_status(err)
fmt.Printf("%+v\n", resp.Template)
```

参数说明

```
type CreateVideoTargetRecTemplateOptions struct {
    XMLName         xml.Name         `xml:"Request"`
    Tag             string           `xml:"Tag,omitempty"`
    Name            string           `xml:"Name,omitempty"`
    VideoTargetRec *VideoTargetRec `xml:"VideoTargetRec,omitempty"
json:"VideoTargetRec,omitempty"`
}

type VideoTargetRec struct {
    Body        string        `xml:"Body,omitempty"`
    Pet         string        `xml:"Pet,omitempty"`
    Car         string        `xml:"Car,omitempty"`
    Face        string        `xml:"Face,omitempty"`
    Plate       string        `xml:"Plate,omitempty"`
    ProcessType string        `xml:"ProcessType,omitempty"`
    TransTpl    *VTRTranscode `xml:"TransTpl,omitempty"`
}

type VTRTranscode struct {
    Container *Container `xml:"Container,omitempty"`
    Video     *VTRVideo  `xml:"Video,omitempty"`
}

type Container struct {
    Format      string      `xml:"Format,omitempty"`
    ClipConfig *ClipConfig `xml:"ClipConfig,omitempty"`
}
```

```
type ClipConfig struct {
    Duration string `xml:"Duration"`
}

type VTRVideo struct {
    Codec   string `xml:"Codec"`
    Bitrate string `xml:"Bitrate,omitempty"`
    Crf     string `xml:"Crf,omitempty"`
    Width   string `xml:"Width,omitempty"`
    Height  string `xml:"Height,omitempty"`
    Fps     string `xml:"Fps,omitempty"`
}
```

具体的数据描述如下：

Container 类型 CreateVideoTargetRecTemplateOptions 的具体数据描述如下：

参数名称	描述	类型	是否必选
Tag	模板类型：VideoTargetRec	String	是
Name	模板名称仅支持中文、英文、数字、_、-和*	String	是
VideoTargetRec	视频目标检测参数	Container	是

Container 类型 VideoTargetRec 的具体数据描述如下：

参数名称	描述	类型	是否必选
Body	是否开启人体检测	string	否
Pet	是否开启宠物检测	string	否
Car	是否开启车辆检测	string	否
Plate	是否开启人脸检测	string	否
Face	是否开启车牌检测	string	否
ProcessType	后处理类型，该参数不为空则标识开启检测后处理，支持马赛克类型，只有 Face、Plate 两种检测支持检测后处	string	否

	理		
TransTpl	识别后处理参数，当 ProcessType 不为空时生效且必填	Container	否

注意：

- Body、Pet、Car、Face、Plate 不能同时为 false。
- 开启检测后处理即 ProcessType 不为空时，Body、Pet、Car 不能为 true。

Container 类型 TransTpl 的具体数据描述如下：

参数名称	描述	类型	是否必选
Container	封装格式	Container	是
Video	视频参数	Container	否

Container 类型 Container 的具体数据描述如下：

参数名称	描述	类型	是否必选
Format	封装格式：支持 mp4，flv，mkv，hls，ts，avi，mov	String	是
ClipConfig	分片配置，当 format 为 hls 时有效	Container	否

Container 类型 ClipConfig 的具体数据描述如下：

参数名称	描述	类型	是否必选
Duration	分片时长，默认 5s	String	否

Container 类型 Video 的具体数据描述如下：

参数名称	描述	类型	是否必选
Codec	编解码格式	string	否
Bitrate	视频码率	string	否
Crf	码率-质量控制因子	string	否

Width	宽	string	否
Height	高	string	否
Fps	帧率	string	否

返回结果说明

```
type CreateMediaTemplateResult struct {
    XMLName    xml.Name  `xml:"Response"`
    RequestId string    `xml:"RequestId,omitempty"`
    Template  *Template `xml:"Template,omitempty"`
}

type Template struct {
    TemplateId        string             `xml:"TemplateId,omitempty"`
    Tag               string             `xml:"Code,omitempty"`
    Name              string             `xml:"Name,omitempty"`
    TransTpl          *Transcode         `xml:"TransTpl,omitempty"`
    CreateTime        string             `xml:"CreateTime,omitempty"`
    UpdateTime        string             `xml:"UpdateTime,omitempty"`
    BucketId          string             `xml:"BucketId,omitempty"`
    Category          string             `xml:"Category,omitempty"`
    VideoTargetRec    *VideoTargetRec    `xml:"VideoTargetRec,omitempty"
json:"VideoTargetRec,omitempty"`
}
```

具体的数据描述如下：

Container 类型 CreateMediaTemplateResult 的具体数据描述如下：

节点名称（关键字）	描述	类型
RequestId	请求的唯一 ID	String
Template	保存模板详情的容器	Container

Container 类型 Template 的具体数据描述如下：

节点名称（关键字）	描述	类型
TemplateId	模板 ID	String
Name	模板名称	String

BucketId	模板所属存储桶	String
Category	模板属性，Custom 或者 Official	String
Tag	模板类型，VideoTargetRec	String
UpdateTime	更新时间	String
CreateTime	创建时间	String
VideoTargetRec	同请求体中的 VideoTargetRec	Container

更新模板

最近更新时间：2024-12-27 16:37:05

简介

本文档提供关于创建视频目标检测模板的相关 API 概览以及 SDK 示例代码。

API	操作描述
更新模板	更新一个视频目标检测模板。

更新模板

腾讯云数据万象通过 UpdateVideoTargetRecTemplate 接口更新一个视频目标检测模板。

方法原型

```
func (s *CIService) UpdateVideoTargetRecTemplate(ctx context.Context, opt
*CreateVideoTargetRecTemplateOptions, templateId string)
(*CreateMediaTemplateResult, *Response, error)
```

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")
cu, _ := url.Parse("https://examplebucket-
1250000000.ci.COS_REGION.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"),
        },
})

opt := &cos.CreateVideoTargetRecTemplateOptions{
    Tag:  "VideoTargetRec",
    Name: "VideoTargetRec-test",
    VideoTargetRec: &cos.VideoTargetRec{
        Body:        "false",
        Pet:         "false",
        Car:         "false",
        Face:        "false",
        Plate:       "true",
```

```
        ProcessType: "Mosaic",
        TransTpl: &cos.VTRTranscode{
            Container: &cos.Container{
                Format: "mp4",
            },
            Video: &cos.VTRVideo{
                Codec: "h.264",
                Width: "1280",
                Fps:   "30",
            },
        },
    },
}
templateId := "t1234567890abcdef1234567890abcdef1"
resp, _, err := c.CI.UpdateVideoTargetRecTemplate(context.Background(), opt,
templateId)
log_status(err)
fmt.Printf("%+v\n", resp.Template)
```

参数说明

```
type CreateVideoTargetRecTemplateOptions struct {
    XMLName         xml.Name         `xml:"Request"`
    Tag             string           `xml:"Tag,omitempty"`
    Name            string           `xml:"Name,omitempty"`
    VideoTargetRec *VideoTargetRec `xml:"VideoTargetRec,omitempty"
json:"VideoTargetRec,omitempty"`
}

type VideoTargetRec struct {
    Body        string        `xml:"Body,omitempty"`
    Pet         string        `xml:"Pet,omitempty"`
    Car         string        `xml:"Car,omitempty"`
    Face        string        `xml:"Face,omitempty"`
    Plate       string        `xml:"Plate,omitempty"`
    ProcessType string        `xml:"ProcessType,omitempty"`
    TransTpl    *VTRTranscode `xml:"TransTpl,omitempty"`
}

type VTRTranscode struct {
    Container *Container `xml:"Container,omitempty"`
    Video     *VTRVideo  `xml:"Video,omitempty"`
}

type Container struct {
    Format     string      `xml:"Format,omitempty"`
```

```
    ClipConfig *ClipConfig `xml:"ClipConfig,omitempty"`
}

type ClipConfig struct {
    Duration string `xml:"Duration"`
}

type VTRVideo struct {
    Codec   string `xml:"Codec"`
    Bitrate string `xml:"Bitrate,omitempty"`
    Crf     string `xml:"Crf,omitempty"`
    Width   string `xml:"Width,omitempty"`
    Height  string `xml:"Height,omitempty"`
    Fps     string `xml:"Fps,omitempty"`
}
```

具体的数据描述如下：

Container 类型 CreateVideoTargetRecTemplateOptions 的具体数据描述如下：

参数名称	描述	类型	是否必选
Tag	模板类型：VideoTargetRec	String	是
Name	模板名称 仅支持中文、英文、数字、_、-和*	String	是
VideoTargetRec	视频目标检测参数	Container	是

Container 类型 VideoTargetRec 的具体数据描述如下：

参数名称	描述	类型	是否必选
Body	是否开启人体检测	string	否
Pet	是否开启宠物检测	string	否
Car	是否开启车辆检测	string	否
Plate	是否开启人脸检测	string	否
Face	是否开启车牌检测	string	否

版权所有：腾讯云计算（北京）有限责任公司

ProcessType	后处理类型，该参数不为空则标识开启检测后处理，支持马赛克类型，只有 Face、Plate 两种检测支持检测后处理	string	否
TransTpl	识别后处理参数，当 ProcessType 不为空时生效且必填	Container	否

注意：

- Body、Pet、Car、Face、Plate 不能同时为 false。
- 开启检测后处理即 ProcessType 不为空时，Body、Pet、Car 不能为 true。

Container 类型 TransTpl 的具体数据描述如下：

参数名称	描述	类型	是否必选
Container	封装格式	Container	是
Video	视频参数	Container	否

Container 类型 Container 的具体数据描述如下：

参数名称	描述	类型	是否必选
Format	封装格式：支持 mp4，flv，mkv，hls，ts，avi，mov	String	是
ClipConfig	分片配置，当 format 为 hls 时有效	Container	否

Container 类型 ClipConfig 的具体数据描述如下：

参数名称	描述	类型	是否必选
Duration	分片时长，默认5s	String	否

Container 类型 Video 的具体数据描述如下：

参数名称	描述	类型	是否必选
Codec	编解码格式	string	否
Bitrate	视频码率	string	否

Crf	码率，质量控制因子	string	否
Width	宽	string	否
Height	高	string	否
Fps	帧率	string	否

返回结果说明

```
type CreateMediaTemplateResult struct {
    XMLName    xml.Name  `xml:"Response"`
    RequestId string    `xml:"RequestId,omitempty"`
    Template  *Template `xml:"Template,omitempty"`
}

type Template struct {
    TemplateId        string             `xml:"TemplateId,omitempty"`
    Tag               string             `xml:"Code,omitempty"`
    Name              string             `xml:"Name,omitempty"`
    TransTpl          *Transcode         `xml:"TransTpl,omitempty"`
    CreateTime        string             `xml:"CreateTime,omitempty"`
    UpdateTime        string             `xml:"UpdateTime,omitempty"`
    BucketId          string             `xml:"BucketId,omitempty"`
    Category          string             `xml:"Category,omitempty"`
    VideoTargetRec    *VideoTargetRec    `xml:"VideoTargetRec,omitempty"
json:"VideoTargetRec,omitempty"`
}
```

具体的数据描述如下：

Container 类型 CreateMediaTemplateResult 的具体数据描述如下：

节点名称（关键字）	描述	类型
RequestId	请求的唯一 ID	String
Template	保存模板详情的容器	Container

Container 类型 Template 的具体数据描述如下：

节点名称（关键字）	描述	类型
TemplateId	模板 ID	String

Name	模板名称	String
BucketId	模板所属存储桶	String
Category	模板属性，Custom 或者 Official	String
Tag	模板类型，VideoTargetRec	String
UpdateTime	更新时间	String
CreateTime	创建时间	String
VideoTargetRec	同请求体中的 VideoTargetRec	Container

COS iOS SDK 快速入门

最近更新时间：2024-12-27 16:37:05

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。
> 如果您仍在使用 JSON 版本 SDK，请 升级到 XML iOS SDK 。

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见 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：

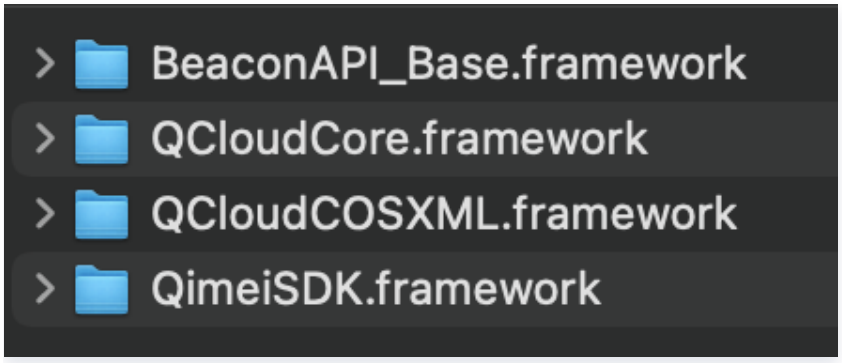
```
pod 'QCloudCOSXML/Transfer'
```

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。



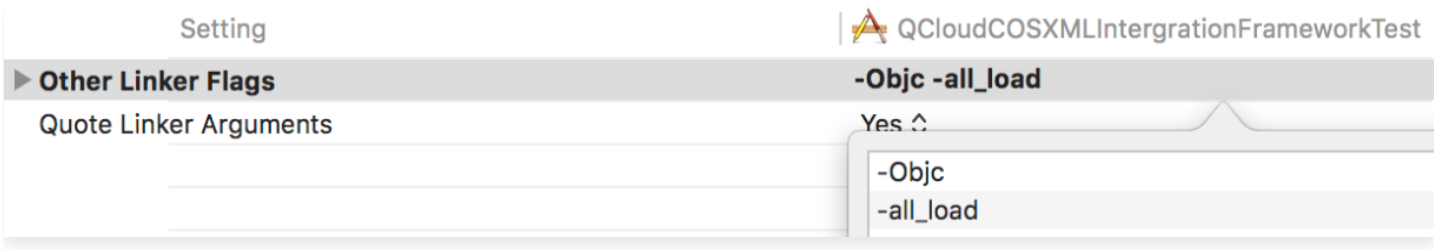
并添加以下依赖库：

- 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
```

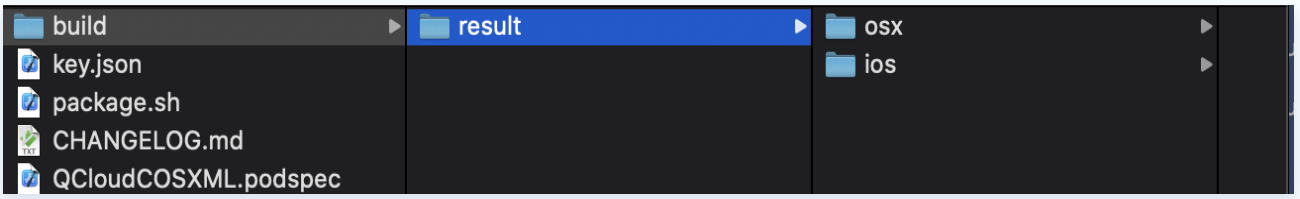


说明

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。

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
registerDefaultCOSTransferMangerWithConfiguration:
        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.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.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://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.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;
        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.cloud.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.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://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(expiredTime)!)

        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,
QCloudCredentailFenceQueueDelegate>

// 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.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
registerDefaultCOSTransferMangerWithConfiguration:
        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
requestCreatorWithContinue:(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.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.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://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.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;
        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.cloud.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.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://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(expiredTime)!)

        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见 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。详细的指引请参见 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。由于该方式存在泄露密钥的风险，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.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
    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.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 OS 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"];

//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: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.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(requ
est);
```

说明

更多完整示例，请前往 GitHub 查看。

快速体验

最近更新时间：2024-12-27 16:37:05

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，如下图所示。



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 ，如下图所示。



查询对象列表

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，如下图所示。



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，如下图所示。



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，如下图所示。


Carrier
4:26 PM
我的存储桶
文件列表
上传
Ffff/
Jjj/
536291（1）.mp4ss
创建时间：2019-08-20 09:55:53
大小:12MB
下载
删除
无更多数据

存储桶操作

最近更新时间：2024-12-27 16:37:05

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.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* 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.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.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
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。

内容识别
图片标签

最近更新时间：2024-12-27 16:37:05

简介

本文档提供关于识别图片标签功能相关的 API 概览以及 SDK 示例代码。

API	说明
图片标签	图片标签功能通过借助持久化处理接口，实现对对象存储（Cloud Object Storage，COS）存量数据的图片标签识别

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 。

下载时识别

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。

示例代码

Objective-C

```
QCloudCIPicRecognitionRequest *request = [QCloudCIPicRecognitionRequest 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 = imageTestBucket.name;
// 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 = @"pic-recognition.png";

[request setFinishBlock:^(QCloudCIPicRecognitionResults *_Nullable result,
NSError *_Nullable error) {
  //result：返回的标签信息
}];

[[QCloudCOSXMLService defaultCOSXML] CIPicRecognition:request];
```

> **说明**
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let put = QCloudCIPicRecognitionRequest();

// 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"
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.setFinish { (outoutObject, error) in
    //outoutObject
};
QCloudCOSXMLService.defaultCOSXML().ciPicRecognition(put);
```

> **说明**
> 更多完整示例，请前往 GitHub 查看。

图片二维码

最近更新时间：2024-12-27 16:37:05

简介

本文档提供关于图片二维码相关的 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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 。

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。

下载时识别

功能说明

二维码识别功能可支持在下载时进行识别。

示例代码

Objective-C

```
QCloudQRCodeRecognitionRequest * request = [QCloudQRCodeRecognitionRequest
new];
request.bucket = @"sample-1250000000";
request.regionName = @"COS_REGIONNAME";
// 万象处理能力，二维码识别固定为 QRcode;是否必传: true;
request.ciProcess = @"QRcode";
// 二维码覆盖功能，将对识别出的二维码覆盖上马赛克。取值为0或1。0表示不开启二维码覆盖，1表示
开启二维码覆盖，默认值0;是否必传: false;
request.cover = 0;
[request setFinishBlock:^(QCloudRecognitionQRcodeResponse * outputObject,
NSError *error) {
  // result: QCloudRecognitionQRcodeResponse 包含所有的响应;
  // 具体查看代码注释或api文档:
https://cloud.tencent.com/document/product/460/37513
}];
```

```
[[QCloudCOSXMLService defaultCOSXML] CIQRCodeRecognition:request];
```

> **说明**
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let request = QCloudQRCodeRecognitionRequest();
request.bucket = "sample-1250000000";
request.regionName = "COS_REGIONNAME";
// 万象处理能力，二维码识别固定为 QRcode;是否必传: true;
request.ciProcess = "QRcode";
// 二维码覆盖功能，将对识别出的二维码覆盖上马赛克。取值为0或1。0表示不开启二维码覆盖，1表示开启二维码覆盖，默认值0;是否必传: false;
request.cover = 0;
request.setFinish { (outoutObject, error) in

};
QCloudCOSXMLService.defaultCOSXML().ciqrCodeRecognition(request);
```

> **说明**
> 更多完整示例，请前往 GitHub 查看。

语音识别
任务接口

最近更新时间：2024-12-27 16:37:05

简介

本文档提供关于提交语音识别任务、查询指定的语音识别任务、批量拉取语音识别任务相关的 API 概览以及 SDK 示例代码。

API	说明
提交语音识别任务	接口用于提交一个语音识别任务。
查询指定的语音识别任务	用于查询指定的语音识别任务。
批量拉取语音识别任务	用于拉取符合条件的语音识别任务。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 。

提交语音识别任务

功能说明

接口用于提交一个语音识别任务。

> ⚠ 注意
> COS iOS SDK 版本需要大于等于 v6.1.3。

示例代码

Objective-C

```
QCloudPostAudioDiscernTaskRequest * request =
[[QCloudPostAudioDiscernTaskRequest alloc]init];

// 存储桶名称，格式为 BucketName-APPID
request.bucket = @"examplebucket-1250000000";
request.regionName = @"regionName";

QCloudPostAudioDiscernTaskInfo* taskInfo = [QCloudPostAudioDiscernTaskInfo
new];
taskInfo.Tag = @"SpeechRecognition";
```

```
// 操作规则
QCloudPostAudioDiscernTaskInfoInput * input =
QCloudPostAudioDiscernTaskInfoInput.new;
input.Object = @"test1";
// 待操作的语音文件
taskInfo.Input = input;
QCloudPostAudioDiscernTaskInfoOperation * op =
[QCloudPostAudioDiscernTaskInfoOperation new];
QCloudPostAudioDiscernTaskInfoOutput * output =
QCloudPostAudioDiscernTaskInfoOutput.new;
output.Region = @"regionName";
output.Bucket = @"examplebucket-1250000000";
output.Object = @"test";
// 结果输出地址
op.Output = output;

QCloudPostAudioDiscernTaskInfoSpeechRecognition * speechRecognition =
[QCloudPostAudioDiscernTaskInfoSpeechRecognition new];
speechRecognition.EngineModelType =@"16k_zh";
speechRecognition.ChannelNum = 1;
speechRecognition.ResTextFormat = 0;
speechRecognition.ConvertNumMode = 0;
// 当 Tag 为 SpeechRecognition 时有效，指定该任务的参数
op.SpeechRecognition = speechRecognition;
// 操作规则
taskInfo.Operation = op;
//  语音识别任务
request.taskInfo = taskInfo;

[request setFinishBlock:^(QCloudPostAudioDiscernTaskResult * _Nullable result,
NSError * _Nullable error) {
    // outputObject 提交审核反馈信息 包含用于查询的 job id，详细字段请查看 API 文档或者
SDK 源码
    // QCloudPostAudioDiscernTaskResult 类;
}];
[[QCloudCOSXMLService defaultCOSXML] PostAudioDiscernTask:request];
```

说明

更多完整示例，请前往 GitHub 查看。

Swift

```
let request = QCloudPostAudioDiscernTaskRequest.init();

// 存储桶名称，格式为 BucketName-APPID
```

```
request.bucket = "examplebucket-1250000000";
request.regionName = "regionName";

let taskInfo = QCloudPostAudioDiscernTaskInfo.init();
taskInfo.tag = "SpeechRecognition";

// 操作规则
let input = QCloudPostAudioDiscernTaskInfoInput.init();
input.object = "test1";
// 待操作的语音文件
taskInfo.input = input;
let op = QCloudPostAudioDiscernTaskInfoOperation.init();
let output = QCloudPostAudioDiscernTaskInfoOutput.init();
output.region = "regionName";
output.bucket = "examplebucket-1250000000";
output.object = "test";
// 结果输出地址
op.output = output;

let speechRecognition =
QCloudPostAudioDiscernTaskInfoSpeechRecognition.init();
speechRecognition.engineModelType = "16k_zh";
speechRecognition.channelNum = 1;
speechRecognition.resTextFormat = 0;
speechRecognition.convertNumMode = 0;
// 当 Tag 为 SpeechRecognition 时有效，指定该任务的参数
op.speechRecognition = speechRecognition;
// 操作规则
taskInfo.operation = op;
// 语音识别任务
request.taskInfo = taskInfo;

[request .setFinish({ outputObject, error in
    // outputObject 提交审核反馈信息 包含用于查询的 job id，详细字段请查看 API 文档或者
SDK 源码
    // QCloudPostAudioDiscernTaskResult 类;
})];
QCloudCOSXMLService.defaultCOSXML().postAudioDiscernTask(request);
```

说明

更多完整示例，请前往 GitHub 查看。

查询指定的语音识别任务

功能说明

用于查询指定的语音识别任务。

> **注意**
> COS iOS SDK 版本需要大于等于 v6.1.3。

示例代码

Objective-C

```
QCloudGetAudioDiscernTaskRequest * request =
[[QCloudGetAudioDiscernTaskRequest alloc]init];

// 存储桶名称，格式为 BucketName-APPID
request.bucket = @"examplebucket-1250000000";

// QCloudPostAudioDiscernTaskRequest接口返回的jobid
request.jobId = @"jobid";

request.regionName = @"regionName";

request.finishBlock = ^(QCloudGetAudioDiscernTaskResult * outputObject,
NSError *error) {
    // outputObject 详细字段请查看 API 文档或者 SDK 源码
    // QCloudGetAudioDiscernTaskResult 类;
};
[[QCloudCOSXMLService defaultCOSXML] GetAudioDiscernTask:request];
```

> **说明**
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let request = QCloudGetAudioDiscernTaskRequest.init();

// 存储桶名称，格式为 BucketName-APPID
request.bucket = "examplebucket-1250000000";

// QCloudPostAudioDiscernTaskRequest 接口返回的 jobid
request.jobId = "jobid";

request.regionName = "regionName";

request.setFinish { outputObject, error in
    // outputObject 详细字段请查看 API 文档或者 SDK 源码
```

```
    // QCloudGetAudioDiscernTaskResult 类;
};
QCloudCOSXMLService.defaultCOSXML().getAudioDiscernTask(request);
```

> **说明**
> 更多完整示例，请前往 GitHub 查看。

批量拉取语音识别任务

功能说明

用于拉取符合条件的语音识别任务。

> **注意**
> COS iOS SDK 版本需要大于等于 v6.1.3。

示例代码

Objective-C

```
QCloudBatchGetAudioDiscernTaskRequest * request =
[[QCloudBatchGetAudioDiscernTaskRequest alloc]init];

// 存储桶名称，格式为 BucketName-APPID
request.bucket = @"examplebucket-1250000000";

request.regionName = @"regionName";

request.states = QCloudTaskStatesSuccess | QCloudTaskStatesCancel;

// 其他更多参数请查阅sdk文档或源码注释

request.finishBlock = ^(QCloudBatchGetAudioDiscernTaskResult * outputObject,
NSError *error) {
    // outputObject 任务结果，详细字段请查看 API 文档或者 SDK 源码
    // QCloudBatchGetAudioDiscernTaskResult 类;
};
[[QCloudCOSXMLService defaultCOSXML] BatchGetAudioDiscernTask:request];
```

> **说明**
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let request = QCloudBatchGetAudioDiscernTaskRequest.init();

// 存储桶名称，格式为 BucketName-APPID
request.bucket = "examplebucket-1250000000";

request.regionName = "regionName";

request.states = QCloudTaskStatesEnum(rawValue:
QCloudTaskStatesEnum.success.rawValue |
QCloudTaskStatesEnum.cancel.rawValue)!;

// 其他更多参数请查阅 SDK 文档或源码注释

request.setFinish { outputObject, error in
    // outputObject 任务结果，详细字段请查看 API 文档或者 SDK 源码
    // QCloudBatchGetAudioDiscernTaskResult 类;
};
QCloudCOSXMLService.defaultCOSXML().batchGetAudioDiscernTask(request);
```

说明

更多完整示例，请前往 GitHub 查看。

语音 bucket 接口

最近更新时间：2024-12-27 16:37:05

简介

本文档提供关于查询语音识别开通状态相关的 API 概览以及 SDK 示例代码。

API	说明
查询语音识别开通状态	接口用于查询存储桶是否已开通语音识别功能。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 。

查询语音识别开通状态

功能说明

接口用于查询存储桶是否已开通语音识别功能。

> ⚠ 注意
> COS iOS SDK 版本需要大于等于 v6.1.3。

示例代码

Objective-C

```
QCloudGetAudioDiscernOpenBucketListRequest * request =
[[QCloudGetAudioDiscernOpenBucketListRequest alloc]init];

// 存储桶名称前缀，前缀搜索
request.bucketName = @"examplebucket-1250000000";

request.regionName = @"regionName";
// 地域信息，以“,”分隔字符串，支持 All、ap-shanghai、ap-beijing
request.regions = @"regions";

request.finishBlock = ^(QCloudGetAudioOpenBucketListResult * outputObject,
NSError *error) {
    // outputObject 详细字段请查看 API 文档或者 SDK 源码
    // QCloudGetAudioOpenBucketListResult 类;
};
[[QCloudCOSXMLService defaultCOSXML] GetAudioDiscernOpenBucketList:request];
```

> **说明**
> 更多完整示例，请前往 GitHub 查看。

Swift

```
let request = QCloudGetAudioDiscernOpenBucketListRequest.init();

// 存储桶名称前缀，前缀搜索
request.bucketName = "examplebucket-1250000000";

request.regionName = "regionName";
// 地域信息，以“,”分隔字符串，支持 All、ap-shanghai、ap-beijing
request.regions = "regions";

request.setFinish { outputObject, error in
    // outputObject 详细字段请查看 API 文档或者 SDK 源码
    // QCloudGetAudioOpenBucketListResult 类;
}
QCloudCOSXMLService.defaultCOSXML().getAudioDiscernOpenBucketList(request);
```

> **说明**
> 更多完整示例，请前往 GitHub 查看。

队列接口

最近更新时间：2024-12-27 16:37:05

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，请参见 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 =
[[QCloudGetAudioDiscernTaskQueueRequest alloc]init];

    // 存储桶名称，格式为 BucketName-APPID
    request.bucket = @"examplebucket-1250000000";

    request.regionName = @"regionName";

    // 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";

    // 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 =
[[QCloudUpdateAudioDiscernTaskQueueRequest 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;

// 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;

// 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。

Java SDK
快速入门

最近更新时间：2024-12-27 16:37:05

简介

数据万象的媒体处理及文档预览接口集成至对象存储服务 XML Java SDK。

下载与安装

相关资源

- 对象存储服务的 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 常见问题请参见：常见问题 。

> **说明：**
> 强烈推荐您及时更新到最新的SDK版本，避免因版本落后而影响您的体验。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。如果您仍在使用 JSON 版本 SDK，请升级到 XML Java SDK 。

环境依赖

- SDK 支持 JDK 1.8及以上版本。
- 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包 com.qcloud.cos.* 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。
- 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>5.6.227</version>
</dependency>
```

> **说明：**
> 依赖坐标可能并非最新版本，请 单击此处 获取最新版本。

- 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

下面为您介绍如何使用 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。数据万象的媒体处理接口集成至对象存储服务 XML Java SDK，您可完成 模板操作 、任务操作 、工作流操作 、队列操作 等相关操作。

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

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，参考示例如下：

```
// 1 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);
// 2 设置 bucket 的区域, COS 地域的简称请参照
https://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);
// 3 生成 cos 客户端。
COSClient client = new COSClient(cred, clientConfig);
```

您也可以使用临时密钥初始化 COSClient，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 = "COS_SECRETID";
String tmpSecretKey = "COS_SECRETKEY";
String sessionToken = "COS_TOKEN";
BasicSessionCredentials cred = new BasicSessionCredentials(tmpSecretId,
tmpSecretKey, sessionToken);
// 2 设置 bucket 的区域, COS 地域的简称请参阅
https://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
参阅源码或者常见问题 Java SDK 部分
Region region = new Region("COS_REGION");
ClientConfig clientConfig = new ClientConfig(region);
// 3 生成 cos 客户端
COSClient client = 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	存储桶所在的区域，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 HTTP 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	int
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

查询开通媒体处理功能的桶列表

查询当前账号下已经开通媒体处理功能的桶列表。

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);
```

创建任务

创建一个媒体处理任务。

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("p9900025e4ec44b5e8225e70a521*****");
//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。

```
MediaJobsRequest request = new MediaJobsRequest();
request.setBucketName("examplebucket-1250000000");
request.setJobId("jae776cb4ec3011eab2cdd3817d4*****");
Boolean response = client.cancelMediaJob(request);
```

查询任务

根据任务 id 查询任务详情。

```
//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

查询队列中的任务列表。

> **注意：**
> 任务记录保留一个月，请及时保存任务记录。建议配置任务回调进行任务结果查询。

请求示例

```
MediaJobsRequest request = new MediaJobsRequest();
request.setBucketName("examplebucket-1250000000");
request.setTag("Transcode");
MediaListJobResponse response = client.describeMediaJobs(request);
List<MediaJobObject> jobsDetail = response.getJobsDetail();
```

关闭客户端

关闭 client，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)
client.shutdown();
```

常见问题

最近更新时间：2024-12-27 16:37:05

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，因此建议通过 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. 确认是否是通过公网访问 COS 或 CI。
   目前同地域 CVM 访问 COS 一般通过内网访问（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 ）。如果是通过公网访问，需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. 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，解决办法可参考步骤1和步骤2。

上传文件请求路径中带有“+”，报错 Code: 403 SignatureDoesNotMatch，如何处理?

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：

- 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COS的文件目录?

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 =
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 地域的简称请参见
https://cloud.tencent.com/document/product/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 或 CI 的用户，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 地域的简称请参见
https://cloud.tencent.com/document/product/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：

```
// 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
SelfDefinedEndpointBuilder();
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。

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 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 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。

如何使用安全的base64编码?

1. 将普通 BASE64 编码结果中的加号（+）替换成连接号（-）；
2. 将编码结果中的正斜线（/）替换成下划线（_）；
3. 将编码结果中的“=”去掉。

```
public static String safeUrlBase64Encode(byte[] data) {
    String encodeBase64 = new String(Base64.getEncoder().encode(data));
    String safeBase64Str = encodeBase64.replace('+', '-');
    safeBase64Str = safeBase64Str.replace('/', '_');
    safeBase64Str = safeBase64Str.replaceAll("=", "");
    return safeBase64Str;
}
```

AI 内容识别
AI通用接口示例

最近更新时间：2024-12-27 16:37:05

简介

本文档提供关于 AI 相关通用接口 API 概览以及 SDK 示例代码。

> **注意：**
> 此 demo 适用接口为：图片上色、图像超分、商品抠图、图像增强、图像智能裁剪、图像修复、人脸智能美颜。该功能需要 COS Java SDK v5.6.193 及以上版本。旧版本 SDK 该接口可能未覆盖或缺少更新参数，使用时建议升级到 最新版本 。

请求示例

下载时处理示例代码

```
public static void getGoodsMatting(COSClient client) {
    //图片所在bucket名称
    String bucketName = "demo-1234567890";
    //图片在bucket中的相对位置，比如根目录下file文件夹中的demo.png路径为file/demo.png
    String key = "car.jpg";
    GetObjectRequest getObj = new GetObjectRequest(bucketName, key);
    //具体参数请参考API 接口不同，对应的ci-process内容和使用的参数有区别
    getObj.putCustomQueryParameter("ci-process", "GoodsMatting");
    getObj.putCustomQueryParameter("center-layout", "1");
    ObjectMetadata object = client.getObject(getObj, new File("demo.jpg"));
}
```

云上数据处理

```
public static void postGoodsMatting(COSClient client) {
    String bucketName = "demo-1234567890";
    String key = "car.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("cat2.jpg");
```

```
    rule1.setRule("ci-process=GoodsMatting&center-layout=1&padding-
layout=20x10");
    ruleList.add(rule1);
    picOperations.setRules(ruleList);
    imageReq.setPicOperations(picOperations);
    CIUploadResult ciUploadResult = client.processImage(imageReq);
    System.out.println(Jackson.toJsonString(ciUploadResult));
}
```

上传时处理

> **注意：**
> 此处上传时处理为简单上传，不支持分片上传，大文件建议使用分片上传后调用云上处理接口。

```
public static void putGoodsMatting(COSClient cosClient) {
    String bucketName = "demo-1234567890";
    String key = "car3.jpg";
    File localFile = new File("car.jpg");
    PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key,
localFile);
    PicOperations picOperations = new PicOperations();
    picOperations.setIsPicInfo(1);
    List<PicOperations.Rule> ruleList = new LinkedList<>();
    PicOperations.Rule rule1 = new PicOperations.Rule();
    rule1.setBucket(bucketName);
    rule1.setFileId(key);
    rule1.setRule("ci-process=GoodsMatting&center-layout=1&padding-
layout=20x10");
    ruleList.add(rule1);
    picOperations.setRules(ruleList);
    putObjectRequest.setPicOperations(picOperations);
    PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
    CIUploadResult ciUploadResult = putObjectResult.getCiUploadResult();
    System.out.println(Jackson.toJsonString(putObjectResult));
}
```

返回结果说明

- 成功： 返回任务详情响应包装类，下载时处理为 ObjectMetadata 对象,云上数据处理和上传时处理为 CIUploadResult 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 。

第三方资源请求示例：

部分接口支持参数 detect-url，以图像超分接口为例。**在创建 cosclient 时，需要关闭 RequestPath 校验。**

```
public class AISuperResolutionDemo {

    public static void main(String[] args) {
        // 1 初始化用户身份信息（secretId, secretKey）。
        COSClient cosClient = getCosClient("sid", "skey", "region");
        // 2 调用要使用的方法。
        getAISuperResolution(cosClient);
        cosClient.shutdown();
    }

    public static COSClient getCosClient(String secretId, String secretKey,
String _region) {
        // 1 初始化用户身份信息（secretId, secretKey）。
        COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
        // 2 设置 bucket 的区域, CI 地域的简称请参照
https://cloud.tencent.com/document/product/436/6224
        // clientConfig 中包含了设置 region, https(默认 https), 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(_region);
        ClientConfig clientConfig = new ClientConfig(region);
        clientConfig.setCheckRequestPath(false);
        // 3 生成 cos 客户端。
        return new COSClient(cred, clientConfig);
    }

    public static void getAISuperResolution(COSClient client) {
        //图片所在bucket名称
        String bucketName = "demo-1234567890";
        //图片在bucket中的相对位置，比如根目录下file文件夹中的demo.png路径为
file/demo.png
        //使用detect-url参数时 这里设置为空字符串  部分接口有限制 key和detect-url不可
同时存在
        String key = "";
        GetObjectRequest getObj = new GetObjectRequest(bucketName, key);
        //具体参数请参考API
        getObj.putCustomQueryParameter("ci-process", "AISuperResolution");
        getObj.putCustomQueryParameter("detect-url", "https://demo-
1234567890.cos.ap-chongqing.myqcloud.com/1.jpg");
        ObjectMetadata object = client.getObject(getObj, new
File("demo.png"));
    }

}
```

版权所有：腾讯云计算（北京）有限责任公司

图片二维码

最近更新时间：2024-12-27 16:37:05

简介

本文档提供关于对象存储图片二维码相关的 API 概览以及 SDK 示例代码。

API	操作描述
二维码生成	指定的文本信息（URL 或文本），生成对应的二维码或条形码
二维码生成	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

二维码生成

功能说明

数据万象二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码。

方法原型

```
String generateQrcode(GenerateQrcodeRequest request);
```

请求示例

```
//1.创建二维码生成请求对象
GenerateQrcodeRequest request = new GenerateQrcodeRequest();
//2.添加请求参数 参数详情请见 API 接口文档
request.setBucketName("examplebucket-1250000000");
request.setQrcodeContent("数据万象");
request.setWidth("400");
request.setMode("0");
String imageBase64 = cosClient.generateQrcode(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
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储	Strin	是

	桶概述	g	
QrcodeContent	可识别的二维码文本信息	String	是
width	指定生成的二维码或条形码的宽度，高度会进行等比压缩	String	是
Mode	生成的二维码类型，可选值：0或1。0为二维码，1为条形码，默认值为0	int	否

返回结果说明

- 成功：成功则返回 String，信息为 base64编码的图片。
- 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 。

二维码识别

对象存储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。

示例代码1：上传时识别二维码

```
// bucket 名需包含 appid
// API 请参见 https://cloud.tencent.com/document/product/436/54070
String bucketName = "examplebucket-1250000000";
String key = "qrcode.png";
File localFile = new File("E://qrcode.png");
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key,
localFile);
PicOperations picOperations = new PicOperations();
picOperations.setIsPicInfo(1);
List<PicOperations.Rule> ruleList = new LinkedList<>();
PicOperations.Rule rule = new PicOperations.Rule();
rule.setBucket(bucketName);
rule.setFileId("qrcode-1.png");
rule.setRule("QRcode/cover/1");
ruleList.add(rule);
picOperations.setRules(ruleList);
putObjectRequest.setPicOperations(picOperations);
try {
    PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
    CIUploadResult ciUploadResult = putObjectResult.getCiUploadResult();
    System.out.println(putObjectResult.getRequestId());
    System.out.println(ciUploadResult.getOriginalInfo().getEtag());
    for(CIObject ciObject:ciUploadResult.getProcessResults().getObjectList())
{
```

```
            System.out.println(ciObject.getLocation());
        }
    } catch (CosServiceException e) {
        e.printStackTrace();
    } catch (CosClientException e) {
        e.printStackTrace();
    }
```

示例代码2：下载时识别二维码

```
GetObjectRequest getObj = new GetObjectRequest(bucketName, key);
// 这里是图片二维码识别参数，具体请参考数据万象 API，这里仅是示例
String cover = "0";
getObj.putCustomQueryParameter("ci-process", "QRcode");
getObj.putCustomQueryParameter("cover", cover);
```

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：

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
codeStatus	二维码识别结果。0表示未识别到二维码，1表示识别到二维码	Integer
QRcodeInfoList	二维码识别结果，可能有多个	List

QRcodeInfo 用于记录二维码识别结果，其主要成员说明如下：

成员名称	描述	类型
codeUrl	二维码的内容可能识别不出	String
codeLocation	图中识别到的二维码位置坐标	CodeLocation

CodeLocation 用于记录图中识别到的二维码位置坐标 ，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
points	二维码坐标点的集合	List

版权所有：腾讯云计算（北京）有限责任公司

图片标签

最近更新时间：2024-12-27 16:37:05

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
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 。

汽车识别

最近更新时间：2024-12-27 16:37:05

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
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 。

回调信息处理

最近更新时间：2024-12-27 16:37:05

简介

本文档提供关于数据万象标准处理回调信息解析的 SDK 示例代码。

XML 回调内容解析

功能说明

将数据万象公共的 XML 回调信息解析为 Java 对象。

使用示例

本示例用于解析回调数据，可通过获取 Unmarshallers 来进行 XML 数据的解析。

> **说明**
> 您可以通过查看 COSClient 的接口获取到需要使用的 XML 解析器，也可以通过查看功能对应的查询接口实现使用的是哪个 Unmarshaller。

以文档预览回调信息为例，对应的查询接口为 describeDocProcessJob。

- COSClient 中的代码片段，查看 invoke 方法中是使用哪个 Unmarshaller。

```
@Override
public DocJobResponse describeDocProcessJob(DocJobRequest request) {
  this.checkCIRequestCommon(request);
  CosHttpRequest<DocJobRequest> httpRequest =
this.createRequest(request.getBucketName(), "/doc_jobs/" +
request.getJobId(), request, HttpMethodName.GET);
  return this.invoke(httpRequest, new
Unmarshallers.DescribeDocJobUnmarshaller());
}
```

- 使用 Unmarshaller 进行解析响应内容。

```
public static void processCINotifyResponse(String response) throws
Exception {
  Unmarshallers.DescribeDocJobUnmarshaller describeDocJobUnmarshaller = new
Unmarshallers.DescribeDocJobUnmarshaller();
  InputStream is = new ByteArrayInputStream(response.getBytes());
  DocJobResponse docJobResponse =
describeDocJobUnmarshaller.unmarshall(is);
}
```

JSON 回调内容解析

功能说明

将数据万象公共的 JSON 回调信息解析为 Java 对象。

使用示例

> **说明**
> - 本质上处理 JSON 回调是将 JSON 转为 XML 后再进行处理。
> - Demo 中使用了 org.json 将 Json 转为 XML，并未在 SDK 的 Pom 中提供该依赖，需要自行导入。

以图片内容审核回调接口为例，对应的查询接口为 describeAuditingImageJob。

- COSClient 中的代码片段，查看 invoke 方法中是使用哪个 Unmarshaller。

```
@Override
public ImageAuditingResponse
describeAuditingImageJob(DescribeImageAuditingRequest imageAuditingRequest)
{
  rejectNull(imageAuditingRequest.getBucketName(),
          "The bucketName parameter must be specified setting the object
tags");
  rejectNull(imageAuditingRequest.getJobId(),
          "The jobId parameter must be specified setting the object tags");
  CosHttpRequest<DescribeImageAuditingRequest> request =
createRequest(imageAuditingRequest.getBucketName(), "/image/auditing/" +
imageAuditingRequest.getJobId(), imageAuditingRequest, HttpMethodName.GET);
  return invoke(request, new
Unmarshallers.ImageAuditingDescribeJobUnmarshaller());
}
```

- 将 JSON 响应转为 XML 后使用 Unmarshaller 进行解析响应内容。

```
//回调 demo
public static void processJsonCINotifyResponse(String jsonResponse) throws
Exception {
  JSONObject response = new JSONObject(jsonResponse);
  JSONObject json = new JSONObject();
  json.put("Response",response);
  String xml = XML.toString(json);
  Unmarshallers.ImageAuditingDescribeJobUnmarshaller imageJobUnmarshaller =
new Unmarshallers.ImageAuditingDescribeJobUnmarshaller();
  InputStream is = new ByteArrayInputStream(xml.getBytes());
  ImageAuditingResponse imageAuditingResponse =
imageJobUnmarshaller.unmarshall(is);
```

```
}
```

上述示例使用的是 org.json.json，SDK 中并没有提供，如需使用请添加如下依赖。

```
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20180130</version>
</dependency>
```

异常处理

最近更新时间：2024-12-27 16:37:05

简介

调用 SDK 请求对象存储（Cloud Object Storage，COS）或数据万象（Cloud Infinite，CI）服务失败时，抛出的异常皆是 RuntimeExcpetion。
目前 SDK 常见的异常有 CosClientException，CosServiceException，CIServiceException 和 IllegalArgumentException。

客户端异常

CosClientException 是由于客户端无法和服务端完成正常的交互而导致的失败。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。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

CosServiceException 和 CIServiceException 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

针对请求可能遇到不同的报错情况，我们为您提供了 CI 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 CI 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**，请您耐心等待几分钟，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。

JavaScript SDK 快速入门

最近更新时间：2025-03-14 19:19:32

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。
> 如果您仍在使用 JSON 版本 SDK，请 升级到 XML JavaScript SDK 。

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。



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：

单击此处 下载最新 cos-js-sdk-v5.min.js。

```
<script src="https://cdn.jsdelivr.net/npm/cos-js-sdk-v5/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');
```

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.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.cloud.tencent.com/cos5/bucket/
// 关于地域的详情见 https://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://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://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。

```
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 SecurityToken
            });
        };
        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.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
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（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
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：

- 实例化时，传入 SecretId、SecretKey，每次需要签名都由实例内部计算。
- 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。
- 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);
});
```

内容识别
服务和队列配置
开通AI内容识别服务

最近更新时间：2024-12-27 16:37:05

简介

开通AI内容识别服务功能是由 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。
本文档提供关于开通AI内容识别服务的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
开通AI内容识别服务	本接口用于开通AI 内容识别（异步）服务并生成队列。

开通AI内容识别服务

功能说明

本接口用于开通AI 内容识别（异步）服务并生成队列。

使用示例

```
function openAIBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'ai_bucket'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
```

```
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
openAIBucket();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Url	请求url，固定值	String	是
Key	固定值：ai_bucket	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Respon se	响应结果，详情请参见 开通AI内容识别服务	Object

相关参考

API 文档

查询 AI 内容识别服务

最近更新时间：2024-12-27 16:37:05

简介

查询AI内容识别服务功能是由 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。
本文档提供关于查询AI内容识别服务的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
查询AI内容识别服务	本接口用于查询已经开通 AI 内容识别（异步）服务的存储桶。

查询AI内容识别服务

功能说明

本接口用于查询已经开通 AI 内容识别（异步）服务的存储桶。

使用示例

```
function getAIBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'ai_bucket'; // 固定值
  const host = `ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;

  cos.request(
    {
      Method: 'GET', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Query: {
        // 地域信息，例如 ap-shanghai、ap-beijing，若查询多个地域以“,”分隔字符串，详情
请参见 地域与域名;是否必传：否
        // regions:  '',
```

```
        // 存储桶名称，以“,”分隔，支持多个存储桶，精确搜索;是否必传：否
        // bucketNames:  '',
        // 存储桶名称前缀，前缀搜索;是否必传：否
        // bucketName:  '',
        // 第几页;是否必传：否
        // pageNumber: 1,
        // 每页个数，大于0且小于等于100的整数;是否必传：否
        // pageSize: 10,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
getAIBucket();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：GET	String	是
Url	请求url，固定值	String	是
Key	固定值：ai_bucket	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
regions	地域信息，例如 ap-shanghai、ap-beijing，若查询多个地域以“,”分隔字符串，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	Integer	否
pageSize	每页个数，大于0且小于等于100的整数	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果 详情请参见 查询 AI 内容识别服务	Object

相关链接

API 文档

关闭AI内容识别服务

最近更新时间：2024-12-27 16:37:05

简介

关闭AI内容识别服务功能是由 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。
本文档提供关于关闭AI内容识别服务的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
关闭AI内容识别服务	本接口用于关闭 AI 内容识别（异步）服务并删除队列。

关闭AI内容识别服务

功能说明

本接口用于关闭AI 内容识别（异步）服务并删除队列。

使用示例

```
function closeAIBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'ai_bucket'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;

  cos.request(
    {
      Method: 'DELETE', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
```

```
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
closeAIBucket();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：DELETE	String	是
Url	请求url，固定值	String	是
Key	固定值：ai_bucket	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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

- Response	响应结果 详情请参见 关闭AI内容识别服务	Object

相关参考

API 文档

查询AI内容识别队列

最近更新时间：2024-12-27 16:37:05

简介

查询AI内容识别队列功能是由 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。
本文档提供关于查询AI内容识别队列的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
查询AI内容识别队列	本接口用于查询 AI 内容识别（异步）队列。

查询AI内容识别队列

功能说明

本接口用于查询AI 内容识别（异步）队列。

使用示例

```
function getAIQueue() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'ai_queue'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;

  cos.request(
    {
      Method: 'GET', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Query: {
        // 队列 ID，以“,”符号分割字符串;是否必传：否
        // queueIds:  "",
```

```
        // Active 表示队列内的作业会被调度执行Paused 表示队列暂停，作业不再会被调度执行，
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;是否必传：否
        state: 'Active',
        // 第几页，默认值1;是否必传：否
        // pageNumber: 0,
        // 每页个数，默认值10;是否必传：否
        // pageSize: 0,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
getAIQueue();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：GET	String	是
Url	请求url，固定值	String	是
Key	固定值：ai_queue	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
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调度执行Paused 表示队列暂停，作业不再会被调度执行，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
pageNumber	第几页，默认值1。	Integer	否

pageSize	每页个数，默认值10。	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果 详情请参见 查询AI内容识别队列	Object

相关参考

API 文档

更新AI内容识别队列

最近更新时间：2024-12-27 16:37:05

简介

更新AI内容识别队列功能是由 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。
本文档提供关于更新AI内容识别队列的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
更新AI内容识别队列	本接口用于更新AI 内容识别（异步）的队列。

更新AI内容识别队列

功能说明

本接口用于更新AI 内容识别（异步）的队列。

使用示例

```
function updateAIQueue() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const queueId = 'xxxx'; // 队列id
  const key = `ai_queue/${queueId}`; // 固定格式
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 队列名称，仅支持中文、英文、数字、_、-和*，长度不超过 128;是否必传：是
      Name: 'my_queue',
      // Active 表示队列内的作业会被调度执行Paused 表示队列暂停，作业不再会被调度执行，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;是否必传：是
      State: 'Active',
      // 回调配置;是否必传：是
      NotifyConfig: {
```

```
        // 回调开关OffOn;是否必传：否
        State: 'On',
        // 回调事件TaskFinish：任务完成WorkflowFinish：工作流完成;是否必传：否
        Event: 'TaskFinish',
        // 回调格式XMLJSON;是否必传：否
        // ResultFormat: '',
        // 回调类型UrlTDMQ;是否必传：否
        Type: 'Url',
        // 回调地址，不能为内网地址。;是否必传：否
        Url: 'http://example.com',
        // TDMQ 使用模式Topic：主题订阅Queue：队列服务;是否必传：否
        // MqMode: '',
        // TDMQ 所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、
hk（中国香港）;是否必传：否
        // MqRegion: '',
        // TDMQ 主题名称;是否必传：否
        // MqName: '',
      },
    },
  });

  cos.request(
    {
      Method: 'PUT', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
updateAIQueue();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：PUT	String	是
Url	请求url，固定值	String	是
Key	ai_queue/{queueId}	String	是
ContentType	固定值	String	是
Body	请求体详情请参见 更新AI内容识别队列	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果 详情请参见 更新AI内容识别队列	Object

相关参考

API 文档

图片上色

最近更新时间：2025-02-11 15:37:22

简介

本文档提供关于数据万象图片上色的 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> - COS Javascript SDK 版本需要使用 v1.3.2及以上版本。
> - 待处理的图片，原图大小不超过5MB。

API	说明
图片上色	腾讯云数据万象通过 AIImageColoring 接口对黑白图像进行上色。

上传时处理

下面示例展示了如何在上传图片时自动实现图片上色。
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) {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.putObject(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: file.name, // 对象文件名，例如：folder/document.jpg，可以设置为本地文件的
名称file.name
      Body: file,  // 选择要上传的文件
      Headers: {
        // 通过 AIImageColoring 接口使用图片上色功能
        'Pic-Operations': JSON.stringify({
          is_pic_info: 1,
```

```
            rules: [{ fileid: 'test.jpg', rule: 'ci-process=AIImageColoring' }]
          }),
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(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);
};
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容，一般传入选择的文件 file 对象	File/Blob	是
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AIImageColoring	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图片上色	Object

对云上数据进行图片上色

下面示例展示了如何在对已存储在 COS 的图片进行图片上色操作，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上色 -->
<input id="submitBtn" type="submit" />
```

```
function handleFileInBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
```

```
        Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
        Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
        Method: 'POST',  // 固定值
        Action: 'image_process',  // 固定值
        Headers: {
          // 通过 AIImageColoring 接口使用图片上色功能
          'Pic-Operations': JSON.stringify({
            rules: [{ fileid: 'test.jpg', rule: 'ci-process=AIImageColoring' }]
           }),
        },
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      }
    );
}

document.getElementById('submitBtn').onclick = function (e) {
  handleFileInBucket();
};
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AIImageColoring	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图片上色	Object

下载时使用图片上色

下面示例展示了如何在下载图片时实现图片上色。

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上色 -->
<input id="submitBtn" type="submit" />
```

```
function getObject() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Key: '1.jpg',  // 对象文件名，例如：folder/document.jpg。
      Query: {
        'ci-process': 'AIImageColoring',
```

```
        // 'detect-url': '', // 待处理图片url，与Key二选一传递
      },
      RawBody: true, // 固定为true，不解析请求体Body
      DataType: 'blob', // 设置为blob，返回Body为图片blob
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	否
Query	其他请求参数	Container	是
RawBody	固定为 true，不解析请求体 Body	Boolean	是
DataType	设置为 blob，返回 Body 为图片 blob	String	是

Query 参数说明：

参数名称	描述	类型	是否必选

ci-process	数据万象处理能力，图片上色固定为 AIImageColoring	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接 detect-url 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 图片上色	Object

相关链接

- API 文档
- 功能指南

图像超分

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图像超分的 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> - COS Javascript SDK 版本需要使用 v1.3.2及以上版本。
> - 待处理的图片，原图大小不超过5MB，分辨率不得超过1920px * 1080px。

API	说明
图像超分	腾讯云数据万象通过 AISuperResolution 接口对图像进行超分辨率处理，当前默认超分为宽高的2倍。

上传时处理

下面示例展示了如何在上传图片时自动实现图像超分。
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) {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.putObject(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: file.name, // 对象文件名，例如：folder/document.jpg，可以设置为本地文件的
名称file.name
      Body: file,  // 选择要上传的文件
      Headers: {
        // 通过 AISuperResolution 接口使用图像超分功能
        'Pic-Operations': JSON.stringify({
```

```
            is_pic_info: 1,
            rules: [{ fileid: 'test.jpg', rule: 'ci-process=AISuperResolution'
}]
          }),
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(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);
};
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容，一般传入选择的文件 file 对象	File/Blob	是
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AISuperResolution	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图片超分	Object

对云上数据进行图像超分

下面示例展示了如何对已存储在 COS 的图片进行图像超分操作，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像超分 -->
<input id="submitBtn" type="submit" />
```

```
function handleFileInBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
```

```
        Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
        Method: 'POST',  // 固定值
        Action: 'image_process',  // 固定值
        Headers: {
          // 通过 AISuperResolution 接口使用图像超分功能
          'Pic-Operations': JSON.stringify({
          rules: [{ fileid: 'test.jpg', rule: 'ci-process=AISuperResolution' }]
         }),
        },
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      }
    );
}

document.getElementById('submitBtn').onclick = function (e) {
  handleFileInBucket();
};
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AISuperResolution	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 图片超分	Object

下载时使用图像超分

下面示例展示了如何下载图片时实现图像超分。

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像超分 -->
<input id="submitBtn" type="submit" />
```

```
function getObject() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
```

```
      Method: 'GET',  // 固定值
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Query: {
        'ci-process': 'AISuperResolution',  // 固定值
        // 'detect-url': '', // 待处理图片url，与Key二选一传递
      },
      RawBody: true, // 固定为true，不解析请求体Body
      DataType: 'blob', // 设置为blob，返回Body为图片blob
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	否
Query	其他请求参数	Container	是
RawBody	固定为 true，不解析请求体 Body	Boolean	是
DataType	设置为 blob，返回 Body 为图片 blob	String	是

Query 参数说明：

参数名称	描述	类型	是否必选
ci-process	数据万象处理能力，图像超分固定为 AISuperResolution	String	是
detect-url	处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接 detect-url 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 图像超分	Object

相关链接

- API 文档
- 功能指南

图像增强

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图像增强的 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> - COS Javascript SDK 版本需要使用 v1.3.2及以上版本。
> - 待处理的图片，原图大小不超过5MB。

API	说明
图像增强	腾讯云数据万象通过 AIEnhanceImage 接口对图像进行增强处理。

上传时处理

下面示例展示了如何在上传图片时自动实现图像增强。
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) {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.putObject(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: file.name, // 对象文件名，例如：folder/document.jpg，可以设置为本地文件的名称file.name
      Body: file,  // 选择要上传的文件
      Headers: {
        // 通过 AIEnhanceImage 接口使用图像增强功能：指定图片锐化强度值 5，去噪强度值 3
        'Pic-Operations': JSON.stringify({
          is_pic_info: 1,
```

```
            rules: [{ fileid: 'test.jpg', rule: 'ci-process=AIEnhanceImage' }]
          }),
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(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);
};
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容，一般传入选择的文件 file 对象	File/Blob	是
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AIEnhanceImage	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果，详情请参见 图像增强	Object

对云上数据进行图像增强

下面示例展示了如何在对已存储在 COS 的图片进行图像增强操作，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像增强 -->
<input id="submitBtn" type="submit" />
```

```
function handleFileInBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
```

```
        Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
        Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
        Method: 'POST',  // 固定值
        Action: 'image_process',  // 固定值
        Headers: {
          // 通过 AIEnhanceImage 接口使用图像增强功能：指定图片锐化强度值 5，去噪强度值 3
          'Pic-Operations': JSON.stringify({
            rules: [{ fileid: 'test.jpg', rule: 'ci-process=AIEnhanceImage' }]
           }),
        },
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      }
    );
}

document.getElementById('submitBtn').onclick = function (e) {
  handleFileInBucket();
};
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AIEnhanceImage	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
Response	响应结果，详情请参见 图像增强	Object

下载时使用图像增强

下面示例展示了如何在下载图片时实现图像增强。

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像增强 -->
<input id="submitBtn" type="submit" />
```

```
function getObject() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
```

```
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Query: {
        'ci-process': 'AIEnhanceImage',  // 固定值
        denoise: 5, // 非必需，去噪强度值，取值范围为 0 - 5 之间的整数，值为 0 时不进行去噪操作，默认值为3
        sharpen: 5, // 非必需，锐化强度值，取值范围为 0 - 5 之间的整数，值为 0 时不进行锐化操作，默认值为3*/,
        detect-url: 'http://www.example.com/abc.jpg'  // 非必须，可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，detect-url 需要进行 UrlEncode
      },
      RawBody: true, // 固定为true，不解析请求体Body
      DataType: 'blob', // 设置为blob，返回Body为图片blob
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	否
Query	其他请求参数	Container	是

RawBody	固定为 true，不解析请求体 Body	Boolean	是
DataType	设置为 blob，返回 Body 为图片 blob	String	是

Query 参数说明：

参数名称	描述	类型	是否必选
ci-process	数据万象处理能力，图像增强固定为 AIEnhanceImage。	String	是
denoise	去噪强度值，取值范围为0 - 5之间的整数，值为0时不进行去噪操作，默认值为3。	Integer	否
sharpen	锐化强度值，取值范围为0 - 5之间的整数，值为0时不进行锐化操作，默认值为3。	Integer	否
detect-url	处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。detect-url 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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

– Response	响应结果，详情请参见 图像增强	Object

相关链接

- API 文档
- 功能指南

图像智能裁剪

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图像智能裁剪的 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> - COS Javascript SDK 版本需要使用 v1.3.2及以上版本。
> - 待处理的图片，原图大小不超过5MB。

API	说明
图像智能裁剪	腾讯云数据万象通过 AIImageCrop 接口对图像进行智能裁剪。

上传时处理

下面示例展示了如何在上传图片时自动实现图像智能裁剪。
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) {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.putObject(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: file.name, // 对象文件名，例如：folder/document.jpg，可以设置为本地文件的
名称file.name
      Body: file,  // 选择要上传的文件
      Headers: {
        // 通过 AIImageCrop 接口使用图像智能裁剪功能：指定图片剪区域的宽度 1，高度 2
        'Pic-Operations': JSON.stringify({
```

```
            is_pic_info: 1,
            rules: [{ fileid: 'test.jpg', rule: 'ci-
process=AIImageCrop&width=1&height=2' }]
          }),
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(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);
};
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容，一般传入选择的文件 file 对象	File/Blob	是
Pic-Operations	详情请参见 图像智能裁剪处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图像智能裁剪	Object

对云上数据进行图像智能裁剪

下面示例展示了如何在对已存储在 COS 的图片进行图像智能裁剪操作，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像智能裁剪 -->
<input id="submitBtn" type="submit" />
```

```
function handleFileInBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
```

```
        Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
        Method: 'POST',  // 固定值
        Action: 'image_process',  // 固定值
        Headers: {
          // 通过 AIImageCrop 接口使用图像智能裁剪功能：指定图片剪区域的宽度 1，高度 2
          'Pic-Operations': JSON.stringify({
            rules: [{ fileid: 'test.jpg', rule: 'ci-
process=AIImageCrop&width=1&height=2' }]
          }),
        },
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      }
    );
}

document.getElementById('submitBtn').onclick = function (e) {
  handleFileInBucket();
};
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是
Pic-Operations	详情请参见 图像智能裁剪处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图像智能裁剪	Object

下载时使用图像智能裁剪

下面示例展示了如何在下载图片时实现图像智能裁剪。

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像智能裁剪 -->
<input id="submitBtn" type="submit" />
```

```
function getObject() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
```

```
        Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
        Method: 'GET',  // 固定值
        Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
        Query: {
          'ci-process': 'AIImageCrop',  // 固定值
          width: 5, // 必需，需要裁剪区域的宽度
          height: 5, // 必需，需要裁剪区域的高度
          //fixed:1, // 非必需，是否严格按照 width 和 height 的值进行输出
          //detect-url: 'http://www.example.com/abc.jpg'  // 非必须，可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，detect-url 需要进行 UrlEncode
        },
        RawBody: true, // 固定为true，不解析请求体Body
        DataType: 'blob', // 设置为blob，返回Body为图片blob
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      }
    );
}
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg。	String	否
Query	其他请求参数	Container	是
RawBody	固定为 true，不解析请求体 Body	Boolean	是

DataType	设置为 blob，返回 Body 为图片 blob	String	是

Query 参数说明：

参数名称	描述	类型	是否必选
ci-process	数据万象处理能力，智能裁剪固定为 AIImageCrop	String	是
width	需要裁剪区域的宽度，与 height 共同组成所需裁剪的图片宽高比例；输入数字请大于0、小于图片宽度的像素值	Integer	是
height	需要裁剪区域的高度，与 width 共同组成所需裁剪的图片宽高比例；输入数字请大于0、小于图片高度的像素值；width : height 建议取值在[1, 2.5]之间，超过这个范围可能会影响效果	Integer	是
fixed	是否严格按照 width 和 height 的值进行输出 • 取值为0时，宽高比例（width : height）会简化为最简分数，即如果 width 输入10、height 输入20，会简化为1：2 • 取值为1时，输出图片的宽度等于 width，高度等于 height 默认值为0	Integer	否
detect-url	处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接 detect-url 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-statusC	请求返回的 HTTP 状态码，例如200、403、404等	Number

ode		
– headers	请求返回的头部信息	Object
– Respon se	响应结果，详情请参见 图像智能裁剪	Object

相关链接

- API 文档
- 功能指南

图像修复

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图像修复的 API 概览以及 SDK 示例代码。

> **注意：**
> - COS Javascript SDK 版本需要使用 v1.3.2及以上版本。
> - 待处理的图片，原图大小不超过32MB。
> - 宽高不超过50000像素且总像素不超过2.5亿像素，处理结果图宽高设置不超过9999像素。

API	说明
图像修复	腾讯云数据万象通过 ImageRepair 接口检测图片中的水印并将其擦除。

上传时处理

下面示例展示了如何在上传图片时自动实现图像修复。
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) {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.putObject(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: file.name, // 对象文件名，例如：folder/document.jpg，可以设置为本地文件的
名称file.name
      Body: file,  // 选择要上传的文件
      Headers: {
```

```
        // 通过 ImageRepair 接口使用图像修复功能：指定遮罩图片地址 URL 安全的 Base64
编码：aHR0c...XXX
        'Pic-Operations': JSON.stringify({
          is_pic_info: 1,
          rules: [{ fileid: 'test.jpg', rule: 'ci-
process=ImageRepair&MaskPic=aHR0c...XXX' }]
         }),
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(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);
};
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg。	String	是
Body	文件内容，一般传入选择的文件 file 对象	File/Blob	是
Pic-Operati	详情请参见 图像修复处理参数	JsonString	是

ons			

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果，详情请参见 图像修复	Object

对云上数据进行图像修复

下面示例展示了如何在对已存储在 COS 的图片进行图像修复操作，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像修复 -->
<input id="submitBtn" type="submit" />
```

```
function handleFileInBucket() {
  // sdk引入以及初始化请参考: https://cloud.tencent.com/document/product/436/11459
  cos.request(
```

```
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Method: 'POST',  // 固定值
      Action: 'image_process',  // 固定值
      Headers: {
        // 通过 ImageRepair 接口使用图像修复功能：指定遮罩图片地址 URL 安全的 Base64
编码：aHR0c...XXX
        'Pic-Operations': JSON.stringify({
          rules: [{ fileid: 'test.jpg', rule: 'ci-
process=ImageRepair&MaskPic=aHR0c...XXX' }]
         }),
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}

document.getElementById('submitBtn').onclick = function (e) {
  handleFileInBucket();
};
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg。	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是

Pic-Operations	详情请参见 图像修复处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 图像修复	Object

下载时使用图像修复

下面示例展示了如何在下载图片时实现图像修复。

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像修复 -->
<input id="submitBtn" type="submit" />
```

```
function getObject() {
```

```
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Query: {
        'ci-process': 'ImageRepair',  // 固定值
        MaskPic: 'aHR0c...XXX' /*非必需，遮罩（白色区域为需要去除的水印位置）图片地址，需要经过 URL 安全的 Base64 编码*/,
        //MaskPoly:'[[[608, 794], [1024, 794]],[[1295, 62],[1295, 30]]]' /* 非必需，顺时针输入多边形的每个点的坐标,MaskPoly 同时与 MaskPic 填写时，优先采用 MaskPic 的值*/,
      },
      RawBody: true, // 固定为true，不解析请求体Body
      DataType: 'blob', // 设置为blob，返回Body为图片blob
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是

Query	其他请求参数	Container	是
RawBody	固定为 true，不解析请求体 Body	Boolean	是
DataType	设置为 blob，返回 Body 为图片 blob	String	是

Query 参数说明：

参数名称	描述	类型	是否必选
ci-process	数据万象处理能力，图像修复固定为 ImageRepair	String	是
MaskPic	遮罩（白色区域为需要去除的水印位置）图片地址，需要经过 URL 安全的 Base64 编码。例如，遮罩图片为 `http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/shuiyin_2.png`，则该处编码后的字符串为 `aHR0cDovL2V4YW1wbGVidWNrZXQtMTI1MDAwMDAwMC5jb3MuYXAtc2hhbmdoYWkubXlxY2xvdWQuY29tL3NodWl5aW5fMi5wbmc`	String	否
MaskPoly	例如：[[[608, 794], [1024, 794], [1024, 842], [608, 842]],[[1295, 62], [1295, 30], [1597, 32],[1597, 64]]]，顺时针输入多边形的每个点的坐标,每个多边形：[[x1, y1], [x2, y2]...]，形式为三维矩阵（多个多边形：[多边形1, 多边形2]）或二维矩阵（单个多边形），且需要经过 URL 安全的 Base64 编码。MaskPoly 同时与 MaskPic 填写时，优先采用 MaskPic 的值	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 图像修复	Object

相关链接

- API 文档
- 功能指南

图片二维码

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于图片二维码相关的 API 概览以及 SDK 示例代码。

API	说明
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。
二维码生成	二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码。

二维码识别

功能说明

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。

请求示例1：上传时识别

```
<!-- 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 identifyQrcode_put(file) {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  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 */
  };
  // 通过input组件选择文件得到 file 代码省略
  cos.putObject({
      Bucket: config.Bucket,
      Region: config.Region,
      Key: '上传二维码.png', // 对象文件名，例如：folder/document.jpg，可以设置为本地文件的名称file.name
      Body: file, // file 是 input[type="file"]标签选择本地文件后得到的 file 对象
```

```
        Headers: {
          'Pic-Operations': JSON.stringify({
            is_pic_info: 1,
            rules: [{ fileid: 'test.jpg', rule: 'QRcode/cover/0' }]
          }),
        },
        onProgress: function (progressData) {
            console.log('onProgress', JSON.stringify(progressData));
        },
    }, function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(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;
    }
    identifyQrcode_put(file);
};
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容，一般传入选择的文件 file 对象	File/Blob	是
Pic-Operations	详情请参见 图像修复处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 二维码识别	Object

请求示例2：下载时识别

```
function identifyQrcode_get() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.request({
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Query: {
        'ci-process': 'QRcode'  // 固定值
      },
  },
  function(err, data){
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
```

```
        }
    });
}
identifyQrcode_get();
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必填
ci-process	万象处理能力，二维码识别固定为 QRcode	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 二维码识别	Object

二维码生成

功能说明

二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码。

示例代码

```
function generateQrcode() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.request({
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Query: {
        'ci-process': 'qrcode-generate', /* 必须，对象存储处理能力，二维码生成参数为qrcode-generate */
        'qrcode-content': '二维码文案', /* 必须，可识别的二维码文本信息     */
        // mode: 0, /* 非必须，生成的二维码类型，可选值：0或1。0为二维码，1为条形码，默认值为0  */
        width: 200, /* 必须，指定生成的二维码或条形码的宽度，高度会进行等比压缩   */
      },
  },
  function(err, data){
    if (!err) {
      // 获得二维码 base64
      var imgBase64 = data.Response.ResultImage;
      // 比如可拼接前缀直接展示在 img 里
      // document.querySelector('#img').src = '' + imgBase64;
    }
  });
}
generateQrcode();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，二维码生成参数为 qrcode-generate	String	是
qrcode-content	可识别的二维码文本信息	String	是
mode	生成的二维码类型，可选值：0或1。0为二维码，1为条形码，默认值为0	Integer	否
width	指定生成的二维码或条形码的宽度，高度会进行等比压缩	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 图片二维码生成	Object

相关链接

API 文档

图片标签

最近更新时间：2025-06-09 19:23:32

简介

本文档提供关于图片标签的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
图片标签	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。

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。

使用示例

```
function detectLabel() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/dog.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 固定值detect-label;是否必传：是
        'ci-process': 'detect-label',
        // 本次调用支持的识别场景，可选值如下：web，针对网络图片优化；camera，针对手机摄像头拍摄图片优化；album，针对手机相册、网盘产品优化；news，针对新闻、资讯、广电等行业优化；如果不传此参数，则默认为camera。支持多场景（scenes）一起检测，以，分隔。例如，使用scenes=web，camera 即对一张图片使用两个模型同时检测，输出两套识别结果。;是否必传：否
        // scenes: '',
```

```
          // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url
时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写
Key 。detect-url 示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后
为http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg;是否必传：否
          // 'detect-url': '',
        },
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      }
    );
}
detectLabel();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	固定值detect-label	String	是

s			
scenes	本次调用支持的识别场景，如果不传此参数，则默认为camera。可选值如下： • web，针对网络图片优化； • camera，针对手机摄像头拍摄图片优化； • album，针对手机相册、网盘产品优化； • news，针对新闻、资讯、广电等行业优化； 支持多场景（scenes）一起检测，以，分隔。例如，使用 scenes=web，camera 即对一张图片使用两个模型同时检测，输出两套识别结果。	String	否
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 Key。 detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果，详情请参见 图片标签	Object

相关链接

- API 文档
- 功能指南

图片质量评分

最近更新时间：2024-12-27 16:37:05

简介

本文档提供关于数据万象图片质量评分的 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> - COS Javascript SDK 版本需要大于等于 v1.3.2。
> - 图片支持格式：PNG、JPG、JPEG。
> - 图片大小：所下载图片经 Base64 编码后不超过 4MB。
> - 图片像素：建议大于 50*50 像素，否则影响识别效果。
> - 长宽比：建议长边与短边的比例小于 5:1。

API	说明
图片质量评分	图片质量评分功能为同步请求方式，您可以通过本接口对图片文件进行检测，从多方面评估，最终给出综合可观的清晰度评分和主观的美观度评分。

图片质量评分

功能说明

图片质量评分功能为同步请求方式，您可以通过本接口对图片文件进行检测，从多方面评估，最终给出综合可观的清晰度评分和主观的美观度评分。该接口属于 GET 请求。

使用示例

```
function assessQuality() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/dog.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，图像质量检测固定为 AssessQuality。;是否必传：是
```

```
        'ci-process': 'AssessQuality',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
assessQuality();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，必须，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，图像质量检测固定为 AssessQuality。	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果 详情请参见 图片质量评分	Object

相关链接

API 文档
功能指南

图片文字识别（OCR）

最近更新时间：2024-12-27 16:37:05

简介

本文档提供关于图片文字识别(OCR)的相关 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
图片文字识别(OCR)	通用文字识别功能（Optical Character Recognition，OCR）基于行业前沿的深度学习技术，将图片上的文字内容，智能识别为可编辑的文本，可应用于随手拍扫描、纸质文档电子化、电商广告审核等多种场景，大幅提升信息处理效率。

图片文字识别(OCR)

功能说明

通用文字识别功能（Optical Character Recognition，OCR）基于行业前沿的深度学习技术，将图片上的文字内容，智能识别为可编辑的文本，可应用于随手拍扫描、纸质文档电子化、电商广告审核等多种场景，大幅提升信息处理效率。

使用示例

```
function cOSOCR() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/0.jpg', // 对象文件名，例如：folder/document.jpg；与detect-url二选一
传递
      Query: {
        // 数据万象处理能力，图片文字识别固定为OCR;是否必传：是
        'ci-process': 'OCR',
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url
时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写
```

```
Key。detect-url 示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后为http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg;是否必传：否
        // 'detect-url': '',
        // ocr的识别类型，有效值为general，accurate，efficient，fast，handwriting。general表示通用印刷体识别；accurate表示印刷体高精度版；efficient表示印刷体精简版；fast表示印刷体高速版；handwriting表示手写体识别。默认值为general。;是否必传：否
        type: 'general',
        // type值为general时有效，表示识别语言类型。支持自动识别语言类型，同时支持自选语言种类，默认中英文混合(zh)，各种语言均支持与英文混合的文字识别。可选值：zh：中英混合zh_rare：支持英文、数字、中文生僻字、繁体字，特殊符号等auto：自动mix：混合语种jap：日语kor：韩语spa：西班牙语fre：法语ger：德语por：葡萄牙语vie：越语may：马来语rus：俄语ita：意大利语hol：荷兰语swe：瑞典语fin：芬兰语dan：丹麦语nor：挪威语hun：匈牙利语tha：泰语hi：印地语ara：阿拉伯语;是否必传：否
        'language-type': 'zh',
        // type值为general，fast时有效，表示是否开启PDF识别，有效值为true和false，默认值为false，开启后可同时支持图片和PDF的识别。;是否必传：否
        ispdf: false,
        // type值为general，fast时有效，表示需要识别的PDF页面的对应页码，仅支持PDF单页识别，当上传文件为PDF且ispdf参数值为true时有效，默认值为1。;是否必传：否
        // 'pdf-pagenumber': 0,
        // type值为general，accurate时有效，表示识别后是否需要返回单字信息，有效值为true和false，默认为false;是否必传：否
        isword: false,
        // type值为handwriting时有效，表示是否开启单字的四点定位坐标输出，有效值为true和false，默认值为false。;是否必传：否
        'enable-word-polygon': false,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
cOSOCR();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，图片文字识别固定为OCR	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 Key。 detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`	String	否
type	OCR 的识别类型，默认值为 general。有效值有： • general：表示通用印刷体识别。 • accurate：表示印刷体高精度版。 • efficient：表示印刷体精简版。 • fast：表示印刷体高速版。 • handwriting：表示手写体识别。	String	否
languageType	type 值为 general 时有效，表示识别语言类型。支持自动识别语言类型，同时支持自选语言种类，默认中英文混合(zh)。 各种语言均支持与英文混合的文字识别。可选值有： • zh：中英混合 • zh_rare：支持英文、数字、中文生僻字、繁体字，特殊符号等 • auto：自动 • mix：混合语种 • jap：日语	String	否

	• kor：韩语 • spa：西班牙语 • fre：法语 • ger：德语 • por：葡萄牙语 • vie：越语 • may：马来语 • rus：俄语 • ita：意大利语 • hol：荷兰语 • swe：瑞典语 • fin：芬兰语 • dan：丹麦语 • nor：挪威语 • hun：匈牙利语 • tha：泰语 • hi：印地语 • ara：阿拉伯语		
ispdf	type 值为 general，fast 时有效，表示是否开启 PDF 识别，有效值为 true 和 false，默认值为 false，开启后可同时支持图片和 PDF 的识别。	boolean	否
pdfPagenumber	type 值为 general，fast 时有效，表示需要识别的 PDF 页面的对应页码，仅支持 PDF 单页识别，当上传文件为 PDF且 ispdf 参数值为 true 时有效，默认值为1。	Integer	否
isword	type 值为 general，accurate 时有效，表示识别后是否需要返回单字信息，有效值为 true 和false，默认为 false。	boolean	否
enableWordPolygon	type 值为 handwriting 时有效，表示是否开启单字的四点定位坐标输出，有效值为 true 和false，默认值为 false。	boolean	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图片文字识别(OCR)	Object

相关链接

API 文档
功能指南

实时文字翻译

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于实时文字翻译的相关 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
实时文字翻译	腾讯云数据万象通过 AutoTranslationBlock 接口对文字块进行翻译，请求时需要携带签名。

实时文字翻译

功能说明

腾讯云数据万象通过 AutoTranslationBlock 接口对文字块进行翻译，请求时需要携带签名。

使用示例

```
function autoTranslationBlock() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Query: {
        // 数据万象处理能力，文本块翻译固定为 AutoTranslationBlock。;是否必传：是
        'ci-process': 'AutoTranslationBlock',
        // 待翻译的文本;是否必传：是
        InputText: '我是张三',
        // 输入语言，如 "zh";是否必传：是
        SourceLang: 'zh',
        // 输出语言，如 "en";是否必传：是
        TargetLang: 'en',
        // 文本所属业务领域，如：  "ecommerce", //缺省值为 general;是否必传：否
        // TextDomain: '',
```

```
          // 文本类型，如: "title", //缺省值为 sentence;是否必传：否
          // TextStyle: '',
        },
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      }
    );
}
autoTranslationBlock();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，文本块翻译固定为 AutoTranslationBlock。	String	是
InputText	待翻译的文本	String	是
SourceLang	输入语言，如 "zh"	String	是

TargetLang	输出语言，如 "en"	String	是
TextDomain	文本所属业务领域，如: "ecommerce", 缺省值为 general	String	否
TextStyle	文本类型，如: "title", 缺省值为 sentence	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果，详情请参见 实时文字翻译	Object

相关链接

API 文档

人脸识别
人脸检测

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于人脸检测的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
人脸检测	人脸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人脸位置。该接口属于 GET 请求。

人脸检测

功能说明

人脸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人脸位置。该接口属于 GET 请求。

使用示例

```
function aIDetectFace() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/1.jpg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，人脸特效固定为 DetectFace。;是否必传：是
        'ci-process': 'DetectFace',
        // 最多处理的人脸数目。默认值为1（仅检测图片中面积最大的那张人脸），最大值为120。此参数用于控制处理待检测图片中的人脸个数，值越小，处理速度越快。;是否必传：否
        'max-face-num': 1,
      },
    },
```

```
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
aIDetectFace();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg。	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，人脸特效固定为 DetectFace。	String	是
max-face-num	最多处理的人脸数目。默认值为1（仅检测图片中面积最大的那张人脸），最大值为120。此参数用于控制处理待检测图片中的人脸个数，值越小，处理速度越快。	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果 详情请参见 人脸检测	Object

相关链接

API 文档

人脸特效

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于人脸特效的相关 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
人脸特效	人脸特效，提供人脸美颜、人像变换、人像分割功能。

人脸特效

功能说明

人脸特效，提供人脸美颜、人像变换、人像分割功能。

使用示例

```
function aIFaceEffect() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/1.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 万象处理能力，人脸特效固定为face-effect;是否必传：是
        'ci-process': 'face-effect',
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。 detect-url 示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后为 http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg。;是否必传：否
        // 'detect-url': '',
        // 人脸特效类型，人脸美颜：face-beautify；人脸性别转换：face-gender-transformation；人脸年龄变化：face-age-transformation；人像分割：face-segmentation;
```

```
是否必传：是
        type: 'face-beautify',
        // type为face-beautify时生效，美白程度，取值范围[0,100]。0不美白，100代表最高程度。默认值30;是否必传：否
        whitening: 50,
        // type为face-beautify时生效，磨皮程度，取值范围[0,100]。0不磨皮，100代表最高程度。默认值10;是否必传：否
        smoothing: 50,
        // type为face-beautify时生效，瘦脸程度，取值范围[0,100]。0不瘦脸，100代表最高程度。默认值70;是否必传：否
        faceLifting: 50,
        // type为face-beautify时生效，大眼程度，取值范围[0,100]。0不大眼，100代表最高程度。默认值70;是否必传：否
        eyeEnlarging: 50,
        // type为face-gender-transformation时生效，选择转换方向，0：男变女，1：女变男。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。;是否必传：否
        // gender: 0,
        // type为face-age-transformation时生效，变化到的人脸年龄,[10,80]。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。;是否必传：否
        // age: 0,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
aIFaceEffect();
```

参数说明

cos.request 方法参数说明：

参数名称	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

Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	万象处理能力，人脸特效固定为face-effect。	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg` 。	String	否
type	人脸特效类型，可选值有： • face-beautify：人脸美颜； • face-gender-transformation：人脸性别转换 • face-age-transformation：人脸年龄变化 • face-segmentation：人像分割	String	是
whitening	type 为 face-beautify 时生效，美白程度，取值范围[0,100]。0不美白，100代表最高程度。默认值30	Integer	否
smoothing	type 为 face-beautify 时生效，磨皮程度，取值范围[0,100]。0不磨皮，100代表最高程度。默认值10	Integer	否
faceLifting	type 为 face-beautify 时生效，瘦脸程度，取值范围[0,100]。0不瘦脸，100代表最高程度。默认值70	Integer	否
eyeEnlarging	type 为 face-beautify 时生效，大眼程度，取值范围[0,100]。0不大眼，100代表最高程度。默认值70	Integer	否
gender	type 为 face-gender-transformation 时生效，选择转换方向，0：男变女，1：女变男。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。	Integer	否
age	type 为 face-age-transformation时生效，变化到的人脸年龄，[10,80]。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。	Integer	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) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 人脸特效	Object

相关链接

API 文档
功能指南

人体识别

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于人体识别的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
人体识别	腾讯云数据万象通过 AIBodyRecognition 接口识别并输出画面中人体，输出其位置（矩形框）和置信度。图片人体识别请求包属于 GET 请求，请求时需要携带签名。

人体识别

功能说明

腾讯云数据万象通过 AIBodyRecognition 接口识别并输出画面中人体，输出其位置（矩形框）和置信度。图片人体识别请求包属于 GET 请求，请求时需要携带签名。

使用示例

```
function aIBodyRecognition() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/1.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，人体识别固定为AIBodyRecognition;是否必传：是
        'ci-process': 'AIBodyRecognition',
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 Key。detect-url 示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后为 http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg;是否必传：否
        // 'detect-url': '',
```

```
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
aIBodyRecognition();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，人体识别固定为 AIBodyRecognition	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 Key。	String	否

	detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`		

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 人体识别	Object

相关链接

API 文档

宠物识别

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于宠物识别的相关 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
宠物识别	腾讯云数据万象通过 detect-pet 接口识别并输出画面中宠物，输出其位置（矩形框）和置信度。图片宠物识别请求包属于 GET 请求，请求时需要携带签名。

宠物识别

功能说明

腾讯云数据万象通过 detect-pet 接口识别并输出画面中宠物，输出其位置（矩形框）和置信度。图片宠物识别请求包属于 GET 请求，请求时需要携带签名。

使用示例

```
function aIDetectPet() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'dog.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，宠物识别固定为 detect-pet;是否必传：是
        'ci-process': 'detect-pet',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
```

```
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
aIDetectPet();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，宠物识别固定为 detect-pet	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果，详情请参见 宠物识别	Object

相关参考

API 文档

汽车识别

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于车辆车牌识别的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
车辆车牌识别	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。

车辆车牌识别

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。

使用示例

```
function aIDetectCar() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'car.png', // 对象文件名，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，车辆识别固定为 DetectCar;是否必传：是
        'ci-process': 'DetectCar',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
```

```
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
aIDetectCar();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，车辆识别固定为 DetectCar	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 车辆车牌识别	Object

相关链接

API 文档
功能指南

卡证识别

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于卡证识别的相关 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
卡证识别	本接口支持中国大陆居民二代身份证正面（暂不支持背面）、驾驶证主页（暂不支持副页）所有字段的自动定位，暂不支持文本识别，用于对特定字段的抹除、屏蔽，以及进一步的文本识别。

卡证识别

功能说明

本接口支持中国大陆居民二代身份证正面（暂不支持背面）、驾驶证主页（暂不支持副页）所有字段的自动定位，暂不支持文本识别，用于对特定字段的抹除、屏蔽，以及进一步的文本识别。

使用示例

```
function aILicenseRec() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'card.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，卡证识别固定为AILicenseRec;是否必传：是
        'ci-process': 'AILicenseRec',
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url
时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。detect-url
示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后为
http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg;是否必传：否
        // 'detect-url': '',
```

```
        // 卡证识别类型，有效值为IDCard，DriverLicense。<br>IDCard表示身份证；
DriverLicense表示驾驶证，默认：DriverLicense;是否必传：是
        CardType: 'IDCard',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
aILicenseRec();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg。	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，卡证识别固定为AILicenseRec	String	是

detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 、Key ，填写了 detect-url 时，后台会处理 detect-url 链接 。 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg` 。	String	否
CardType	卡证识别类型，默认为DriverLicense。有效值为： • IDCard：表示身份证。 • DriverLicense：表示驾驶证。	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果 详情请参见 卡证识别	Object

相关链接

API 文档

游戏场景识别

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于游戏场景识别的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
游戏场景识别	游戏标签功能实现游戏图片场景的识别，返回图片中置信度较高的游戏类别标签。游戏标签识别请求包属于 GET 请求，请求时需要携带签名。

游戏场景识别

功能说明

游戏标签功能实现游戏图片场景的识别，返回图片中置信度较高的游戏类别标签。游戏标签识别请求包属于 GET 请求，请求时需要携带签名。

使用示例

```
function aIGameRec() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/game.jpeg', // 对象文件名，例如：folder/document.jpg；与detect-url
二选一传递
      Query: {
        // 数据万象处理能力，游戏场景识别固定为 AIGameRec;是否必传：是
        'ci-process': 'AIGameRec',
        // 待检查图片url，需要进行urlencode;与Key二选一传递
        // 'detect-url': '',
      },
    },
```

```
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
aIGameRec();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，游戏场景识别固定为 AIGameRec。	String	是
detect-url	处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。 detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为	String	否

	`http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`。		

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果 详情请参见 游戏场景识别	Object

相关参考

API 文档

人脸核身
身份证识别

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于身份证识别的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
身份证识别	支持中国大陆居民二代身份证正反面所有字段的识别，包括姓名、性别、民族、出生日期、住址、公民身份证号、签发机关、有效期限；具备身份证照片、人像照片的裁剪功能和翻拍、PS、复印件告警功能，以及边框和框内遮挡告警、临时身份证告警和身份证有效期不合法告警等扩展功能。

身份证识别

功能说明

支持中国大陆居民二代身份证正反面所有字段的识别，包括姓名、性别、民族、出生日期、住址、公民身份证号、签发机关、有效期限；具备身份证照片、人像照片的裁剪功能和翻拍、PS、复印件告警功能，以及边框和框内遮挡告警、临时身份证告警和身份证有效期不合法告警等扩展功能。

使用示例

```
function aIIDCardOCR() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/card.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，身份证识别固定为 IDCardOCR;是否必传：是
        'ci-process': 'IDCardOCR',
```

```
        // FRONT：身份证有照片的一面（人像面）BACK：身份证有国徽的一面（国徽面）该参数如果不填，将为您自动判断身份证正反面;是否必传：否
        CardSide: 'FRONT',
        // 以下可选字段均为 bool 类型，默认 false：CropIdCard，身份证照片裁剪（去掉证件外多余的边缘、自动矫正拍摄角度）CropPortrait，人像照片裁剪（自动抠取身份证头像区域）CopyWarn，复印件告警BorderCheckWarn，边框和框内遮挡告警ReshootWarn，翻拍告警DetectPsWarn，PS 检测告警TempIdWarn，临时身份证告警InvalidDateWarn，身份证有效日期不合法告警Quality，图片质量分数（评价图片的模糊程度）MultiCardDetect，是否开启多卡证检测参数设置方式参考：Config = {"CropIdCard":true,"CropPortrait":true};是否必传：否
        // Config: {},
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
aIIDCardOCR();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，身份证识别固定为 IDCardOCR	String	是
CardSide	FRONT：身份证有照片的一面（人像面） BACK：身份证有国徽的一面（国徽面） 该参数如果不填，将为您自动判断身份证正反面	String	否
Config	以下可选字段均为 bool 类型，默认 false： • CropIdCard，身份证照片裁剪（去掉证件外多余的边缘、自动矫正拍摄角度） • CropPortrait，人像照片裁剪（自动抠取身份证头像区域） • CopyWarn，复印件告警 • BorderCheckWarn，边框和框内遮挡告警 • ReshootWarn，翻拍告警 • DetectPsWarn，PS 检测告警 • TempIdWarn，临时身份证告警 • InvalidDateWarn，身份证有效日期不合法告警 • Quality，图片质量分数（评价图片的模糊程度） • MultiCardDetect，是否开启多卡证检测 参数设置方式参考： Config = {"CropIdCard":true,"CropPortrait":true}	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Respon se	响应结果 详情请参见 身份证识别	Object

相关参考

API 文档

活体人脸核身

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于活体人脸核身的相关 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
活体人脸核身	集成了活体检测和跟权威库进行比对的能力，传入一段视频和姓名、身份证号信息即可进行验证。对录制的自拍视频进行活体检测，从而确认当前用户为真人，可防止照片、视频、静态3D建模等各种不同类型的攻击。检测为真人后，再判断该视频中的人与权威库的证件照是否属于同一个人，实现用户身份信息核实。

活体人脸核身

功能说明

集成了活体检测和跟权威库进行比对的能力，传入一段视频和姓名、身份证号信息即可进行验证。对录制的自拍视频进行活体检测，从而确认当前用户为真人，可防止照片、视频、静态3D建模等各种不同类型的攻击。检测为真人后，再判断该视频中的人与权威库的证件照是否属于同一个人，实现用户身份信息核实。

使用示例

```
function livenessRecognition() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/tf.mp4', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，人脸核身固定为 LivenessRecognition;是否必传：是
        'ci-process': 'LivenessRecognition',
        // 身份证号;是否必传：是
        IdCard: 'xxxxxxx',
```

```
          // 姓名。中文请使用 UTF-8编码;是否必传：是
          Name: '张三',
          // 活体检测类型，取值：LIP/ACTION/SILENTLIP 为数字模式，ACTION 为动作模式，
SILENT 为静默模式，三种模式选择一种传入;是否必传：是
          LivenessType: 'SILENT',
          // 数字模式传参：数字验证码（1234），需先调用接口获取数字验证码动作模式传参：传动作
顺序（2，1 or 1，2），需先调用接口获取动作顺序静默模式传参：空;是否必传：否
          ValidateData: '',
          // 需要返回多张最佳截图，取值范围1 - 10，不设置默认返回一张最佳截图;是否必传：否
          BestFrameNum: 1,
        },
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
livenessRecognition();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填

ci-process	数据万象处理能力，人脸核身固定为 LivenessRecognition	String	是
IdCard	身份证号	String	是
Name	姓名。中文请使用 UTF-8编码	String	是
LivenessType	活体检测类型，取值：LIP/ACTION/SILENTLIP 为数字模式，ACTION 为动作模式，SILENT 为静默模式，三种模式选择一种传入	String	是
ValidateData	数字模式传参：数字验证码（1234），需先调用接口获取数字验证码动作模式传参：传动作顺序（2，1 or 1，2），需先调用接口获取动作顺序静默模式传参：空	String	否
BestFrameNum	需要返回多张最佳截图，取值范围1 - 10，不设置默认返回一张最佳截图	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果 详情请参见 活体人脸核身	Object

相关参考

API 文档
功能指南

版权所有：腾讯云计算（北京）有限责任公司

获取数字验证码

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于获取数字验证码的相关 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
获取数字验证码	使用数字活体检测模式前，需调用本接口获取数字验证码。

获取数字验证码

功能说明

使用数字活体检测模式前，需调用本接口获取数字验证码。

使用示例

```
function getLiveCode() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Query: {
        // 数据万象处理能力，获取数字验证码固定为 GetLiveCode;是否必传：是
        'ci-process': 'GetLiveCode',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
```

```
            console.log(data.Response);
        }
    }
  );
}
getLiveCode();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，获取数字验证码固定为 GetLiveCode	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 获取数字验证码	Object

相关参考

API 文档

获取动作顺序

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于获取动作顺序的相关 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
获取动作顺序	使用动作活体检测模式前，需调用本接口获取动作顺序。

获取动作顺序

功能说明

使用动作活体检测模式前，需调用本接口获取动作顺序。

使用示例

```
function getActionSequence() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Query: {
        // 数据万象处理能力，获取动作顺序固定为 GetActionSequence;是否必传：是
        'ci-process': 'GetActionSequence',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
```

```
            console.log(data.Response);
        }
    }
  );
}
getActionSequence();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，获取动作顺序固定为 GetActionSequence	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 获取动作顺序	Object

相关参考

API 文档

商品抠图

最近更新时间：2025-01-25 18:31:21

简介

本文档提供关于商品抠图的相关 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
商品抠图	腾讯云数据万象通过 GoodsMatting 接口检测图片中的商品信息，生成只包含商品信息的图片，支持持久化、云上处理及下载时处理。

功能说明

腾讯云数据万象通过 GoodsMatting 接口检测图片中的商品信息，生成只包含商品信息的图片，支持持久化、云上处理及下载时处理。

使用示例

```
function goodsMatting() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/goods.jpeg', // 对象文件名，例如：folder/document.jpg；与detect-url
二选一传递
      Query: {
        // ;是否必传：是
        'ci-process': 'GoodsMatting',
        // ;是否必传：否
        // 'detect-url': 'https://example.com/abc.jpg',
      },
      RawBody: true,
      DataType: 'blob',
```

```
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
goodsMatting();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	否
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，商品抠图固定为 GoodsMatting。	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接 。 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请见 商品抠图	Object

相关链接

API 文档
功能指南

内容翻译
提交任务

最近更新时间：2024-12-27 16:37:06

简介

内容翻译功能是由 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。
本文档提供关于内容翻译的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
提交任务	提交一个翻译任务。

提交任务

功能说明

提交一个翻译任务。

使用示例

```
function postTranslation() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'jobs'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 创建任务的 Tag：Translation;是否必传：是
      Tag: 'Translation',
      // 待操作的对象信息;是否必传：是
      Input: {
        // 源文档文件名单文件（docx/xlsx/html/markdown/txt）：不超过800万字符有页数的（pdf/pptx）：不超过300页文本文件（txt）：不超过10MB二进制文件（pdf/docx/pptx/xlsx）：
```

```
不超过60MB图片文件（jpg/jpeg/png/webp）：不超过10MB;是否必传：是
        Object: 'ci/test.txt',
        // 文档语言类型zh：简体中文zh-hk：繁体中文zh-tw：繁体中文zh-tr：繁体中文en：英语
ar：阿拉伯语de：德语es：西班牙语fr：法语id：印尼语it：意大利语ja：日语pt：葡萄牙语ru：俄语
ko：韩语km：高棉语lo：老挝语;是否必传：是
        Lang: 'zh-hk',
        // 文档类型pdfdocxpptxxlsxtxtxmlhtml：只能翻译 HTML 里的文本节点，需要通过 JS
动态加载的不进行翻译markdownjpgjpegpngwebp;是否必传：是
        Type: 'txt',
        // 原始文档类型仅在 Type=pdf/jpg/jpeg/png/webp 时使用，当值为pdf时，仅支持
docx、pptx当值为jpg/jpeg/png/webp时，仅支持txt;是否必传：否
        // BasicType: '',
      },
      // 操作规则;是否必传：是
      Operation: {
        // 翻译参数;是否必传：是
        Translation: {
          // 目标语言类型源语言类型为 zh/zh-hk/zh-tw/zh-tr 时支持：en、ar、de、es、
fr、id、it、ja、it、ru、ko、km、lo、pt源语言类型为 en 时支持：zh、zh-hk、zh-tw、zh-
tr、ar、de、es、fr、id、it、ja、it、ru、ko、km、lo、pt其他类型时支持：zh、zh-hk、zh-
tw、zh-tr、en;是否必传：是
          Lang: 'en',
          // 文档类型，源文件类型与目标文件类型映射关系如下：docx：docxpptx：pptxxlsx：
xlsxtxt：txtxml：xmlhtml：htmlmarkdown：markdownpdf：pdf、docxpng：txtjpg：
txtjpeg：txtwebp：txt;是否必传：是
          Type: 'txt',
        },
        // 结果输出地址，当NoNeedOutput为true时非必选;是否必传：否
        Output: {
          // 存储桶的地域;是否必传：是
          Region: config.Region,
          // 存储结果的存储桶;是否必传：是
          Bucket: config.Bucket,
          // 输出结果的文件名;是否必传：是
          Object: 'ci/trans_test.txt',
        },
        // 透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
        // UserData: '',
        // 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;是否必传：否
        // JobLevel: '',
        // 仅输出结果，不生成结果文件。取值：true/false。该参数原文档类型为图片时有效。默
认值 false;是否必传：否
        // NoNeedOutput: '',
      },
      // 任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
      // CallBackFormat: '',
```

```
      // 任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
      // CallBackType: '',
      // 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;是否必传：否
      // CallBack: '',
      // 任务回调TDMQ配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig;是否必传：否
      // CallBackMqConfig: {
      // 消息队列所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）;是否必传：是
      // MqRegion: '',
      // 消息队列使用模式，默认 Queue ：主题订阅：Topic队列服务：Queue;是否必传：是
      // MqMode: '',
      // TDMQ 主题名称;是否必传：是
      // MqName: '',
      // },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
postTranslation();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填

Method	固定值：POST	String	是
Key	固定值：jobs	String	是
Url	请求url，固定值	String	是
Body	请求体，详情请参见 提交任务	Object	是
ContentType	固定值	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 提交任务	Object

相关链接

API 文档

词性分析
提交任务

最近更新时间：2024-12-27 16:37:06

简介

词性分析功能是由 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。
本文档提供关于词性分析的相关 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
提交任务	提交一个分词任务。

提交任务

功能说明

提交一个分词任务。

使用示例

```
function postWordsGeneralize() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'jobs'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 创建任务的 Tag：WordsGeneralize;是否必传：是
      Tag: 'WordsGeneralize',
      // 待操作的对象信息;是否必传：是
      Input: {
        // 文件路径;是否必传：是
        Object: 'test.txt',
```

```
      },
      // 操作规则;是否必传：是
      Operation: {
        // 指定分词参数;是否必传：是
        WordsGeneralize: {
          // ner 方式，支持 NerBasic 和 DL，默认值 DL;是否必传：否
          NerMethod: 'DL',
          // 分词粒度，支持 SegBasic 和 MIX，默认值 MIX;是否必传：否
          SegMethod: 'MIX',
        },
        // 透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
        // UserData: '',
        // 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;是否必传：否
        // JobLevel: '',
      },
      // 任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
      // CallBackFormat: '',
      // 任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
      // CallBackType: '',
      // 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;是否必传：否
      // CallBack: '',
      // 任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig;是否必传：否
      // CallBackMqConfig: {
      // 消息队列所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）;是否必传：是
      // MqRegion: '',
      // 消息队列使用模式，默认 Queue ：主题订阅：Topic队列服务：Queue;是否必传：是
      // MqMode: '',
      // TDMQ 主题名称;是否必传：是
      // MqName: '',
      // },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
```

```
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
postWordsGeneralize();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Key	固定值：jobs	String	是
Url	请求url，固定值	String	是
Body	请求体，详情请参见 提交任务	Object	是
ContentType	固定值	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 提交任务	Object

相关链接

API 文档

视频目标检测
提交任务

最近更新时间：2024-12-27 16:37:06

简介

提交一个视频目标检测任务功能是由 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。
本文档提供关于提交一个视频目标检测任务的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
提交任务	提交一个视频目标检测任务。

提交任务

功能说明

提交一个视频目标检测任务

使用示例

```
function postVideoTargetRec() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'jobs'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 创建任务的 Tag：VideoTargetRec;是否必传：是
      Tag: 'VideoTargetRec',
      // 操作规则;是否必传：是
      Operation: {
        // 视频目标检测模板 ID;与VideoTargetRec二选一传递
        TemplateId: 'xxxx',
```

```
        // 视频目标检测参数，同创建视频目标检测模板接口中的 Request.VideoTargetRec;与TemplateId二选一传递
        // VideoTargetRec: {
        // 是否开启人体检测，取值 true/false;是否必传：否
        // Body: '',
        // 是否开启宠物检测，取值 true/false;是否必传：否
        // Pet: '',
        // 是否开启车辆检测，取值 true/false;是否必传：否
        // Car: '',
        // },
        // 透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
        // UserData: '',
        // 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;是否必传：否
        // JobLevel: '',
      },
      // 待操作的媒体信息;是否必传：是
      Input: {
        // 媒体文件名;是否必传：否
        Object: 'ci/test.mp4',
      },
      // 任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
      // CallBackFormat: '',
      // 任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
      // CallBackType: '',
      // 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;是否必传：否
      // CallBack: '',
      // 任务回调TDMQ配置，当 CallBackType 为 TDMQ 时必填。详情请参见 CallBackMqConfig;是否必传：否
      // CallBackMqConfig: {
      // 消息队列所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）;是否必传：是
      // MqRegion: '',
      // 消息队列使用模式，默认 Queue ：主题订阅：Topic队列服务：Queue;是否必传：是
      // MqMode: '',
      // TDMQ 主题名称;是否必传：是
      // MqName: '',
      // },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
```

```
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
postVideoTargetRec();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Key	固定值：jobs	String	是
Url	请求url，固定值	String	是
Body	请求体，详情请参见 提交任务	Object	是
ContentType	固定值	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 提交任务	Object

相关参考

API 文档
功能指南

创建模板

最近更新时间：2024-12-27 16:37:06

简介

创建视频目标检测模板功能是由 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。
本文档提供关于创建视频目标检测模板的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
创建模板	创建视频目标检测模板。

创建模板

功能说明

创建视频目标检测模板。

使用示例

```
function postVideoTargetTemplete() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'template'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 模板类型：VideoTargetRec;是否必传：是
      Tag: 'VideoTargetRec',
      // 模板名称，仅支持中文、英文、数字、_、-和*，长度不超过 64;是否必传：是
      Name: 'my_videoTargetRec',
      // 视频目标检测 参数;是否必传：是
      VideoTargetRec: {
        // 是否开启人体检测，取值 true/false;是否必传：否
        Body: 'true',
```

```
        // 是否开启宠物检测，取值 true/false;是否必传：否
        Pet: 'true',
        // 是否开启车辆检测，取值 true/false;是否必传：否
        Car: 'false',
      },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
postVideoTargetTemplete();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Url	请求url，固定值	String	是
Key	固定值：template	String	是
ContentType	固定值	String	是
Body	请求体，详情请参见 创建模板	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 创建模板	Object

相关参考

API 文档
功能指南

更新模板

最近更新时间：2024-12-27 16:37:06

简介

更新视频目标检测模板功能是由 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。
本文档提供关于更新视频目标检测模板的相关 API 概览以及 SDK 示例代码。

> **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
更新模板	更新视频目标检测模板。

更新模板

功能说明

更新视频目标检测模板。

使用示例

```
function updateVideoTargetTemplete() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const templateId = 'xxxxxxx'; // 要更新的模板id
  const key = `template/${templateId}`; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 模板类型：VideoTargetRec;是否必传：是
      Tag: 'VideoTargetRec',
      // 模板名称，仅支持中文、英文、数字、_、-和*，长度不超过 64;是否必传：是
      Name: 'my_videoTargetRec2',
      // 视频目标检测 参数;是否必传：是
      VideoTargetRec: {
        // 是否开启人体检测，取值 true/false;是否必传：否
```

```
        Body: 'false',
        // 是否开启宠物检测，取值 true/false;是否必传：否
        Pet: 'false',
        // 是否开启车辆检测，取值 true/false;是否必传：否
        Car: 'true',
      },
    },
  });

  cos.request(
    {
      Method: 'PUT', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
updateVideoTargetTemplete();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：PUT	String	是
Url	请求url，固定值	String	是
Key	template/{TemplateId}	String	是
ContentType	固定值	String	是
Body	请求体，详情请参见 更新模板	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果，详情请参见 更新模板	Object

相关参考

API 文档
功能指南

视频人像分割
提交任务

最近更新时间：2024-12-27 16:37:06

简介

视频人像分割功能是由 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。
本文档提供关于视频人像分割的相关 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> COS Javascript SDK 版本需要大于等于 v1.3.2。

API	操作描述
提交任务	提交一个视频人像抠图任务。

提交任务

功能说明

提交一个视频人像抠图任务。

使用示例

```
function postSegmentVideoBody() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'jobs'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 创建任务的 Tag：SegmentVideoBody;是否必传：是
      Tag: 'SegmentVideoBody',
      // 待操作的对象信息;是否必传：是
      Input: {
        // 文件路径;是否必传：是
        Object: 'ci/test.mp4',
```

```
        },
        // 操作规则;是否必传：是
        Operation: {
          // 视频人像抠图配置;是否必传：否
          SegmentVideoBody: {
            // 抠图模式 Mask：输出alpha通道结果Foreground：输出前景视频Combination：输出抠图后的前景与自定义背景合成后的视频默认值：Mask;是否必传：否
            Mode: 'Mask',
            // 抠图类型HumanSeg：人像抠图GreenScreenSeg：绿幕抠图SolidColorSeg：纯色背景抠图默认值：HumanSeg;是否必传：否
            // SegmentType: '',
            // mode为 Foreground 时参数生效，背景颜色为红色，取值范围 [0, 255]， 默认值为 0;是否必传：否
            // BackgroundRed: '',
            // mode为 Foreground 时参数生效，背景颜色为绿色，取值范围 [0, 255]，默认值为 0;是否必传：否
            // BackgroundGreen: '',
            // mode为 Foreground 时参数生效，背景颜色为蓝色，取值范围 [0, 255]，默认值为 0;是否必传：否
            // BackgroundBlue: '',
            // 传入背景文件。mode为 Combination 时，此参数必填，背景文件需与源文件在同存储桶下;是否必传：否
            // BackgroundLogoUrl: '',
            // 调整抠图的边缘位置，取值范围为[0, 255]，默认值为 0;是否必传：否
            // BinaryThreshold: '',
            // 纯色背景抠图的背景色（红），当 SegmentType 为 SolidColorSeg 生效，取值范围为 [0, 255]，默认值为 0;是否必传：否
            // RemoveRed: '',
            // 纯色背景抠图的背景色（绿），当 SegmentType 为 SolidColorSeg 生效，取值范围为 [0, 255]，默认值为 0;是否必传：否
            // RemoveGreen: '',
            // 纯色背景抠图的背景色（蓝），当 SegmentType 为 SolidColorSeg 生效，取值范围为 [0, 255]，默认值为 0;是否必传：否
            // RemoveBlue: '',
          },
          // 结果输出配置;是否必传：是
          Output: {
            // 存储桶的地域;是否必传：是
            Region: config.Region,
            // 存储结果的存储桶;是否必传：是
            Bucket: config.Bucket,
            // 输出结果的文件名;是否必传：是
            Object: 'ci/out_test.mp4',
          },
          // 透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
          // UserData: '',
```

```
          // 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;是否必传：否
          // JobLevel: '',
        },
        // 任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
        // CallBackFormat: '',
        // 任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
        // CallBackType: '',
        // 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;是否必传：否
        // CallBack: '',
        // 任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见
CallBackMqConfig;是否必传：否
        // CallBackMqConfig: {
        // 消息队列所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk
（中国香港）;是否必传：是
        // MqRegion: '',
        // 消息队列使用模式，默认 Queue ：主题订阅：Topic队列服务：Queue;是否必传：是
        // MqMode: '',
        // TDMQ 主题名称;是否必传：是
        // MqName: '',
        // },
      },
    });

    cos.request(
      {
        Method: 'POST', // 固定值，必须
        Key: key, // 必须
        Url: url, // 请求的url，必须
        Body: body, // 请求体参数，必须
        ContentType: 'application/xml', // 固定值，必须
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
postSegmentVideoBody();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Url	请求url，固定值	String	是
Key	固定值：jobs	String	是
ContentType	固定值	String	是
Body	请求体，详情请参见 提交任务	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果，详情请参见 提交任务	Object

相关参考

API 文档

Node.js SDK 快速入门

最近更新时间：2025-03-14 19:19:32

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

- 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。
- 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 。
- 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。

使用永久密钥初始化

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.cloud.tencent.com/cam/capi 进行
查看和管理
var COS = require('cos-nodejs-sdk-v5');
var cos = new COS({
    SecretId: 'SECRETID',
    SecretKey: 'SECRETKEY'
});
```

使用临时密钥初始化

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 durationSeconds
            });
```

```
        });
    }
});
```

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，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，默认值10485760（10MB）	Number	否
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，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
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（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：

- 实例化时，传入 SecretId、SecretKey，每次需要签名都由实例内部计算。
- 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。
- 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);
}
```

> **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);
});
```

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);
});
```

内容识别
服务和队列配置
开通 AI 内容识别服务

最近更新时间：2024-12-27 16:37:06

简介

开通AI内容识别服务功能是由 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。
本文档提供关于开通AI内容识别服务的相关 API 概览以及 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
开通AI内容识别服务	本接口用于开通AI 内容识别（异步）服务并生成队列。

开通AI内容识别服务

功能说明

本接口用于开通AI 内容识别（异步）服务并生成队列。

使用示例

```
function openAIBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'ai_bucket'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
```

```
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
openAIBucket();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Url	请求url，固定值	String	是
Key	固定值：ai_bucket	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 开通AI内容识别服务	Object

相关参考

API 文档

查询 AI 内容识别服务

最近更新时间：2024-12-27 16:37:06

简介

查询AI内容识别服务功能是由 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。
本文档提供关于查询AI内容识别服务的相关 API 概览以及 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
查询AI内容识别服务	本接口用于查询已经开通 AI 内容识别（异步）服务的存储桶。

查询AI内容识别服务

功能说明

本接口用于查询已经开通AI 内容识别（异步）服务的存储桶。

使用示例

```
function getAIBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'ai_bucket'; // 固定值
  const host = `ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;

  cos.request(
    {
      Method: 'GET', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Query: {
        // 地域信息，例如 ap-shanghai、ap-beijing，若查询多个地域以“,”分隔字符串，详情
请参见 地域与域名;是否必传：否
        // regions:  '',
```

```
        // 存储桶名称，以“,”分隔，支持多个存储桶，精确搜索;是否必传：否
        // bucketNames:  '',
        // 存储桶名称前缀，前缀搜索;是否必传：否
        // bucketName:  '',
        // 第几页;是否必传：否
        // pageNumber: 1,
        // 每页个数，大于0且小于等于100的整数;是否必传：否
        // pageSize: 10,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
getAIBucket();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：GET	String	是
Url	请求url，固定值	String	是
Key	固定值：ai_bucket	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
regions	地域信息，例如 ap-shanghai、ap-beijing，若查询多个地域以“,”分隔字符串，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	Integer	否
pageSize	每页个数，大于0且小于等于100的整数	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果 详情请参见 查询 AI 内容识别服务	Object

相关链接

API 文档

关闭 AI 内容识别服务

最近更新时间：2024-12-27 16:37:06

简介

关闭AI内容识别服务功能是由 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。
本文档提供关于关闭AI内容识别服务的相关 API 概览以及 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
关闭AI内容识别服务	本接口用于关闭AI 内容识别（异步）服务并删除队列。

关闭AI内容识别服务

功能说明

本接口用于关闭AI 内容识别（异步）服务并删除队列。

使用示例

```
function closeAIBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'ai_bucket'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;

  cos.request(
    {
      Method: 'DELETE', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
```

```
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
closeAIBucket();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：DELETE	String	是
Url	请求url，固定值	String	是
Key	固定值：ai_bucket	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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

- Response	响应结果 详情请参见 关闭AI内容识别服务	Object

相关参考

API 文档

查询 AI 内容识别队列

最近更新时间：2024-12-27 16:37:06

简介

查询AI内容识别队列功能是由 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。
本文档提供关于查询AI内容识别队列的相关 API 概览以及 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
查询AI内容识别队列	本接口用于查询AI 内容识别（异步）队列。

查询AI内容识别队列

功能说明

本接口用于查询AI 内容识别（异步）队列。

使用示例

```
function getAIQueue() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'ai_queue'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;

  cos.request(
    {
      Method: 'GET', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Query: {
        // 队列 ID，以“,”符号分割字符串;是否必传：否
        // queueIds:  "",
```

```
        // Active 表示队列内的作业会被调度执行Paused 表示队列暂停，作业不再会被调度执行，
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;是否必传：否
        state: 'Active',
        // 第几页，默认值1;是否必传：否
        // pageNumber: 0,
        // 每页个数，默认值10;是否必传：否
        // pageSize: 0,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
getAIQueue();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：GET	String	是
Url	请求url，固定值	String	是
Key	固定值：ai_queue	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
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调度执行Paused 表示队列暂停，作业不再会被调度执行，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
pageNumber	第几页，默认值1。	Integer	否

pageSize	每页个数，默认值10。	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果 详情请参见 查询AI内容识别队列	Object

相关参考

API 文档

更新 AI 内容识别队列

最近更新时间：2024-12-27 16:37:06

简介

更新AI内容识别队列功能是由 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。
本文档提供关于更新AI内容识别队列的相关 API 概览以及 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
更新AI内容识别队列	本接口用于更新 AI 内容识别（异步）的队列。

更新AI内容识别队列

功能说明

本接口用于更新 AI 内容识别（异步）的队列。

使用示例

```
function updateAIQueue() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const queueId = 'xxxx'; // 队列id
  const key = `ai_queue/${queueId}`; // 固定格式
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 队列名称，仅支持中文、英文、数字、_、-和*，长度不超过 128;是否必传：是
      Name: 'my_queue',
      // Active 表示队列内的作业会被调度执行Paused 表示队列暂停，作业不再会被调度执行，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;是否必传：是
      State: 'Active',
      // 回调配置;是否必传：是
      NotifyConfig: {
```

```
        // 回调开关OffOn;是否必传：否
        State: 'On',
        // 回调事件TaskFinish：任务完成WorkflowFinish：工作流完成;是否必传：否
        Event: 'TaskFinish',
        // 回调格式XMLJSON;是否必传：否
        // ResultFormat: '',
        // 回调类型UrlTDMQ;是否必传：否
        Type: 'Url',
        // 回调地址，不能为内网地址。;是否必传：否
        Url: 'http://example.com',
        // TDMQ 使用模式Topic：主题订阅Queue：队列服务;是否必传：否
        // MqMode: '',
        // TDMQ 所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、
hk（中国香港）;是否必传：否
        // MqRegion: '',
        // TDMQ 主题名称;是否必传：否
        // MqName: '',
      },
    },
  });

  cos.request(
    {
      Method: 'PUT', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
updateAIQueue();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：PUT	String	是
Key	ai_queue/{queueId}	String	是
Url	请求url，固定值	String	是
Body	请求体详情请参见 更新AI内容识别队列	Object	是
ContentType	固定值	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 更新AI内容识别队列	Object

相关参考

API 文档

图片上色

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图片上色的 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> - COS Node.js SDK 版本需要大于等于 v2.11.2。
> - 待处理的图片，原图大小不超过5MB。

API	说明
图片上色	腾讯云数据万象通过 AIImageColoring 接口对黑白图像进行上色。

上传时处理

下面示例展示了如何在上传图片时自动实现图片上色。
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

```
function handleFileInUploading() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const filePath = 'temp-file-to-upload'; // 本地文件路径
  cos.putObject(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: 'folder/document.jpg', // 对象文件名，例如：folder/document.jpg
      Body: fs.createReadStream(filePath), // 上传文件对象，必填
      ContentLength: fs.statSync(filePath).size, // 上传文件对象大小，当上传Body为文件流且需要正确展示上传进度时必填
      Headers: {
        // 通过 AIImageColoring 接口使用图片上色功能
        'Pic-Operations': JSON.stringify({
          is_pic_info: 1,
          rules: [{ fileid: 'test.jpg', rule: 'ci-process=AIImageColoring' }]
        }),
      },
    },
    function (err, data) {
      if (err) {
```

```
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      }
    );
}
handleFileInUploading();
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容	Buffer/String	是
ContentLength	上传文件内容长度，需要展示进度时必填	Number	否
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AIImageColoring	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-	请求返回的头部信息	Obje

headers		ct
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
– Response	响应结果，详情请参见 图片上色	Object

对云上数据进行图片上色

下面示例展示了如何在对已存储在 COS 的图片进行图片上色操作，并将结果存入到 COS。

示例代码

```
function handleFileInBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Method: 'POST',  // 固定值
      Action: 'image_process',  // 固定值
      Headers: {
        // 通过 AIImageColoring 接口使用图片上色功能
        'Pic-Operations': JSON.stringify({
          rules: [{ fileid: 'test.jpg', rule: 'ci-process=AIImageColoring' }]
        }),
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
```

```
  );
}
handleFileInBucket();
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule固定为 ci-process=AIImageColoring	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-	请求返回的头部信息	Obje

headers		ct
– Response	响应结果，详情请参见 图片上色	Object

下载时使用图片上色

下面示例展示了如何在下载图片时实现图片上色。

示例代码

```
function getObject() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Key: '1.jpg',  // 对象文件名，例如：folder/document.jpg。
      Query: {
        'ci-process': 'AIImageColoring',
        // 'detect-url': '', // 待处理图片url，与Key二选一传递
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
        const localPath = 'xxx'; // 填写要写入的本地文件路径
        fs.writeFileSync(localPath, data.Body);  // 将图片内容保存本地路径
      }
    }
  );
}
getObject();
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必

			选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	否
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必选
ci-process	数据万象处理能力，图片上色固定为 AIImageColoring。	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。 detect-url 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-statusC	请求返回的 HTTP 状态码，例如200、403、404等	Number

ode		
– headers	请求返回的头部信息	Object
– Response	响应结果，详情请参见 图片上色	Object

相关链接

API 文档
功能指南

图像超分

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图像超分的 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> - COS Node.js SDK 版本需要大于等于 v2.11.2。
> - 待处理的图片，原图大小不超过5MB，分辨率不得超过1920px * 1080px。

API	说明
图像超分	腾讯云数据万象通过 AISuperResolution 接口对图像进行超分辨率处理，当前默认超分为宽高的2倍。

上传时处理

下面示例展示了如何在上传图片时自动实现图像超分。
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

```
function handleFileInUploading() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const filePath = 'temp-file-to-upload'; // 本地文件路径
  cos.putObject(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: 'folder/document.jpg', // 对象文件名，例如：folder/document.jpg
      Body: fs.createReadStream(filePath), // 上传文件对象，必填
      ContentLength: fs.statSync(filePath).size, // 上传文件对象大小，当上传Body为
文件流且需要正确展示上传进度时必填
      Headers: {
        // 通过 AISuperResolution 接口使用图像超分功能
        'Pic-Operations': JSON.stringify({
          is_pic_info: 1,
          rules: [{ fileid: 'test.jpg', rule: 'ci-process=AISuperResolution'
}]
        }),
      },
    },
```

```
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
handleFileInUploading();
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容	Buffer/String	是
ContentLength	上传文件内容长度，需要展示进度时必填	Number	否
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AISuperResolution	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 图片超分	Object

对云上数据进行图像超分

下面示例展示了如何对已存储在 COS 的图片进行图像超分操作，并将结果存入到 COS。

示例代码

```
function handleFileInBucket() {
  // sdk引入以及初始化请参考: https://cloud.tencent.com/document/product/436/8629
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: '1/jpg', // 对象文件名，例如: folder/document.jpg。
      Method: 'POST',  // 固定值
      Action: 'image_process',  // 固定值
      Headers: {
        // 通过 AISuperResolution 接口使用图像超分功能
        'Pic-Operations': JSON.stringify({
          rules: [{ fileid: 'test.jpg', rule: 'ci-process=AISuperResolution'
}]
        }),
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
```

```
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      }
    );
}
handleFileInBucket();
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AISuperResolution	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图片超分	Object

下载时使用图像超分

下面示例展示了如何下载图片时实现图像超分。

示例代码

```
function getObject() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Query: {
        'ci-process': 'AISuperResolution',  // 固定值
        // 'detect-url': '', // 待处理图片url，与Key二选一传递
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
        const localPath = 'xxx'; // 填写要写入的本地文件路径
        fs.writeFileSync(localPath, data.Body);  // 将图片内容保存本地路径
      }
    }
  );
}
getObject();
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg。	String	否
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必选
ci-process	数据万象处理能力，图像超分固定为 AISuperResolution。	String	是
detect-url	处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。detect-url 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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	Obje

		ct
– 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
– Response	响应结果，详情请参见 图像超分	Object

相关链接

API 文档
功能指南

图像增强

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图像增强的 API 概览以及 SDK 示例代码。

> ⚠ 注意：
> - COS Node.js SDK 版本需要大于等于 v2.11.2。
> - 待处理的图片，原图大小不超过5MB。

API	说明
图像增强	腾讯云数据万象通过 AIEnhanceImage 接口对图像进行增强处理。

上传时处理

下面示例展示了如何在上传图片时自动实现图像增强。
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

```
function handleFileInUploading() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const filePath = 'temp-file-to-upload'; // 本地文件路径
  cos.putObject(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: 'folder/document.jpg', // 对象文件名，例如：folder/document.jpg
      Body: fs.createReadStream(filePath), // 上传文件对象，必填
      ContentLength: fs.statSync(filePath).size, // 上传文件对象大小，当上传Body为文件流且需要正确展示上传进度时必填
      Headers: {
        // 通过 AIEnhanceImage 接口使用图像增强功能：指定图片锐化强度值 5，去噪强度值 3
        'Pic-Operations': JSON.stringify({
          is_pic_info: 1,
          rules: [{ fileid: 'test.jpg', rule: 'ci-process=AIEnhanceImage' }]
        }),
      },
    },
    function (err, data) {
      if (err) {
```

```
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      }
    );
}
handleFileInUploading();
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容	Buffer/String	是
ContentLength	上传文件内容长度，需要展示进度时必填	Number	否
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AIEnhanceImage	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-	请求返回的头部信息	Obje

headers		ct
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
– Response	响应结果，详情请参见 图像增强	Object

对云上数据进行图像增强

下面示例展示了如何在对已存储在 COS 的图片进行图像增强操作，并将结果存入到 COS。

示例代码

```
function handleFileInBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Method: 'POST',  // 固定值
      Action: 'image_process',  // 固定值
      Headers: {
        // 通过 AIEnhanceImage 接口使用图像增强功能：指定图片锐化强度值 5，去噪强度值 3
        'Pic-Operations': JSON.stringify({
          rules: [{ fileid: 'test.jpg', rule: 'ci-process=AIEnhanceImage' }]
        }),
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
```

```
  );
}
handleFileInBucket();
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AIEnhanceImage	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-	请求返回的头部信息	Obje

headers		ct
- Response	响应结果，详情请参见 图像增强	Object

下载时使用图像增强

下面示例展示了如何在下载图片时实现图像增强。

示例代码

```
function getObject() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Query: {
        'ci-process': 'AIEnhanceImage',  // 固定值
        denoise: 5, // 非必需，去噪强度值，取值范围为 0 - 5 之间的整数，值为 0 时不进行去噪操作，默认值为3
        sharpen: 5, // 非必需，锐化强度值，取值范围为 0 - 5 之间的整数，值为 0 时不进行锐化操作，默认值为3
        // detect-url:'http://www.example.com/abc.jpg'  // 非必须，可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，detect-url 需要进行 UrlEncode
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
        const localPath = 'xxx'; // 填写要写入的本地文件路径
        fs.writeFileSync(localPath, data.Body);  // 将图片内容保存本地路径
      }
    }
  );
}
getObject();
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	否
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必选
ci-process	数据万象处理能力，图像增强固定为 AIEnhanceImage。	String	是
denoise	去噪强度值，取值范围为 0 - 5 之间的整数，值为 0 时不进行去噪操作，默认值为 3。	Integer	否
sharpen	锐化强度值，取值范围为 0 - 5 之间的整数，值为 0 时不进行锐化操作，默认值为 3。	Integer	否
detect-url	处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。 detect-url 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-statusC	请求返回的 HTTP 状态码，例如200、403、404等	Number

ode		
– 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
– Response	响应结果，详情请参见 图像增强	Object

相关链接

API 文档
功能指南

图像智能裁剪

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图像智能裁剪的 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> - COS Node.js SDK 版本需要使用 v2.11.2及以上版本。
> - 待处理的图片，原图大小不超过5MB。

API	说明
图像智能裁剪	腾讯云数据万象通过 AIImageCrop 接口对图像进行智能裁剪。

上传时处理

下面示例展示了如何在上传图片时自动实现图像智能裁剪。
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

```
function handleFileInUploading() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const filePath = 'temp-file-to-upload'; // 本地文件路径
  cos.putObject(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: 'folder/document.jpg', // 对象文件名，例如：folder/document.jpg
      Body: fs.createReadStream(filePath), // 上传文件对象，必填
      ContentLength: fs.statSync(filePath).size, // 上传文件对象大小，当上传Body为
文件流且需要正确展示上传进度时必填
      Headers: {
        // 通过 AIImageCrop 接口使用图像智能裁剪功能：指定图片剪区域的宽度 1，高度 2
        'Pic-Operations': JSON.stringify({
          is_pic_info: 1,
          rules: [{ fileid: 'test.jpg', rule: 'ci-
process=AIImageCrop&width=1&height=2' }]
        }),
      },
    },
```

```
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
handleFileInUploading();
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容	Buffer/String	是
ContentLength	上传文件内容长度，需要展示进度时必填	Number	否
Pic-Operations	详情请参见 图像智能裁剪处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图像智能裁剪	Object

对云上数据进行图像智能裁剪

下面示例展示了如何对已存储在 COS 的图片进行图像智能裁剪操作，并将结果存入到 COS。

示例代码

```
function handleFileInBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Method: 'POST',  // 固定值
      Action: 'image_process',  // 固定值
      Headers: {
        // 通过 AIImageCrop 接口使用图像智能裁剪功能：指定图片剪区域的宽度 1，高度 2
        'Pic-Operations': JSON.stringify({
          rules: [{ fileid: 'test.jpg', rule: 'ci-
process=AIImageCrop&width=1&height=2' }]
        }),
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
```

```
            console.log(data);
        }
      }
    );
}
handleFileInBucket();
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是
Pic-Operations	详情请参见 图像智能裁剪处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-status	请求返回的 HTTP 状态码，例如200、403、404等	Number

Code		
– headers	请求返回的头部信息	Object
– Response	响应结果，详情请参见 图像智能裁剪	Object

下载时使用图像智能裁剪

下面示例展示了如何在下载图片时实现图像智能裁剪。

示例代码

```
function getObject() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Query: {
        'ci-process': 'AIImageCrop',  // 固定值
        width: 5, // 必需，需要裁剪区域的宽度
        height: 5, // 必需，需要裁剪区域的高度
        //fixed:1, // 非必需，是否严格按照 width 和 height 的值进行输出
        //'detect-url': 'http://www.example.com/abc.jpg'  // 非必须，可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
        const localPath = 'xxx'; // 填写要写入的本地文件路径
        fs.writeFileSync(localPath, data.Body);  // 将图片内容保存本地路径
      }
    }
  );
```

```
}
getObject();
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	否
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必选
ci-process	数据万象处理能力，智能裁剪固定为 AIImageCrop	String	是
width	需要裁剪区域的宽度，与 height 共同组成所需裁剪的图片宽高比例；输入数字请大于0、小于图片宽度的像素值	Integer	是
height	需要裁剪区域的高度，与 width 共同组成所需裁剪的图片宽高比例；输入数字请大于0、小于图片高度的像素值；width : height 建议取值在[1, 2.5]之间，超过这个范围可能会影响效果	Integer	是
fixed	是否严格按照 width 和 height 的值进行输出 • 取值为0时，宽高比例（width : height）会简化为最简分数，即如果 width 输入10、height 输入20，会简化为1：2 • 取值为1时，输出图片的宽度等于 width，高度等于 height 默认值为0	Integer	否
detect-url	处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。 detect-url 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果，详情请参见 图像智能裁剪	Object

相关链接

- API 文档
- 功能指南

图像修复

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图像修复的 API 概览以及 SDK 示例代码。

> **注意：**
> - COS Node.js SDK 版本需要大于等于 v2.11.2。
> - 待处理的图片，原图大小不超过32MB。
> - 宽高不超过50000像素且总像素不超过2.5亿像素，处理结果图宽高设置不超过9999像素。

API	说明
图像修复	腾讯云数据万象通过 ImageRepair 接口检测图片中的水印并将其擦除。

上传时处理

下面示例展示了如何在上传图片时自动实现图像修复。
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

```
function handleFileInUploading() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const filePath = 'temp-file-to-upload'; // 本地文件路径
  cos.putObject(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: 'folder/document.jpg', // 对象文件名，例如：folder/document.jpg
      Body: fs.createReadStream(filePath), // 上传文件对象，必填
      ContentLength: fs.statSync(filePath).size, // 上传文件对象大小，当上传Body为
文件流且需要正确展示上传进度时必填
      Headers: {
        // 通过 ImageRepair 接口使用图像修复功能：指定遮罩图片地址 URL 安全的 Base64
编码：aHR0c...XXX
        'Pic-Operations': JSON.stringify({
          is_pic_info: 1,
          rules: [{ fileid: 'test.jpg', rule: 'ci-
process=ImageRepair&MaskPic=aHR0c...XXX' }]
        }),
```

```
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
handleFileInUploading();
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg。	String	是
Body	文件内容	Buffer/String	是
Content Length	上传文件内容长度，需要展示进度时必填	Number	否
Pic-Operations	详情请参见 图像修复处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果 详情请参见 图像修复	Object

对云上数据进行图像修复

下面示例展示了如何对已存储在 COS 的图片进行图像修复操作，并将结果存入到 COS。

示例代码

```
function handleFileInBucket() {
  // sdk引入以及初始化请参考: https://cloud.tencent.com/document/product/436/8629
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Key: '1/jpg', // 对象文件名，例如: folder/document.jpg。
      Method: 'POST',  // 固定值
      Action: 'image_process',  // 固定值
      Headers: {
        // 通过 ImageRepair 接口使用图像修复功能：指定遮罩图片地址 URL 安全的 Base64 编码: aHR0c...XXX
        'Pic-Operations': JSON.stringify({
          rules: [{ fileid: 'test.jpg', rule: 'ci-process=ImageRepair&MaskPic=aHR0c...XXX' }]
        }),
      },
    },
    function (err, data) {
      if (err) {
```

```
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
handleFileInBucket();
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg。	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是
Pic-Operations	详情请参见 图像修复处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 图像修复	Object

下载时使用图像修复

下面示例展示了如何在下载图片时实现图像修复。

示例代码

```
function getObject() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  cos.request(
    {
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Query: {
        'ci-process': 'ImageRepair',  // 固定值
        MaskPic: 'aHR0c...XXX' /*非必需，遮罩（白色区域为需要去除的水印位置）图片地址，需要经过 URL 安全的 Base64 编码*/,
        //MaskPoly:'[[[608, 794], [1024, 794]],[[1295, 62],[1295, 30]]]' /* 非必需，顺时针输入多边形的每个点的坐标,MaskPoly 同时与 MaskPic 填写时，优先采用 MaskPic 的值*/,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
```

```
  );
}
getObject();
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg。	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必选
ci-process	数据万象处理能力，图像修复固定为 ImageRepair。	String	是
MaskPic	遮罩（白色区域为需要去除的水印位置）图片地址，需要经过 URL 安全的 Base64 编码。例如，遮罩图片为 `http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/shuiyin_2.png`，则该处编码后的字符串为 `aHR0cDovL2V4YW1wbGVidWNrZXQtMTI1MDAwMDAwMC5jb3MuYXAtc2hhbmdoYWkubXlxY2xvdWQuY29tL3NodWl5aW5fMi5wbmc`	String	否
MaskPoly	例如：[[[608, 794], [1024, 794], [1024, 842], [608, 842]],[[1295, 62], [1295, 30], [1597, 32],[1597, 64]]]，顺时针输入多边形的每个点的坐标,每个多边形: [[x1, y1], [x2, y2]...], 形式为三维矩阵（多个多边形：[多边形1, 多边形2]）或二维矩阵（单个多边形），且需要经过 URL 安全的 Base64 编码。MaskPoly 同时与 MaskPic 填写时，优先采用 MaskPic 的值。	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图像修复	Object

相关链接

API 文档
功能指南

图片二维码

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于图片二维码相关的 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> COS Node.js SDK 版本需要大于等于 v2.11.2。

API	说明
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。
二维码生成	二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码。

二维码识别

功能说明

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。

请求示例1：上传时识别

```
function identifyQrcode_put() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const filePath = 'temp-file-to-upload'; // 本地文件路径
  const 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 */
  };
  // 通过input组件选择文件得到 file 代码省略
  cos.putObject({
      Bucket: config.Bucket,
      Region: config.Region,
      Key: '上传二维码.png', // 对象文件名，例如：folder/document.jpg
      Body: fs.createReadStream(filePath), // 上传文件对象，必填
      ContentLength: fs.statSync(filePath).size, // 上传文件对象大小，当上传Body为
文件流且需要正确展示上传进度时必填
      Headers: {
        'Pic-Operations': JSON.stringify({
```

```
            is_pic_info: 1,
            rules: [{ fileid: 'test.jpg', rule: 'QRcode/cover/0' }]
          }),
      },
      onProgress: function (progressData) {
          console.log('onProgress', JSON.stringify(progressData));
      },
  }, function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
  });
}
identifyQrcode_put();
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg。	String	是
Body	文件内容	Buffer/String	是
Content Length	上传文件内容长度，需要展示进度时必填	Number	否
Pic-Operations	详情请参见 图像修复处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 二维码识别	Object

请求示例2：下载时识别

```
function identifyQrcode_get() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  cos.request({
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Query: {
        'ci-process': 'QRcode'  // 固定值
      },
  },
  function(err, data){
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
  });
}
```

```
identifyQrcode_get();
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg。	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必填
ci-process	万象处理能力，二维码识别固定为 QRcode	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 二维码识别	Object

二维码生成

功能说明

二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码。

示例代码

```
function generateQrcode() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  cos.request({
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Query: {
        'ci-process': 'qrcode-generate', /* 必须，对象存储处理能力，二维码生成参数为qrcode-generate */
        'qrcode-content': '二维码文案', /* 必须，可识别的二维码文本信息     */
        // mode: 0, /* 非必须，生成的二维码类型，可选值：0或1。0为二维码，1为条形码，默认值为0  */
        width: 200, /* 必须，指定生成的二维码或条形码的宽度，高度会进行等比压缩   */
      },
  },
  function(err, data){
    if (!err) {
      // 获得二维码 base64
      var imgBase64 = data.Response.ResultImage;
      console.log(imgBase64);
    }
  });
}
generateQrcode();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，二维码生成参数为 qrcode-generate	String	是
qrcode-content	可识别的二维码文本信息	String	是
mode	生成的二维码类型，可选值：0或1。0为二维码，1为条形码，默认值为0	Integer	否
width	指定生成的二维码或条形码的宽度，高度会进行等比压缩	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 图片二维码生成	Object

相关链接

API 文档

图片标签

最近更新时间：2025-06-09 19:23:32

简介

本文档提供关于图片标签的相关 API 概览以及 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
图片标签	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。

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。

使用示例

```
function detectLabel() {
  // sdk 引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/dog.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 固定值detect-label;是否必传：是
        'ci-process': 'detect-label',
        // 本次调用支持的识别场景，可选值如下：web，针对网络图片优化；camera，针对手机摄像头拍摄图片优化；album，针对手机相册、网盘产品优化；news，针对新闻、资讯、广电等行业优化；如果不传此参数，则默认为camera。支持多场景（scenes）一起检测，以，分隔。例如，使用scenes=web，camera 即对一张图片使用两个模型同时检测，输出两套识别结果。;是否必传：否
        // scenes: '',
```

```
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url
时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再
填写 ObjectKey detect-url 示例：http://www.example.com/abc.jpg ，需要进行
UrlEncode，处理后为http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg;是否必
传：否
        // 'detect-url': '',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
detectLabel();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填

ci-process	固定值 detect-label	String	是
scenes	本次调用支持的识别场景，如果不传此参数，则默认为 camera。可选值如下： • web，针对网络图片优化； • camera，针对手机摄像头拍摄图片优化； • album，针对手机相册、网盘产品优化； • news，针对新闻、资讯、广电等行业优化； 支持多场景（scenes）一起检测，以，分隔。例如，使用 scenes=web，camera 即对一张图片使用两个模型同时检测，输出两套识别结果。	String	否
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 Key 。 detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Respon	响应结果，详情请参见 图片标签	Object

se		

相关链接

- API 文档
- 功能指南

图片质量评分

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于数据万象图片质量评分的 API 概览以及 SDK 示例代码。

注意：

- COS Node.js SDK 版本需要大于等于 v2.11.2。
- 图片支持格式：PNG、JPG、JPEG。
- 图片大小：所下载图片经 Base64 编码后不超过 4MB。
- 图片像素：建议大于 50*50 像素，否则影响识别效果。
- 长宽比：建议长边与短边的比例小于 5:1。

API	说明
图片质量评分	图片质量评分功能为同步请求方式，您可以通过本接口对图片文件进行检测，从多方面评估，最终给出综合可观的清晰度评分和主观的美观度评分。

图片质量评分

功能说明

图片质量评分功能为同步请求方式，您可以通过本接口对图片文件进行检测，从多方面评估，最终给出综合可观的清晰度评分和主观的美观度评分。该接口属于 GET 请求。

使用示例

```
function assessQuality() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/dog.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，图像质量检测固定为 AssessQuality。;是否必传：是
```

```
        'ci-process': 'AssessQuality',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
assessQuality();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，必须，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，图像质量检测固定为 AssessQuality。	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果 详情请参见 图片质量评分	Object

相关链接

API 文档
功能指南

图片文字识别（OCR）

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于图片文字识别(OCR)的相关 API 概览以及 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
图片文字识别(OCR)	通用文字识别功能（Optical Character Recognition，OCR）基于行业前沿的深度学习技术，将图片上的文字内容，智能识别为可编辑的文本，可应用于随手拍扫描、纸质文档电子化、电商广告审核等多种场景，大幅提升信息处理效率。

图片文字识别(OCR)

功能说明

通用文字识别功能（Optical Character Recognition，OCR）基于行业前沿的深度学习技术，将图片上的文字内容，智能识别为可编辑的文本，可应用于随手拍扫描、纸质文档电子化、电商广告审核等多种场景，大幅提升信息处理效率。

使用示例

```
function cOSOCR() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/0.jpg', // 对象文件名，例如：folder/document.jpg；与detect-url二选一
传递
      Query: {
        // 数据万象处理能力，图片文字识别固定为OCR;是否必传：是
        'ci-process': 'OCR',
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url
时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再
填写 ObjectKey detect-url 示例：http://www.example.com/abc.jpg ，需要进行
```

```
UrlEncode，处理后为http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg;是否必传：否
        // 'detect-url': '',
        // ocr的识别类型，有效值为general，accurate，efficient，fast，handwriting。general表示通用印刷体识别；accurate表示印刷体高精度版；efficient表示印刷体精简版；fast表示印刷体高速版；handwriting表示手写体识别。默认值为general。;是否必传：否
        type: 'general',
        // type值为general时有效，表示识别语言类型。支持自动识别语言类型，同时支持自选语言种类，默认中英文混合(zh)，各种语言均支持与英文混合的文字识别。可选值：zh：中英混合zh_rare：支持英文、数字、中文生僻字、繁体字，特殊符号等auto：自动mix：混合语种jap：日语kor：韩语spa：西班牙语fre：法语ger：德语por：葡萄牙语vie：越语may：马来语rus：俄语ita：意大利语hol：荷兰语swe：瑞典语fin：芬兰语dan：丹麦语nor：挪威语hun：匈牙利语tha：泰语hi：印地语ara：阿拉伯语;是否必传：否
        'language-type': 'zh',
        // type值为general，fast时有效，表示是否开启PDF识别，有效值为true和false，默认值为false，开启后可同时支持图片和PDF的识别。;是否必传：否
        ispdf: false,
        // type值为general，fast时有效，表示需要识别的PDF页面的对应页码，仅支持PDF单页识别，当上传文件为PDF且ispdf参数值为true时有效，默认值为1。;是否必传：否
        // 'pdf-pagenumber': 0,
        // type值为general，accurate时有效，表示识别后是否需要返回单字信息，有效值为true和false，默认为false;是否必传：否
        isword: false,
        // type值为handwriting时有效，表示是否开启单字的四点定位坐标输出，有效值为true和false，默认值为false。;是否必传：否
        'enable-word-polygon': false,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
cOSOCR();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，图片文字识别固定为OCR	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`	String	否
type	OCR的识别类型，默认值为general。有效值有： • general：表示通用印刷体识别。 • accurate：表示印刷体高精度版。 • efficient：表示印刷体精简版。 • fast：表示印刷体高速版。 • handwriting：表示手写体识别。	String	否
languageType	type值为general时有效，表示识别语言类型。支持自动识别语言类型，同时支持自选语言种类，默认中英文混合(zh)。 各种语言均支持与英文混合的文字识别。可选值有： • zh：中英混合 • zh_rare：支持英文、数字、中文生僻字、繁体字，特殊符号等 • auto：自动 • mix：混合语种 • jap：日语	String	否

	• kor：韩语 • spa：西班牙语 • fre：法语 • ger：德语 • por：葡萄牙语 • vie：越语 • may：马来语 • rus：俄语 • ita：意大利语 • hol：荷兰语 • swe：瑞典语 • fin：芬兰语 • dan：丹麦语 • nor：挪威语 • hun：匈牙利语 • tha：泰语 • hi：印地语 • ara：阿拉伯语		
ispdf	type值为general，fast时有效，表示是否开启PDF识别，有效值为true和false，默认值为false，开启后可同时支持图片和PDF的识别。	boolean	否
pdfPagenumber	type值为general，fast时有效，表示需要识别的PDF页面的对应页码，仅支持PDF单页识别，当上传文件为PDF且ispdf参数值为true时有效，默认值为1。	Integer	否
isword	type值为general，accurate时有效，表示识别后是否需要返回单字信息，有效值为true和false，默认为false。	boolean	否
enableWordPolygon	type值为handwriting时有效，表示是否开启单字的四点定位坐标输出，有效值为true和false，默认值为false。	boolean	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图片文字识别(OCR)	Object

相关链接

API 文档
功能指南

实时文字翻译

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于实时文字翻译的相关 API 概览以及 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
实时文字翻译	腾讯云数据万象通过 AutoTranslationBlock 接口对文字块进行翻译，请求时需要携带签名。

实时文字翻译

功能说明

腾讯云数据万象通过 AutoTranslationBlock 接口对文字块进行翻译，请求时需要携带签名。

使用示例

```
function autoTranslationBlock() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Query: {
        // 数据万象处理能力，文本块翻译固定为 AutoTranslationBlock。;是否必传：是
        'ci-process': 'AutoTranslationBlock',
        // 待翻译的文本;是否必传：是
        InputText: '我是张三',
        // 输入语言，如 "zh";是否必传：是
        SourceLang: 'zh',
        // 输出语言，如 "en";是否必传：是
        TargetLang: 'en',
        // 文本所属业务领域，如：  "ecommerce", //缺省值为 general;是否必传：否
        // TextDomain: '',
```

```
          // 文本类型，如: "title", //缺省值为 sentence;是否必传：否
          // TextStyle: '',
        },
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      }
    );
}
autoTranslationBlock();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，文本块翻译固定为 AutoTranslationBlock。	String	是
InputText	待翻译的文本	String	是
SourceLang	输入语言，如 "zh"	String	是

TargetLang	输出语言，如 "en"	String	是
TextDomain	文本所属业务领域，如: "ecommerce", 缺省值为 general	String	否
TextStyle	文本类型，如: "title", 缺省值为 sentence	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果，详情请参见 实时文字翻译	Object

相关链接

API 文档

人脸识别
人脸检测

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于人脸检测的相关 API 概览以及 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
人脸检测	人脸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人脸位置。该接口属于 GET 请求。

人脸检测

功能说明

人脸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人脸位置。该接口属于 GET 请求。

使用示例

```
function aIDetectFace() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/1.jpg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，人脸特效固定为 DetectFace。;是否必传：是
        'ci-process': 'DetectFace',
        // 最多处理的人脸数目。默认值为1（仅检测图片中面积最大的那张人脸），最大值为120。此参数用于控制处理待检测图片中的人脸个数，值越小，处理速度越快。;是否必传：否
        'max-face-num': 1,
      },
    },
```

```
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
aIDetectFace();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg。	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，人脸特效固定为 DetectFace。	String	是
max-face-num	最多处理的人脸数目。默认值为1（仅检测图片中面积最大的那张人脸），最大值为120。此参数用于控制处理待检测图片中的人脸个数，值越小，处理速度越快。	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果 详情请参见 人脸检测	Object

相关链接

API 文档

人脸特效

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于人脸特效的相关 API 概览以及 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
人脸特效	人脸特效，提供人脸美颜、人像变换、人像分割功能。

人脸特效

功能说明

人脸特效，提供人脸美颜、人像变换、人像分割功能。

使用示例

```
function aIFaceEffect() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/1.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 万象处理能力，人脸特效固定为face-effect;是否必传：是
        'ci-process': 'face-effect',
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey detect-url 示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后为http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg。;是否必传：否
        // 'detect-url': '',
```

```
        // 人脸特效类型，人脸美颜：face-beautify；人脸性别转换：face-gender-
transformation；人脸年龄变化：face-age-transformation；人像分割：face-segmentation;
是否必传：是
        type: 'face-beautify',
        // type为face-beautify时生效，美白程度，取值范围[0,100]。0不美白，100代表最高
程度。默认值30;是否必传：否
        whitening: 50,
        // type为face-beautify时生效，磨皮程度，取值范围[0,100]。0不磨皮，100代表最高
程度。默认值10;是否必传：否
        smoothing: 50,
        // type为face-beautify时生效，瘦脸程度，取值范围[0,100]。0不瘦脸，100代表最高
程度。默认值70;是否必传：否
        faceLifting: 50,
        // type为face-beautify时生效，大眼程度，取值范围[0,100]。0不大眼，100代表最高
程度。默认值70;是否必传：否
        eyeEnlarging: 50,
        // type为face-gender-transformation时生效，选择转换方向，0：男变女，1：女变
男。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。;是否必传：否
        // gender: 0,
        // type为face-age-transformation时生效，变化到的人脸年龄,[10,80]。无默认值，
为必选项。限制：仅对图片中面积最大的人脸进行转换。;是否必传：否
        // age: 0,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
aIFaceEffect();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	万象处理能力，人脸特效固定为 face-effect。	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg` 。	String	否
type	人脸特效类型，可选值有： • face-beautify：人脸美颜； • face-gender-transformation：人脸性别转换 • face-age-transformation：人脸年龄变化 • face-segmentation：人像分割	String	是
whitening	type 为 face-beautify 时生效，美白程度，取值范围[0,100]。0不美白，100代表最高程度。默认值30	Integer	否
smoothing	type 为 face-beautify 时生效，磨皮程度，取值范围[0,100]。0不磨皮，100代表最高程度。默认值10	Integer	否
faceLifting	type 为 face-beautify 时生效，瘦脸程度，取值范围[0,100]。0不瘦脸，100代表最高程度。默认值70	Integer	否
eyeEnlarging	type 为 face-beautify 时生效，大眼程度，取值范围[0,100]。0不大眼，100代表最高程度。默认值70	Integer	否
gender	type 为 face-gender-transformation 时生效，选择转换方向，0：男变女，1：女变男。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。	Integer	否

age	type 为 face-age-transformation 时生效，变化到的人脸年龄，[10,80]。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 人脸特效	Object

相关链接

API 文档
功能指南

人体识别

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于人体识别的相关 API 概览以及 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
人体识别	腾讯云数据万象通过 AIBodyRecognition 接口识别并输出画面中人体，输出其位置（矩形框）和置信度。图片人体识别请求包属于 GET 请求，请求时需要携带签名。

人体识别

功能说明

腾讯云数据万象通过 AIBodyRecognition 接口识别并输出画面中人体，输出其位置（矩形框）和置信度。图片人体识别请求包属于 GET 请求，请求时需要携带签名。

使用示例

```
function aIBodyRecognition() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/1.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，人体识别固定为AIBodyRecognition;是否必传：是
        'ci-process': 'AIBodyRecognition',
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 Key。detect-url 示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后为 http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg;是否必传：否
        // 'detect-url': '',
```

```
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
aIBodyRecognition();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，人体识别固定为 AIBodyRecognition	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 Key。	String	否

	detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`		

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 人体识别	Object

相关链接

API 文档

宠物识别

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于宠物识别的相关 API 概览以及 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
宠物识别	腾讯云数据万象通过 detect-pet 接口识别并输出画面中宠物，输出其位置（矩形框）和置信度。图片宠物识别请求包属于 GET 请求，请求时需要携带签名。

宠物识别

功能说明

腾讯云数据万象通过 detect-pet 接口识别并输出画面中宠物，输出其位置（矩形框）和置信度。图片宠物识别请求包属于 GET 请求，请求时需要携带签名。

使用示例

```
function aIDetectPet() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'dog.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，宠物识别固定为 detect-pet;是否必传：是
        'ci-process': 'detect-pet',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
```

```
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
aIDetectPet();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，宠物识别固定为 detect-pet	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果，详情请参见 宠物识别	Object

相关参考

API 文档

汽车识别

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于车辆车牌识别的相关 API 概览以及 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
车辆车牌识别	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。

车辆车牌识别

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。

使用示例

```
function aIDetectCar() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'car.png', // 对象文件名，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，车辆识别固定为 DetectCar;是否必传：是
        'ci-process': 'DetectCar',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
```

```
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
aIDetectCar();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，车辆识别固定为 DetectCar	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 车辆车牌识别	Object

相关链接

API 文档
功能指南

卡证识别

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于卡证识别的相关 API 概览以及 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
卡证识别	本接口支持中国大陆居民二代身份证正面（暂不支持背面）、驾驶证主页（暂不支持副页）所有字段的自动定位，暂不支持文本识别，用于对特定字段的抹除、屏蔽，以及进一步的文本识别。

卡证识别

功能说明

本接口支持中国大陆居民二代身份证正面（暂不支持背面）、驾驶证主页（暂不支持副页）所有字段的自动定位，暂不支持文本识别，用于对特定字段的抹除、屏蔽，以及进一步的文本识别。

使用示例

```
function aILicenseRec() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'card.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，卡证识别固定为AILicenseRec;是否必传：是
        'ci-process': 'AILicenseRec',
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url
时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。detect-url
示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后为
http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg;是否必传：否
        // 'detect-url': '',
```

```
        // 卡证识别类型，有效值为IDCard，DriverLicense。<br>IDCard表示身份证；
DriverLicense表示驾驶证，默认：DriverLicense;是否必传：是
        CardType: 'IDCard',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
aILicenseRec();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg。	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，卡证识别固定为AILicenseRec	String	是

detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 、Key ，填写了 detect-url 时，后台会处理 detect-url 链接 。 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg` 。	String	否
CardType	卡证识别类型，默认为DriverLicense。有效值为： • IDCard：表示身份证。 • DriverLicense：表示驾驶证。	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果 详情请参见 卡证识别	Object

相关链接

API 文档

游戏场景识别

最近更新时间：2024-12-27 16:37:06

简介

本文档提供关于游戏场景识别的相关 API 概览以及 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
游戏场景识别	游戏标签功能实现游戏图片场景的识别，返回图片中置信度较高的游戏类别标签。游戏标签识别请求包属于 GET 请求，请求时需要携带签名。

游戏场景识别

功能说明

游戏标签功能实现游戏图片场景的识别，返回图片中置信度较高的游戏类别标签。游戏标签识别请求包属于 GET 请求，请求时需要携带签名。

使用示例

```
function aIGameRec() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/game.jpeg', // 对象文件名，例如：folder/document.jpg；与detect-url
二选一传递
      Query: {
        // 数据万象处理能力，游戏场景识别固定为 AIGameRec;是否必传：是
        'ci-process': 'AIGameRec',
        // 待检查图片url，需要进行urlencode;与Key二选一传递
        // 'detect-url': '',
      },
    },
```

```
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
aIGameRec();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，游戏场景识别固定为 AIGameRec。	String	是
detect-url	处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。 detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为	String	否

	`http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`。		

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果 详情请参见 游戏场景识别	Object

相关参考

API 文档

人脸核身
身份证识别

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于身份证识别的相关 API 概览以及 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
身份证识别	支持中国大陆居民二代身份证正反面所有字段的识别，包括姓名、性别、民族、出生日期、住址、公民身份证号、签发机关、有效期限；具备身份证照片、人像照片的裁剪功能和翻拍、PS、复印件告警功能，以及边框和框内遮挡告警、临时身份证告警和身份证有效期不合法告警等扩展功能。

身份证识别

功能说明

支持中国大陆居民二代身份证正反面所有字段的识别，包括姓名、性别、民族、出生日期、住址、公民身份证号、签发机关、有效期限；具备身份证照片、人像照片的裁剪功能和翻拍、PS、复印件告警功能，以及边框和框内遮挡告警、临时身份证告警和身份证有效期不合法告警等扩展功能。

使用示例

```
function aIIDCardOCR() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/card.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，身份证识别固定为 IDCardOCR;是否必传：是
        'ci-process': 'IDCardOCR',
```

```
        // FRONT：身份证有照片的一面（人像面）BACK：身份证有国徽的一面（国徽面）该参数如果不填，将为您自动判断身份证正反面;是否必传：否
        CardSide: 'FRONT',
        // 以下可选字段均为 bool 类型，默认 false：CropIdCard，身份证照片裁剪（去掉证件外多余的边缘、自动矫正拍摄角度）CropPortrait，人像照片裁剪（自动抠取身份证头像区域）CopyWarn，复印件告警BorderCheckWarn，边框和框内遮挡告警ReshootWarn，翻拍告警DetectPsWarn，PS 检测告警TempIdWarn，临时身份证告警InvalidDateWarn，身份证有效日期不合法告警Quality，图片质量分数（评价图片的模糊程度）MultiCardDetect，是否开启多卡证检测参数设置方式参考：Config = {"CropIdCard":true,"CropPortrait":true};是否必传：否
        // Config: {},
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
aIIDCardOCR();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，身份证识别固定为 IDCardOCR	String	是
CardSide	FRONT：身份证有照片的一面（人像面） BACK：身份证有国徽的一面（国徽面） 该参数如果不填，将为您自动判断身份证正反面	String	否
Config	以下可选字段均为 bool 类型，默认 false： • CropIdCard，身份证照片裁剪（去掉证件外多余的边缘、自动矫正拍摄角度） • CropPortrait，人像照片裁剪（自动抠取身份证头像区域） • CopyWarn，复印件告警 • BorderCheckWarn，边框和框内遮挡告警 • ReshootWarn，翻拍告警 • DetectPsWarn，PS 检测告警 • TempIdWarn，临时身份证告警 • InvalidDateWarn，身份证有效日期不合法告警 • Quality，图片质量分数（评价图片的模糊程度） • MultiCardDetect，是否开启多卡证检测 参数设置方式参考： Config = {"CropIdCard":true,"CropPortrait":true}	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果 详情请参见 身份证识别	Object

相关参考

API 文档

活体人脸核身

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于活体人脸核身的相关 API 概览以及 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
活体人脸核身	集成了活体检测和跟权威库进行比对的能力，传入一段视频和姓名、身份证号信息即可进行验证。对录制的自拍视频进行活体检测，从而确认当前用户为真人，可防止照片、视频、静态3D建模等各种不同类型的攻击。检测为真人后，再判断该视频中的人与权威库的证件照是否属于同一个人，实现用户身份信息核实。

活体人脸核身

功能说明

集成了活体检测和跟权威库进行比对的能力，传入一段视频和姓名、身份证号信息即可进行验证。对录制的自拍视频进行活体检测，从而确认当前用户为真人，可防止照片、视频、静态3D建模等各种不同类型的攻击。检测为真人后，再判断该视频中的人与权威库的证件照是否属于同一个人，实现用户身份信息核实。

使用示例

```
function livenessRecognition() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/tf.mp4', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，人脸核身固定为 LivenessRecognition;是否必传：是
        'ci-process': 'LivenessRecognition',
        // 身份证号;是否必传：是
        IdCard: 'xxxxxxx',
```

```
          // 姓名。中文请使用 UTF-8编码;是否必传：是
          Name: '张三',
          // 活体检测类型，取值：LIP/ACTION/SILENTLIP 为数字模式，ACTION 为动作模式，
SILENT 为静默模式，三种模式选择一种传入;是否必传：是
          LivenessType: 'SILENT',
          // 数字模式传参：数字验证码（1234），需先调用接口获取数字验证码动作模式传参：传动作
顺序（2，1 or 1，2），需先调用接口获取动作顺序静默模式传参：空;是否必传：否
          ValidateData: '',
          // 需要返回多张最佳截图，取值范围1 - 10，不设置默认返回一张最佳截图;是否必传：否
          BestFrameNum: 1,
        },
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
livenessRecognition();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，必须，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填

ci-process	数据万象处理能力，人脸核身固定为 LivenessRecognition	String	是
IdCard	身份证号	String	是
Name	姓名。中文请使用 UTF-8编码	String	是
LivenessType	活体检测类型，取值：LIP/ACTION/SILENTLIP 为数字模式，ACTION 为动作模式，SILENT 为静默模式，三种模式选择一种传入	String	是
ValidateData	数字模式传参：数字验证码（1234），需先调用接口获取数字验证码动作模式传参：传动作顺序（2，1 or 1，2），需先调用接口获取动作顺序静默模式传参：空	String	否
BestFrameNum	需要返回多张最佳截图，取值范围1 - 10，不设置默认返回一张最佳截图	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果 详情请参见 活体人脸核身	Object

相关参考

API 文档
功能指南

版权所有：腾讯云计算（北京）有限责任公司

获取数字验证码

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于获取数字验证码的相关 API 概览以及 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
获取数字验证码	使用数字活体检测模式前，需调用本接口获取数字验证码。

获取数字验证码

功能说明

使用数字活体检测模式前，需调用本接口获取数字验证码。

使用示例

```
function getLiveCode() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Query: {
        // 数据万象处理能力，获取数字验证码固定为 GetLiveCode;是否必传：是
        'ci-process': 'GetLiveCode',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
```

```
            console.log(data.Response);
        }
    }
  );
}
getLiveCode();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，获取数字验证码固定为 GetLiveCode	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 获取数字验证码	Object

相关参考

API 文档

获取动作顺序

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于获取动作顺序的相关 API 概览以及 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
获取动作顺序	使用动作活体检测模式前，需调用本接口获取动作顺序。

获取动作顺序

功能说明

使用动作活体检测模式前，需调用本接口获取动作顺序。

使用示例

```
function getActionSequence() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Query: {
        // 数据万象处理能力，获取动作顺序固定为 GetActionSequence;是否必传：是
        'ci-process': 'GetActionSequence',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
```

```
            console.log(data.Response);
        }
    }
  );
}
getActionSequence();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，获取动作顺序固定为 GetActionSequence	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 获取动作顺序	Object

相关参考

API 文档

商品抠图

最近更新时间：2025-01-25 18:31:21

简介

本文档提供关于商品抠图的相关 API 概览以及 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
商品抠图	腾讯云数据万象通过 GoodsMatting 接口检测图片中的商品信息，生成只包含商品信息的图片，支持持久化、云上处理及下载时处理。

功能说明

腾讯云数据万象通过 GoodsMatting 接口检测图片中的商品信息，生成只包含商品信息的图片，支持持久化、云上处理及下载时处理。

使用示例

```
function goodsMatting() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/goods.jpeg', // 对象文件名，例如：folder/document.jpg；与detect-url
二选一传递
      Query: {
        // ;是否必传：是
        'ci-process': 'GoodsMatting',
        // ;是否必传：否
        // 'detect-url': 'https://example.com/abc.jpg',
      },
      RawBody: true,
    },
```

```
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
        // 写入本地文件
        // fs.writeFileSync('./test.jpeg', Buffer.from(data.Body));
      }
    }
  );
}
goodsMatting();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	否
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，商品抠图固定为 GoodsMatting	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接	String	否

	示例：`http://www.example.com/abc.jpg`		

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请见 商品抠图	Object

相关链接

API 文档
功能指南

内容翻译
提交任务

最近更新时间：2024-12-27 16:37:07

简介

内容翻译功能是由 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。
本文档提供关于内容翻译的相关 API 概览以及 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
提交任务	提交一个翻译任务。

提交任务

功能说明

提交一个翻译任务。

使用示例

```
function postTranslation() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'jobs'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 创建任务的 Tag：Translation;是否必传：是
      Tag: 'Translation',
      // 待操作的对象信息;是否必传：是
      Input: {
        // 源文档文件名单文件（docx/xlsx/html/markdown/txt）：不超过800万字符有页数的（pdf/pptx）：不超过300页文本文件（txt）：不超过10MB二进制文件（pdf/docx/pptx/xlsx）：
```

```
不超过60MB图片文件（jpg/jpeg/png/webp）：不超过10MB;是否必传：是
        Object: 'ci/test.txt',
        // 文档语言类型zh：简体中文zh-hk：繁体中文zh-tw：繁体中文zh-tr：繁体中文en：英语
ar：阿拉伯语de：德语es：西班牙语fr：法语id：印尼语it：意大利语ja：日语pt：葡萄牙语ru：俄语
ko：韩语km：高棉语lo：老挝语;是否必传：是
        Lang: 'zh-hk',
        // 文档类型pdfdocxpptxxlsxtxtxmlhtml：只能翻译 HTML 里的文本节点，需要通过 JS
动态加载的不进行翻译markdownjpgjpegpngwebp;是否必传：是
        Type: 'txt',
        // 原始文档类型仅在 Type=pdf/jpg/jpeg/png/webp 时使用，当值为pdf时，仅支持
docx、pptx当值为jpg/jpeg/png/webp时，仅支持txt;是否必传：否
        // BasicType: '',
      },
      // 操作规则;是否必传：是
      Operation: {
        // 翻译参数;是否必传：是
        Translation: {
          // 目标语言类型源语言类型为 zh/zh-hk/zh-tw/zh-tr 时支持：en、ar、de、es、
fr、id、it、ja、it、ru、ko、km、lo、pt源语言类型为 en 时支持：zh、zh-hk、zh-tw、zh-
tr、ar、de、es、fr、id、it、ja、it、ru、ko、km、lo、pt其他类型时支持：zh、zh-hk、zh-
tw、zh-tr、en;是否必传：是
          Lang: 'en',
          // 文档类型，源文件类型与目标文件类型映射关系如下：docx：docxpptx：pptxxlsx：
xlsxtxt：txtxml：xmlhtml：htmlmarkdown：markdownpdf：pdf、docxpng：txtjpg：
txtjpeg：txtwebp：txt;是否必传：是
          Type: 'txt',
        },
        // 结果输出地址，当NoNeedOutput为true时非必选;是否必传：否
        Output: {
          // 存储桶的地域;是否必传：是
          Region: config.Region,
          // 存储结果的存储桶;是否必传：是
          Bucket: config.Bucket,
          // 输出结果的文件名;是否必传：是
          Object: 'ci/trans_test.txt',
        },
        // 透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
        // UserData: '',
        // 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;是否必传：否
        // JobLevel: '',
        // 仅输出结果，不生成结果文件。取值：true/false。该参数原文档类型为图片时有效。默
认值 false;是否必传：否
        // NoNeedOutput: '',
      },
      // 任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
      // CallBackFormat: '',
```

```
      // 任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
      // CallBackType: '',
      // 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;是否必传：否
      // CallBack: '',
      // 任务回调TDMQ配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig;是否必传：否
      // CallBackMqConfig: {
      // 消息队列所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）;是否必传：是
      // MqRegion: '',
      // 消息队列使用模式，默认 Queue ：主题订阅：Topic队列服务：Queue;是否必传：是
      // MqMode: '',
      // TDMQ 主题名称;是否必传：是
      // MqName: '',
      // },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
postTranslation();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填

Method	固定值：POST	String	是
Key	固定值：jobs	String	是
Url	请求url，固定值	String	是
Body	请求体，详情请参见 提交任务	Object	是
ContentType	固定值	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 提交任务	Object

相关链接

API 文档

视频目标检测
提交任务

最近更新时间：2024-12-27 16:37:07

简介

提交一个视频目标检测任务功能是由 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。
本文档提供关于提交一个视频目标检测任务的相关 API 概览以及 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
提交任务	提交一个视频目标检测任务。

提交任务

功能说明

提交一个视频目标检测任务

使用示例

```
function postVideoTargetRec() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'jobs'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 创建任务的 Tag：VideoTargetRec;是否必传：是
      Tag: 'VideoTargetRec',
      // 操作规则;是否必传：是
      Operation: {
        // 视频目标检测模板 ID;与VideoTargetRec二选一传递
        TemplateId: 'xxxx',
```

```
        // 视频目标检测参数，同创建视频目标检测模板接口中的 Request.VideoTargetRec;与TemplateId二选一传递
        // VideoTargetRec: {
        // 是否开启人体检测，取值 true/false;是否必传：否
        // Body: '',
        // 是否开启宠物检测，取值 true/false;是否必传：否
        // Pet: '',
        // 是否开启车辆检测，取值 true/false;是否必传：否
        // Car: '',
        // },
        // 透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
        // UserData: '',
        // 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;是否必传：否
        // JobLevel: '',
      },
      // 待操作的媒体信息;是否必传：是
      Input: {
        // 媒体文件名;是否必传：否
        Object: 'ci/test.mp4',
      },
      // 任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
      // CallBackFormat: '',
      // 任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
      // CallBackType: '',
      // 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;是否必传：否
      // CallBack: '',
      // 任务回调TDMQ配置，当 CallBackType 为 TDMQ 时必填。详情请参见 CallBackMqConfig;是否必传：否
      // CallBackMqConfig: {
      // 消息队列所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）;是否必传：是
      // MqRegion: '',
      // 消息队列使用模式，默认 Queue ：主题订阅：Topic队列服务：Queue;是否必传：是
      // MqMode: '',
      // TDMQ 主题名称;是否必传：是
      // MqName: '',
      // },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
```

```
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
postVideoTargetRec();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Key	固定值：jobs	String	是
Url	请求url，固定值	String	是
Body	请求体，详情请参见 提交任务	Object	是
ContentType	固定值	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 提交任务	Object

相关参考

API 文档
功能指南

更新模板

最近更新时间：2024-12-27 16:37:07

简介

更新视频目标检测模板功能是由 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。
本文档提供关于更新视频目标检测模板的相关 API 概览以及 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
更新模板	更新视频目标检测模板。

更新模板

功能说明

更新视频目标检测模板。

使用示例

```
function updateVideoTargetTemplete() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const templateId = 'xxxxxxx'; // 要更新的模板id
  const key = `template/${templateId}`; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 模板类型：VideoTargetRec;是否必传：是
      Tag: 'VideoTargetRec',
      // 模板名称，仅支持中文、英文、数字、_、-和*，长度不超过 64;是否必传：是
      Name: 'my_videoTargetRec2',
      // 视频目标检测 参数;是否必传：是
      VideoTargetRec: {
        // 是否开启人体检测，取值 true/false;是否必传：否
```

```
        Body: 'false',
        // 是否开启宠物检测，取值 true/false;是否必传：否
        Pet: 'false',
        // 是否开启车辆检测，取值 true/false;是否必传：否
        Car: 'true',
      },
    },
  });

  cos.request(
    {
      Method: 'PUT', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
updateVideoTargetTemplete();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：PUT	String	是
Url	请求url，固定值	String	是
Key	template/{TemplateId}	String	是
ContentType	固定值	String	是
Body	请求体，详情请参见 更新模板	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果，详情请参见 更新模板	Object

相关参考

API 文档
功能指南

创建模板

最近更新时间：2024-12-27 16:37:07

简介

创建视频目标检测模板功能是由 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。
本文档提供关于创建视频目标检测模板的相关 API 概览以及 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
创建模板	创建视频目标检测模板。

创建模板

功能说明

创建视频目标检测模板。

使用示例

```
function postVideoTargetTemplete() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'template'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 模板类型：VideoTargetRec;是否必传：是
      Tag: 'VideoTargetRec',
      // 模板名称，仅支持中文、英文、数字、_、-和*，长度不超过 64;是否必传：是
      Name: 'my_videoTargetRec',
      // 视频目标检测 参数;是否必传：是
      VideoTargetRec: {
        // 是否开启人体检测，取值 true/false;是否必传：否
        Body: 'true',
```

```
        // 是否开启宠物检测，取值 true/false;是否必传：否
        Pet: 'true',
        // 是否开启车辆检测，取值 true/false;是否必传：否
        Car: 'false',
      },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
postVideoTargetTemplete();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Url	请求url，固定值	String	是
Key	固定值：template	String	是
ContentType	固定值	String	是
Body	请求体，详情请参见 创建模板	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 创建模板	Object

相关参考

API 文档
功能指南

视频人像分割
提交任务

最近更新时间：2024-12-27 16:37:07

简介

视频人像分割功能是由 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。
本文档提供关于视频人像分割的相关 API 概览以及 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
提交任务	提交一个视频人像抠图任务。

提交任务

功能说明

提交一个视频人像抠图任务。

使用示例

```
function postSegmentVideoBody() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/8629
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'jobs'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 创建任务的 Tag：SegmentVideoBody;是否必传：是
      Tag: 'SegmentVideoBody',
      // 待操作的对象信息;是否必传：是
      Input: {
        // 文件路径;是否必传：是
        Object: 'ci/test.mp4',
```

```
        },
        // 操作规则;是否必传：是
        Operation: {
          // 视频人像抠图配置;是否必传：否
          SegmentVideoBody: {
            // 抠图模式 Mask：输出alpha通道结果Foreground：输出前景视频Combination：输出抠图后的前景与自定义背景合成后的视频默认值：Mask;是否必传：否
            Mode: 'Mask',
            // 抠图类型HumanSeg：人像抠图GreenScreenSeg：绿幕抠图SolidColorSeg：纯色背景抠图默认值：HumanSeg;是否必传：否
            // SegmentType: '',
            // mode为 Foreground 时参数生效，背景颜色为红色，取值范围 [0, 255]， 默认值为 0;是否必传：否
            // BackgroundRed: '',
            // mode为 Foreground 时参数生效，背景颜色为绿色，取值范围 [0, 255]，默认值为 0;是否必传：否
            // BackgroundGreen: '',
            // mode为 Foreground 时参数生效，背景颜色为蓝色，取值范围 [0, 255]，默认值为 0;是否必传：否
            // BackgroundBlue: '',
            // 传入背景文件。mode为 Combination 时，此参数必填，背景文件需与源文件在同存储桶下;是否必传：否
            // BackgroundLogoUrl: '',
            // 调整抠图的边缘位置，取值范围为[0, 255]，默认值为 0;是否必传：否
            // BinaryThreshold: '',
            // 纯色背景抠图的背景色（红），当 SegmentType 为 SolidColorSeg 生效，取值范围为 [0, 255]，默认值为 0;是否必传：否
            // RemoveRed: '',
            // 纯色背景抠图的背景色（绿），当 SegmentType 为 SolidColorSeg 生效，取值范围为 [0, 255]，默认值为 0;是否必传：否
            // RemoveGreen: '',
            // 纯色背景抠图的背景色（蓝），当 SegmentType 为 SolidColorSeg 生效，取值范围为 [0, 255]，默认值为 0;是否必传：否
            // RemoveBlue: '',
          },
          // 结果输出配置;是否必传：是
          Output: {
            // 存储桶的地域;是否必传：是
            Region: config.Region,
            // 存储结果的存储桶;是否必传：是
            Bucket: config.Bucket,
            // 输出结果的文件名;是否必传：是
            Object: 'ci/out_test.mp4',
          },
          // 透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
          // UserData: '',
```

```
        // 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;是否必传：否
        // JobLevel: '',
      },
      // 任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
      // CallBackFormat: '',
      // 任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
      // CallBackType: '',
      // 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;是否必传：否
      // CallBack: '',
      // 任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig;是否必传：否
      // CallBackMqConfig: {
      // 消息队列所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）;是否必传：是
      // MqRegion: '',
      // 消息队列使用模式，默认 Queue ：主题订阅：Topic队列服务：Queue;是否必传：是
      // MqMode: '',
      // TDMQ 主题名称;是否必传：是
      // MqName: '',
      // },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
postSegmentVideoBody();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Url	请求url，固定值	String	是
Key	固定值：jobs	String	是
ContentType	固定值	String	是
Body	请求体，详情请参见 提交任务	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果，详情请参见 提交任务	Object

相关参考

API 文档

PHP SDK 快速入门

最近更新时间：2025-06-09 19:23:32

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。
> 如果您仍在使用 JSON 版本 SDK，请 升级到 XML PHP SDK 。

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。

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 = "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.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.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.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)));
```

> **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 。

若您使用 临时密钥 初始化，请用下面方式创建实例：

```
$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.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.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.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)));
```

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 $rt) {
                // 打印key
                echo($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";
}

### 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);
    } 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);
    }
```

内容识别
人脸特效

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于人脸特效的相关 API 概览以及 SDK 示例代码。

API	操作描述
人脸特效	人脸特效，提供人脸美颜、人像变换、人像分割功能。

人脸特效

功能说明

人脸特效，提供人脸美颜、人像变换、人像分割功能。

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.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.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.cloud.tencent.com/cos5/bucket
$cosClient = new Qcloud\Cos\Client(
    array(
        'region' => $region,
        'scheme' => 'https', // 审核时必须为https
        'credentials'=> array(
            'secretId'  => $secretId ,
            'secretKey' => $secretKey)));
try {
    //人脸特效，提供人脸美颜、人像变换、人像分割功能
    $result = $cosClient->ImageFaceEffect(array(
        'Bucket' => 'test-1250000000', //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
        'Key' => 'folder/image.jpg', // 对象文件名，例如 folder/document.jpg
        'Headers' => array(
            'Content-Type' => 'application/xml',
```

```
        ),
        'type' => 'face-segmentation', // 人脸特效类型，人脸美颜：face-beautify；人脸性别转换：face-gender-transformation；人脸年龄变化：face-age-transformation；人像分割：face-segmentation
        'whitening' => 30, // type为face-beautify时生效，美白程度，取值范围[0,100]。0不美白，100代表最高程度。默认值30
        'smoothing' => 10, // type为face-beautify时生效，磨皮程度，取值范围[0,100]。0不磨皮，100代表最高程度。默认值10
        'faceLifting' => 70, // type为face-beautify时生效，瘦脸程度，取值范围[0,100]。0不瘦脸，100代表最高程度。默认值70
        'eyeEnlarging' => 70, // type为face-beautify时生效，大眼程度，取值范围[0,100]。0不大眼，100代表最高程度。默认值70
        'gender' => 1, // type为face-gender-transformation时生效，选择转换方向，0：男变女，1：女变男。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。
        'age' => 18, // type为face-age-transformation时生效，变化到的人脸年龄，[10,80]。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。

    ));
    // 请求成功
    print_r($result);
} catch (\Exception $e) {
    // 请求失败
    echo($e);
}
```

参数说明

请求参数

参数名称	参数描述	类型	是否必填
detectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey detect-url 示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后为 http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg。	string	否
type	人脸特效类型，人脸美颜：face-beautify；人脸性别转换：face-gender-transformation；人脸年龄变化：face-age-transformation；人像分割：face-segmentation。	string	是
whitening	type为face-beautify时生效，美白程度，取值范围[0,100]。0不美白，100代表最高程度，默认值30。	integer	否
smooth	type为face-beautify时生效，磨皮程度，取值范围[0,100]。0不磨皮，	integ	否

ing	100代表最高程度，默认值10。	er	
faceLifting	type为face-beautify时生效，瘦脸程度，取值范围[0,100]。0不瘦脸，100代表最高程度，默认值70。	integer	否
eyeEnlarging	type为face-beautify时生效，大眼程度，取值范围[0,100]。0不大眼，100代表最高程度，默认值70。	integer	否
gender	type为face-gender-transformation时生效，选择转换方向，0：男变女，1：女变男。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。	integer	否
age	type为face-age-transformation时生效，变化到的人脸年龄,[10,80]。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。	integer	否
Bucket	存储桶名称。	String	是
Key	对象文件名，例如 folder/document.jpg。	String	否

返回参数说明

具体返回参数，请参见 人脸特效 接口。

图片标签

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于图片标签相关的 API 概览以及 SDK 示例代码。

API	说明
图片标签	图片标签功能通过借助持久化处理接口，实现对对象存储（Cloud Object Storage，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对象存储的持久化处理接口，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。

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.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.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.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->DetectLabel(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.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
    (
            [RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
            [Key] => exampleobject
            [Bucket] => examplebucket-1250000000
            [Location] => examplebucket-1250000000.cos.ap-
beijing.myqcloud.com/exampleobject
            [Labels] => Array
            (
                [0] => Array
                (
                    [Confidence] => 83
                    [Name] => 玩具
                )

                [1] => Array
                (
                    [Confidence] => 77
                    [Name] => 毛绒玩具
                )
```

```
                [2] => Array
                (
                    [Confidence] => 15
                    [Name] => 艺术
                )
            )
        )
)
```

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
Labels	Array	标签信息	无

图片二维码

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于图片二维码相关的 API 概览以及 SDK 示例代码。

API	说明
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
二维码生成	二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码

二维码识别

功能说明

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。

请求示例1：上传时识别

```
<?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.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.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.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 {
        $imageQrcodeTemplate = new
Qcloud\Cos\ImageParamTemplate\ImageQrcodeTemplate();//创建二维码识别参数模版实例
```

```
        $imageQrcodeTemplate->setMode(0);//二维码覆盖功能。0表示不开启二维码覆盖，1表示开启二维码覆盖功能，开启后，将对识别出的二维码覆盖上马赛克，默认值0
        $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
        $picOperationsTemplate->addRule($imageQrcodeTemplate, "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.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
                            [CodeStatus] => 1
                            [QRcodeInfo] => Array
                            (
```

```
                                    [0] => Array
                                    (
                                            [CodeUrl] => xxxxxxxxxxxx
                                            [CodeLocation] => Array
                                            (
                                                    [Point] => Array
                                                    (
                                                            [0] => 100,100
                                                            [1] => 100,200
                                                            [2] => 200,200
                                                            [3] => 200,100
                                                    )
                                            )
                                    )
                            )
                            [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(二维码信息)	无

请求示例2：下载时识别

```
<?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.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.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.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 = $cosClient->Qrcode(array(
        'Bucket' => 'examplebucket-1250000000', //存储桶名称，由 BucketName-Appid 组成，可以在 COS 控制台查看 https://console.cloud.tencent.com/cos5/bucket
        'Key' => 'exampleobject',
        'Cover' => 1,//二维码覆盖功能
    ));
    // 请求成功
    print_r($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
Cover	Int	二维码覆盖功能，将对识别出的二维码覆盖上马赛克。0表示不开启二维码覆盖，1表示开启二维码覆盖，默认值0	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
    [structure:protected] =>
    [data:protected] => Array
    (
            [RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
            [ContentLength] => 238186
            [Key] => exampleobject
            [Bucket] => examplebucket-1250000000
            [Location] => examplebucket-1250000000.cos.ap-
beijing.myqcloud.com/exampleobject
            [CodeStatus] => 1
            [QRcodeInfo] => Array
            (
                    [0] => Array
                    (
                            [CodeUrl] => xxxxxxxxxxxx
                            [CodeLocation] => Array
                            (
                                    [Point] => Array
                                    (
                                            [0] => 100,100
                                            [1] => 100,200
                                            [2] => 200,200
                                            [3] => 200,100
                                    )
                             )
                    )
            )
            [ResultImage] =>

    )
)
```

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
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
CodeStatus	Int	二维码识别结果。0表示未识别到二维码，1表示识别到二维码	无
QRcodeInfo	Array	二维码识别结果，可能有多个	无
ResultImage	String	处理后的图片 base64数据，请求参数 cover 为1时返回	无

二维码生成

功能说明

二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码。

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.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.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.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->QrcodeGenerate(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.cloud.tencent.com/cos5/bucket
        'QrcodeContent' => '<https://www.example.com>',//可识别的二维码文本信息
```

```
        'QrcodeMode' => 0 //生成的二维码类型，可选值：0或1。0为二维码，1为条形码，默认值为0
        'QrcodeWidth' => '200',//指定生成的二维码或条形码的宽度，高度会进行等比压缩
    ));
    // 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
QrcodeContent	String	可识别的二维码文本信息	是
QrcodeMode	Int	生成的二维码类型，可选值：0或1。0为二维码，1为条形码，默认值为0	是
QrcodeWidth	String	指定生成的二维码或条形码的宽度，高度会进行等比压缩	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.cos.ap-beijing.myqcloud.com/
            [ResultImage] =>
    )
)
```

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
ResultImage	String	处理后的图片 base64数据	无

图片文字识别（OCR）

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于通用文字识别的 API 概览和 SDK 示例代码。

API	操作描述
通用文字识别	通用文字识别功能将图片上的文字内容，智能识别为可编辑的文本。

通用文字识别

功能说明

通用文字识别功能（Optical Character Recognition，OCR）基于行业前沿的深度学习技术，将图片上的文字内容，智能识别为可编辑的文本，可应用于随手拍扫描、纸质文档电子化、电商广告审核等多种场景，大幅提升信息处理效率。

方法原型

```
public Guzzle\Service\Resource\Model opticalOcrRecognition(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.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.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.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/436/64324 通用文字识别
    $result = $cosClient->opticalOcrRecognition(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.cloud.tencent.com/cos5/bucket
        'Key' => 'test01.pdf',
        'CiProcess' => 'OCR',
        'Type' => 'general',
        'LanguageType' => 'zh',
        'IsPDF' => 'true',
        'PdfPageNumber' => 2,
        'IsWord' => 'true',
        'EnableWordPolygon' => 'false',
    ));
    // 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	是否必选
Key	对象文件名，例如：folder/document.jpg。	String	是
CiProcess	数据万象处理能力，图片文字识别固定为 OCR。	String	是
Type	OCR 的识别类型，有效值为 general，accurate，efficient，fast，handwriting。general 表示通用印刷体识别；accurate 表示印刷体高精度版；efficient 表示印刷体精简版；fast 表示印刷体高速版；handwriting 表示手写体识别。默认值为 general。	String	否
LanguageType	type 值为 general 时有效，表示识别语言类型。支持自动识别语言类型，同时支持自选语言种类，默认中英文混合(zh)，各种语言均支持与英文混合的文字识别。可选值请参见 可识别的语言类型 。	String	否
IsPDF	type 值为 general、fast 时有效，表示是否开启 PDF 识别，有效值为 true 和 false，默认值为 false，开启后可同时支持图片和 PDF 的识别。	Boolean	否
PdfPageNumber	type 值为 general、fast 时有效，表示需要识别的 PDF 页面的对应页码，仅支持 PDF 单页识别，当上传文件为 PDF 且 ispdf 参数值为 true 时有效，默认值为1。	Integer	否
IsWord	type 值为 general、accurate 时有效，表示识别后是否需要返回单字信息，有效值为 true 和 false，默认为 false。	Boolean	否

EnableWordPolygon	type 值为 handwriting 时有效，表示是否开启单字的四点定位坐标输出，有效值为 true 和 false，默认值为 false。	Boolean	否

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] => NjJhOThmYjZfAOIUDOIUSDOITkwYTU1Zg==
    [ContentType] => application/xml
    [ContentLength] => 3144
    [Key] => test01.pdf
    [Bucket] => examplebucket-1250000000
    [Location] => examplebucket-1250000000.cos.ap-
guangzhou.myqcloud.com/test01.pdf
    [Response] => Array
        (
            [Angel] => 359.99
            [Language] => zh
            [PdfPageSize] => 5
            [RequestId] => NjJhOThmYjZfAOIUDOIUSDOITkwYTU1Zg=
            [TextDetections] => Array
                (
                    [0] => Array
                        (
                            [Confidence] => 99
                            [DetectedText] => 第二页
                            [ItemPolygon] => Array
                                (
                                    [Height] => 118
                                    [Width] => 351
                                    [X] => 782
                                    [Y] => 403
```

```
            )

        [Polygon] => Array
            (
                [0] => Array
                    (
                        [X] => 782
                        [Y] => 403
                    )

                [1] => Array
                    (
                        [X] => 1133
                        [Y] => 403
                    )

                [2] => Array
                    (
                        [X] => 1133
                        [Y] => 521
                    )

                [3] => Array
                    (
                        [X] => 782
                        [Y] => 521
                    )

            )

        [Words] => Array
            (
                [0] => Array
                    (
                        [Character] => 第
                        [Confidence] => 100
                        [WordCoordPoint] => Array
                            (
                                [WordCoordinate] => Array
                                    (
                                        [0] => Array
                                            (
                                                [X] => 796
                                                [Y] => 403
                                            )
```

```
                                        [1] => Array
                                            (
                                                [X] => 885
                                                [Y] => 403
                                            )

                                        [2] => Array
                                            (
                                                [X] => 885
                                                [Y] => 521
                                            )

                                        [3] => Array
                                            (
                                                [X] => 796
                                                [Y] => 521
                                            )

                                    )

                                )

                            )

                        [1] => Array
                            (
                                [Character] => 二
                                [Confidence] => 99
                                [WordCoordPoint] => Array
                                    (
                                        [WordCoordinate] => Array
                                            (
                                                [0] => Array
                                                    (
                                                        [X] => 885
                                                        [Y] => 403
                                                    )

                                                [1] => Array
                                                    (
                                                        [X] =>
1034
                                                        [Y] => 403
                                                    )

                                                [2] => Array
```

```
                                                    (
                                                        [X] =>
1034
                                                        [Y] => 521
                                                    )

                                                [3] => Array
                                                    (
                                                        [X] => 885
                                                        [Y] => 521
                                                    )

                                            )

                                        )

                                )

                            [2] => Array
                                (
                                    [Character] => 页
                                    [Confidence] => 99
                                    [WordCoordPoint] => Array
                                        (
                                            [WordCoordinate] => Array
                                                (
                                                    [0] => Array
                                                        (
                                                            [X] =>
1033
                                                            [Y] => 403
                                                        )

                                                    [1] => Array
                                                        (
                                                            [X] =>
1133
                                                            [Y] => 403
                                                        )

                                                    [2] => Array
                                                        (
                                                            [X] =>
1133
                                                            [Y] => 521
                                                        )
```

```
                                                    [3] => Array
                                                        (
                                                            [X] =>
1033
                                                            [Y] => 521
                                                        )

                                                )

                                        )

                                )

                        )

                [1] => Array
                    (
                        [Confidence] => 99
                        [DetectedText] => 内容
                        [ItemPolygon] => Array
                            (
                                [Height] => 50
                                [Width] => 93
                                [X] => 914
                                [Y] => 569
                            )

                        [Polygon] => Array
                            (
                                [0] => Array
                                    (
                                        [X] => 914
                                        [Y] => 569
                                    )

                                [1] => Array
                                    (
                                        [X] => 1007
                                        [Y] => 572
                                    )

                                [2] => Array
                                    (
```

```
                                [X] => 1005
                                [Y] => 622
                            )

                        [3] => Array
                            (
                                [X] => 912
                                [Y] => 619
                            )

                    )

                [Words] => Array
                    (
                        [0] => Array
                            (
                                [Character] => 内
                                [Confidence] => 99
                                [WordCoordPoint] => Array
                                    (
                                        [WordCoordinate] => Array
                                            (
                                                [0] => Array
                                                    (
                                                        [X] => 919
                                                        [Y] => 569
                                                    )

                                                [1] => Array
                                                    (
                                                        [X] => 958
                                                        [Y] => 570
                                                    )

                                                [2] => Array
                                                    (
                                                        [X] => 956
                                                        [Y] => 620
                                                    )

                                                [3] => Array
                                                    (
                                                        [X] => 917
                                                        [Y] => 619
                                                    )
```

```
                                        )

                                    )

                                )

                            [1] => Array
                                (
                                    [Character] => 容
                                    [Confidence] => 99
                                    [WordCoordPoint] => Array
                                        (
                                            [WordCoordinate] => Array
                                                (
                                                    [0] => Array
                                                        (
                                                            [X] => 958
                                                            [Y] => 570
                                                        )

                                                    [1] => Array
                                                        (
                                                            [X] =>
1007
                                                            [Y] => 572
                                                        )

                                                    [2] => Array
                                                        (
                                                            [X] =>
1005
                                                            [Y] => 622
                                                        )

                                                    [3] => Array
                                                        (
                                                            [X] => 956
                                                            [Y] => 620
                                                        )

                                                )

                                        )

                                )
```

```
                                        )

                                )

                        )

                )

        )
```

图片质量评分

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于图片质量评分的 API 概览和 SDK 示例代码。

API	操作描述
图片质量评分	对图片文件进行检测，从多方面评估，最终给出综合可观的清晰度评分和主观的美观度评分。

图片质量评分

功能说明

图片质量评分功能为同步请求方式，您可以通过本接口对图片文件进行检测，从多方面评估，最终给出综合可观的清晰度评分和主观的美观度评分。该接口属于 GET 请求。

方法原型

```
public Guzzle\Service\Resource\Model imageAssessQualityProcess(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.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.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.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 {
    // 图片质量评估 - https://cloud.tencent.com/document/product/460/63228
    $result = $cosClient->imageAssessQualityProcess(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.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	是否必选
Key	对象文件名，例如：folder/document.jpg。	String	是

返回结果示例

```
GuzzleHttp\Command\Result Object
(
    [RequestId] => NjM5MmU2ZWJfOTQyAHSODHAOIDHAIOSDz==
    [ContentType] => application/xml
    [ContentLength] => 317
    [LongImage] => false
    [BlackAndWhite] => false
    [SmallImage] => false
    [BigImage] => false
    [PureImage] => false
    [ClarityScore] => 86
    [AestheticScore] => 92
    [Key] => test.png
    [Bucket] => examplebucket-1250000000
    [Location] => examplebucket-1250000000.cos.ap-
guangzhou.myqcloud.com/test.png
)
```

异步服务和队列配置

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于异步服务和队列配置接口的 API 概览和 SDK 示例代码。

> **注意**
> 旧版本可能存在 bug，使用时建议升级到 最新版本 。

API	操作描述
开通AI 内容识别（异步）服务	开通 AI 内容识别服务并生成队列
查询AI 内容识别（异步）服务	查询 AI 内容识别服务状态
查询AI 内容识别（异步）队列	搜索 AI 内容识别队列
更新AI 内容识别（异步）队列	更新 AI 内容识别队列

开通AI 内容识别（异步）服务

功能说明

开通 AI 内容识别服务并生成队列。

方法原型

```
public Guzzle\Service\Resource\Model openAiService(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.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.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.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 {
    // 开通 AI 内容识别 https://cloud.tencent.com/document/product/460/79593
    $result = $cosClient->openAiService(array(
        'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid
组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
    ));
    print_r($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

返回结果示例

```
GuzzleHttp\Command\Result Object
(
    [RequestId] => NjRjMjIxODBfMzUxNGSIOJOIDAJOIDg=
    [ContentType] => application/xml
    [ContentLength] => 336
    [AiBucket] => Array
        (
            [BucketId] => examplebucket-1250000000
            [Name] => examplebucket-1250000000
            [Region] => ap-guangzhou
            [CreateTime] => 2023-07-28T11:05:49+0800
        )

    [Bucket] => examplebucket-1250000000
    [Location] => examplebucket-1250000000.ci.ap-
guangzhou.myqcloud.com/ai_bucket
)
```

查询AI 内容识别（异步）服务

功能说明

查询 AI 内容识别服务状态。

方法原型

```
public Guzzle\Service\Resource\Model getAiBucketList(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.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.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.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 AI 内容识别服务状态
https://cloud.tencent.com/document/product/460/79594
    $result = $cosClient->getAiBucketList(array(
//        'Regions' => '', // 可选 地域信息，例如 ap-shanghai、ap-beijing，若查询多
个地域以“,”分隔字符串
//        'BucketNames' => '', // 可选 存储桶名称，以“,”分隔，支持多个存储桶，精确搜索
//        'BucketName' => '', // 可选 存储桶名称前缀，前缀搜索
//        'PageNumber' => 1, // 可选 第几页
//        'PageSize' => 20, // 可选 每页个数
    ));
    // 请求成功
    print_r($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，默认值1	String	否
PageSize	每页个数，默认值10	String	否

返回结果示例

```
GuzzleHttp\Command\Result Object
(
    [RequestId] => NjRjMjIxODBfMzUxNGSIOJOIDAJOIDg=
    [ContentType] => application/xml
    [ContentLength] => 861
    [TotalCount] => 40
    [PageNumber] => 1
    [PageSize] => 3
    [AiBucketList] => Array
        (
            [0] => Array
                (

                    [BucketId] => examplebucket-1250000000
                    [Name] => examplebucket-1250000000
                    [Region] => ap-chongqing
                    [CreateTime] => 2023-07-04T15:33:37+0800
                )

            [1] => Array
                (
                    [BucketId] => examplebucket-1250000000
                    [Name] => examplebucket-1250000000
                    [Region] => ap-chongqing
```

```
                    [CreateTime] => 2023-06-29T19:35:43+0800
                )

            [2] => Array
                (

                    [BucketId] => examplebucket-1250000000
                    [Name] => examplebucket-1250000000
                    [Region] => ap-guangzhou
                    [CreateTime] => 2023-06-26T11:20:13+0800
                )

        )

    [Location] => ci.ap-guangzhou.myqcloud.com/ai_bucket
)
```

查询AI 内容识别（异步）队列

功能说明

搜索 AI 内容识别队列。

方法原型

```
public Guzzle\Service\Resource\Model getAiQueueList(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.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.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.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 {
    // 搜索 AI 内容识别队列 https://cloud.tencent.com/document/product/460/79394
    $result = $cosClient->getAiQueueList(array(
        'Bucket' => 'examplebucket-1250000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
//         'QueueIds' => 'xxx', // 队列 ID，以“,”符号分割字符串
//         'State' => 'Active', // Active 表示队列内的作业会被媒体处理服务调度执行，Paused 表示队列暂停，作业不再会被媒体处理调度执行，队列内的所有作业状态维持在暂停状态，已经执行中的任务不受影响
//         'PageNumber' => '1', // 第几页
//         'PageSize' => '10', // 每页个数
    ));
    print_r($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（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	1. Active 表示队列内的作业会被媒体处理服务调度执行 2. 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
PageNumber	第几页，默认值1	String	否
PageSize	每页个数，默认值10	String	否

返回结果示例

```
GuzzleHttp\Command\Result Object
(
    [RequestId] => NjRjMjIxODBfMzUxNGSIOJOIDAJOIDg=
    [ContentType] => application/xml
    [ContentLength] => 760
    [TotalCount] => 1
    [PageNumber] => 1
```

```
    [PageSize] => 10
    [QueueList] => Array
        (
            [0] => Array
                (
                    [BucketId] => examplebucket-1250000000
                    [QueueId] => pc7a00f19asf798sd7f89s7f2e646
                    [Name] => queue-ai-process
                    [State] => Active
                    [MaxSize] => 10000
                    [MaxConcurrent] => 10
                    [Category] => AIProcess
                    [UpdateTime] => 2023-06-26T14:45:41+0800
                    [CreateTime] => 2023-06-26T11:20:13+0800
                    [NotifyConfig] => Array
                        (
                            [Url] =>
                            [State] => Off
                            [Type] =>
                            [Event] =>
                            [ResultFormat] => XML
                            [MqMode] =>
                            [MqRegion] =>
                            [MqName] =>
                        )

                )

        )

    [Bucket] => examplebucket-1250000000
    [Location] => examplebucket-1250000000.ci.ap-
guangzhou.myqcloud.com/ai_queue
)
```

更新AI 内容识别（异步）队列

功能说明

更新 AI 内容识别队列。

方法原型

```
public Guzzle\Service\Resource\Model updateAiQueue(array $args = array());
```

请求示例

```php
<?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.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.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.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 AI 内容识别队列 https://cloud.tencent.com/document/product/460/79397
    $result = $cosClient->updateAiQueue(array(
        'Bucket' => 'examplebucket-125000000', //存储桶名称，由BucketName-Appid 组成，可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
        'Key' => '', // queueId
        'Name' => 'queue-ai-process',
        'State' => 'Active',
        'NotifyConfig' => array(
            'State' => 'Off',
//            'Event' => '',
//            'ResultFormat' => '',
//            'Type' => '',
//            'Url' => '',
//            'MqMode' => '',
//            'MqRegion' => '',
//            'MqName' => '',
        ),
    ));
    // 请求成功
    print_r($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
Key	Request	待更新队列的 队列 ID	String	是	无	无
Name	Request	队列名称，长度不超过128	String	是	无	无
State	Request	1. Active 表示队列内的作业会被媒体处理服务调度执行； 2. 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

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
(
    [RequestId] => NjRjMNjRjMjIxODBfMzUxNGSIOJOIDAJOIDg=
    [ContentType] => application/xml
    [ContentLength] => 671
    [Queue] => Array
        (
            [BucketId] => examplebucket-1250000000
            [QueueId] => pc7a00f19asf798sd7f89s7f2e646
            [Name] => queue-ai-process-3
            [State] => Active
            [MaxSize] => 10000
            [MaxConcurrent] => 10
            [Category] => AIProcess
            [UpdateTime] => 2023-07-28T11:21:37+0800
            [CreateTime] => 2023-06-26T11:20:13+0800
            [NotifyConfig] => Array
                (
                    [State] => Off
                    [Event] =>
                    [ResultFormat] => XML
                    [Type] =>
                    [Url] =>
                    [MqMode] =>
                    [MqRegion] =>
                    [MqName] =>
                )
```

```
        )

    [Key] => pc7a00f19asf798sd7f89s7f2e646
    [Bucket] => examplebucket-1250000000
    [Location] => examplebucket-1250000000.ci.ap-
guangzhou.myqcloud.com/ai_queue/pc7a00f19asf798sd7f89s7f2e646
)
```

Python SDK 快速入门

最近更新时间：2024-12-27 16:37:07

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。
> 如果您仍在使用 JSON 版本 SDK，请 升级到 XML Python SDK 。

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。

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 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.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.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.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://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 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.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.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.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://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 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.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.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://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://cloud.tencent.com/document/product/436/38864
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 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.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.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://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://cloud.tencent.com/document/product/436/36638
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.mycloud.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 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.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.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://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://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 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.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.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://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://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 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.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.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.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://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 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.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.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.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://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 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.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.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.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://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://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 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.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.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.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://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 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.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.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.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://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
    )
    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 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.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.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.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://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 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.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.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.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://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'
    }
)
```

内容识别
语音识别
任务接口

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于语音识别任务接口的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
提交语音识别任务	提交语音识别任务	用于提交一个语音识别任务
查询指定的语音识别任务	查询指定的语音识别任务	用于查询指定的语音识别任务
批量拉取语音识别任务	拉取符合条件的语音识别任务	用于拉取符合条件的语音识别任务

提交语音识别任务

功能说明

ci_create_asr_job 接口用于提交一个语音识别任务。

方法原型

```
def ci_create_asr_job(self, Bucket, OutputBucket, OutputRegion, OutputObject,
                      QueueId=None, InputObject=None, Url=None,
TemplateId=None,
                      SpeechRecognition=None, CallBack=None,
CallBackFormat=None,
                      CallBackType=None, CallBackMqConfig=None, **kwargs)
```

请求示例

```
def ci_create_asr_jobs():
    # 创建语音识别异步任务(使用模板)
    response = client.ci_create_asr_job(
        Bucket=bucket_name,
        TemplateId='t1ada6f282d29742dxxxxxxxxxxxxx',
        InputObject='normal.mp4',
        OutputBucket=bucket_name,
        OutputRegion='ap-chongqing',
        OutputObject='result.txt',
```

```
    )
    print(response)
    return response

def ci_create_asr_jobs():
    # 创建语音识别异步任务(使用参数)
    body = {
        'EngineModelType': '16k_zh',
        'ChannelNum': '1',
        'ResTextFormat': '1',
    }
    response = client.ci_create_asr_job(
        Bucket=bucket_name,
        InputObject='normal.mp4',
        OutputBucket=bucket_name,
        OutputRegion='ap-chongqing',
        OutputObject='result.txt',
        SpeechRecognition=body
    )
    print(response)
    return response
```

参数说明

参数名称	描述	类型	是否必选
Bucket	对象所在存储桶	String	是
QueueId	任务所在的队列 ID，开通语音识别服务后自动生成，请使用 查询队列 获取或通过 万象控制台 在存储桶中查询	String	否
InputObject	待操作的文件对象，InputObject与Url参数需传入其中一个，当都传入时，仅处理InputObject指向的文件	String	否
Url	外网可下载的Url，InputObject与Url参数需传入其中一个，当都传入时，仅处理InputObject指向的文件	String	否
OutputRegion	存储桶的地域	String	是
OutputBucket	存储结果的存储桶	String	是
OutputObject	输出文件路径	String	是
TemplateId	模板ID	String	否

SpeechRecognition	语音识别参数，详见 提交语音识别任务请求详情 中的 SpeechRecognition 详情	dict	否
CallBack	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
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
CallBackMqConfig	任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填，详见 CallBackMqConfig 详情	dict	否

返回结果说明

```
{
    'JobsDetail': {
        'Code': 'Success',
        'CreationTime': '2022-09-14T22:31:38+0800',
        'EndTime': '-',
        'Input': {'BucketId': 'test-12539xxxxx', 'Object': 'normal.mp4',
'Region': 'ap-chongqing'},
        'JobId': 'sf1398ed0343911eda6d733209028ff12',
        'Message': None,
        'Operation': {
            'JobLevel': '0',
            'Output': {'Bucket': 'test-12539xxxxx', 'Object': 'result.txt',
'Region': 'ap-chongqing'},
            'TemplateId': 't1ada6f282d29742db83244e085exxxxxx',
            'TemplateName': 'name'
        },
        'QueueId': 'p7369ebcf8f93414c8250c966xxxxxxxx',
        'StartTime': '-',
        'State': 'Submitted',
        'Tag': 'SpeechRecognition'
    }
}
```

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
JobId	新创建任务的 ID	String
Tag	新创建任务的 Tag：DocProcess	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
QueueId	任务所在的队列 ID，开通语音识别服务后自动生成，请使用 查询队列 获取或通过 万象控制台 在存储桶中查询	String
Input	该任务的输入文件路径	Container
Operation	该任务的规则	Container

查询指定语音识别任务

功能说明

ci_get_asr_job 用于查询指定的语音识别任务。

方法原型

```
def ci_get_asr_job(self, Bucket, JobID, **kwargs):
```

请求示例

```
def ci_get_asr_jobs():
    # 获取语音识别信息
    response = client.ci_get_asr_job(
        Bucket=bucket_name,
        JobID='s18a9xxxxxxxxxxxxxxxxxxxff1aa',
    )
    print(response)
    return response
```

参数说明

参数名称	描述	类型	是否必选

Bucket	对象所在存储桶	String	是
JobID	语音识别任务 ID	String	是

返回结果说明

```
{
    'JobsDetail': [{
        'Code': 'Success',
        'CreationTime': '2022-09-14T22:36:51+0800',
        'EndTime': '2022-09-14T22:37:01+0800',
        'Input': {'BucketId': 'testpic-1253960454', 'Object': 'gaobai.mp4',
'Region': 'ap-chongqing'},
        'JobId': 'sabfb88d6343a11ed8c375fe3d77ef1d7',
        'Message': None,
        'Operation': {
            'JobLevel': '0',
            'Output': {'Bucket': 'testpic-1253960454', 'Object': 'result.txt',
'Region': 'ap-chongqing'},
            'SpeechRecognitionResult': {
                'AudioTime': '215.637',
                'ObjectName': 'result.txt',
                'Result': '[0:0.160,1:0.640]  测试测试\n', 'ResultDetail': None
            },
            'TemplateId': 't1ada6f282d29742db83244e085e920b08',
            'TemplateName': 'name'
        },
        'QueueId': 'p7369ebcf8f93414c8250c9663e1dff5a',
        'StartTime': '2022-09-14T22:36:53+0800',
        'State': 'Success',
        'Tag': 'SpeechRecognition'
    }]
}
```

参数名称	描述	类型
JobsDetail	任务的详细信息，同 CreateSpeechRecognitionJobs 接口的 Response.JobsDetail 节点	Container
NonExistJobIds	查询的 ID 中不存在的任务，所有任务都存在时不返回	String

拉取符合条件的语音识别任务

功能说明

ci_list_doc_jobs 用于拉取符合条件的语音识别任务。

方法原型

```
def ci_list_asr_jobs(self, Bucket, QueueId=None, StartCreationTime=None,
                     EndCreationTime=None, OrderByTime='Desc', States='All',
                     Size=10, NextToken='', **kwargs):
```

请求示例

```
def ci_list_asr_jobs():
    # 获取语音识别任务信息列表
    response = client.ci_list_asr_jobs(
        Bucket=bucket_name,
        Size=10,
    )
    print(response)
    return response
```

参数说明

参数名称	描述	类型	是否必选
Bucket	任务所在存储桶	String	是
QueueId	任务所在的队列 ID，开通语音识别服务后自动生成，请使用 查询队列 获取或通过 万象控制台 在存储桶中查询	String	否
OrderByTime	Desc 或者 Asc。默认为 Desc	String	否
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，最大值为100	Int	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


```
{
    'JobsDetail': [{
        'Code': 'Success',
        'CreationTime': '2022-09-14T22:36:51+0800',
        'EndTime': '2022-09-14T22:37:01+0800',
        'Input': {'BucketId': 'test-125396xxxx', 'Object': 'gaobai.mp4',
'Region': 'ap-chongqing'},
        'JobId': 'sabfb88d6343a11ed8c375fe3dxxxxxxx',
        'Message': None,
        'Operation': {
            'JobLevel': '0',
            'Output': {'Bucket': 'test-125396xxxx', 'Object': 'result.txt',
'Region': 'ap-chongqing'},
            'SpeechRecognitionResult': {
                'AudioTime': '215.637',
                'ObjectName': 'result.txt',
                'Result': '[0:0.160,1:0.640]  测试测试。\n',
                'ResultDetail': None
            },
            'TemplateId': 't1ada6f282d29742db83244e085xxxxxx',
            'TemplateName': 'name'
        },
        'QueueId': 'p7369ebcf8f93414c8250c966xxxxxxx',
        'StartTime': '2022-09-14T22:36:53+0800',
        'State': 'Success',
        'Tag': 'SpeechRecognition'
    }],
    'NextToken': '15966'
}
```


参数名称	描述	类型
JobsDetail	任务的详细信息，同 CreateSpeechRecognitionJobs 接口中的 Response.JobsDetail 节点	Container
NextToken	翻页的上下文 Token	String

语音 bucket 接口

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于查询语音识别开通状态的 API 概览以及 SDK 示例代码。

API	说明
查询语音识别开通状态	用于查询存储桶是否已开通语音识别功能

查询语音识别开通状态

功能说明

ci_get_asr_bucket 接口用于查询存储桶是否已开通语音识别功能。

方法原型

```
def ci_get_asr_bucket(self, Regions='', BucketName='', BucketNames='',
PageNumber='', PageSize='', **kwargs)
```

请求示例

```
def ci_get_asr_bucket():
    # 查询语音识别开通状态
    response = client.ci_get_asr_bucket(
        Regions=region,
        # BucketName='demo',
        BucketNames=bucket_name,
        PageSize=1,
        PageNumber=1
    )
    print(response)
    return response
```

参数说明

参数名称	描述	类型
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
BucketNames	存储桶名称，以“,”分隔，支持多个存储桶，精确搜索	string

BucketName	存储桶名称前缀，前缀搜索	string
PageNumber	第几页	int
pageSize	每页个数	int

返回结果说明

```
{
    'TotalCount': '1',
    'RequestId': 'NjMyMWE2ZjZfZWM0YTYyNjRxxxxxxxxxxxx',
    'PageNumber': '1',
    'PageSize': '1',
    'AsrBucketList': [{
        'Name': 'test-125xxxxxxx',
        'CreateTime': '2022-08-11T00:00:00+0800',
        'Region': 'ap-chongqing',
        'AliasBucketId': None,
        'BucketId': 'test-125xxxxxxxx'
    }]
}
```

参数名称	描述	类型
RequestId	请求的唯一 ID	string
TotalCount	语音识别 Bucket 总数	int
PageNumber	当前页数，同请求中的 pageNumber	int
PageSize	每页个数，同请求中的 pageSize	int
AsrBucketList	语音识别 Bucket 列表	Container
BucketId	存储桶 ID	string
Name	存储桶名称，同 BucketId	string
Region	所在的地域	string
CreateTime	创建时间	string
AliasBucketId	存储桶别名	string

队列接口

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于语音识别队列接口的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
查询语音识别队列	查询语音识别队列	查询语音识别队列信息
更新语音识别队列	更新语音识别队列	更新语音识别队列信息

查询语音识别队列

功能说明

ci_get_asr_queue 接口用于查询语音识别队列。

方法原型

```
def ci_get_asr_queue(self, Bucket, State='All', QueueIds='', PageNumber='',
PageSize='', **kwargs):
```

请求示例

```
def ci_get_asr_queue():
    # 查询语音识别队列信息
    response = client.ci_get_asr_queue(
        Bucket=bucket_name,
        PageNumber=1,
        PageSize=1,
    )
    print(response)
    return response
```

参数说明

参数名称	描述	类型	是否必选
Bucket	队列所在存储桶	String	是

QueueIds	队列 ID（开通语音识别服务后自动生成，可通过 万象控制台 在存储桶中查询），以“,”符号分割字符串	String	否
State	• Active 表示队列内的作业会被语音识别服务调度执行 • 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暂停状态，已经处于执行中的任务将继续执行，不受影响	String	否
PageNumber	第几页	int	否
PageSize	每页个数	int	否

返回结果说明

```
{
    'TotalCount': '1',
    'RequestId': 'NjMyMWExY2RfMTIwNjxxxxxxxxxxxxxxxxxx',
    'PageNumber': '1',
    'PageSize': '1',
    'QueueList': [{
        'QueueId': 'p4bdf22746bxxxxxxxxxxxxxxxxxxxxx',
        'Name': 'speech-queue',
        'State': 'Active',
        'NotifyConfig': {
            'Url': 'http://www.demo.callback.com',
            'Event': 'TaskFinish',
            'Type': 'Url',
            'State': 'On',
            'ResultFormat': 'JSON',
            'MqMode': None,
            'MqName': None,
            'MqRegion': None
        },
        'MaxSize': '10000',
        'MaxConcurrent': '10',
        'CreateTime': '2022-08-11T14:42:01+0800',
        'UpdateTime': '2022-09-14T11:18:08+0800',
        'BucketId': 'test-125xxxxxxxxx',
        'Category': 'Speeching'
    }]
}
```

参数名称	描述	类型

RequestId	请求的唯一 ID	String
TotalCount	队列总数	Int
PageNumber	当前页数，同请求中的 pageNumber	Int
PageSize	每页个数，同请求中的 pageSize	Int
QueueList	队列数组	Container
NonExistPIDs	不存在的队列 ID 列表	String 数组
QueueId	队列 ID，开通语音识别服务后自动生成，请使用 查询队列 获取或通过 万象控制台 在存储桶中查询	String
Name	队列名字	String
State	当前状态，Active 或者 Paused	String
NotifyConfig	回调配置	Container
MaxSize	队列最大长度	Int
MaxConcurrent	当前队列最大并行执行的任务数	Int
UpdateTime	更新时间	String
CreateTime	创建时间	String
Url	回调地址	String
State	开关状态，On 或者 Off	String
Type	回调类型，Url	String
Event	触发回调的事件	String

更新语音识别队列

功能说明

ci_update_asr_queue 接口用于更新语音识别队列。

方法原型

```
def ci_update_asr_queue(self, Bucket, QueueId, Request={}, **kwargs):
```

请求示例

```
def ci_put_asr_queue():
    # 更新语音识别队列信息
    body = {
        'Name': 'speech-queue',
        'QueueID': 'p4bdf22xxxxxxxxxxxxxxxxxxxxxxxf1',
        'State': 'Active',
        'NotifyConfig': {
            'Type': 'Url',
            'Url': 'http://www.demo.callback.com',
            'Event': 'TaskFinish',
            'State': 'On',
            'ResultFormat': 'JSON'
        }
    }
    response = client.ci_update_asr_queue(
        Bucket=bucket_name,
        QueueId='p4bdf22xxxxxxxxxxxxxxxxxxxxxxxf1',
        Request=body,
        ContentType='application/xml'
    )
    print(response)
    return response
```

参数说明

参数名称	描述	类型	是否必选
Bucket	存储桶名称	String	是
QueueID	队列 ID，开通语音识别服务后自动生成，请使用 查询队列 获取或通过 万象控制台 在存储桶中查询	String	是
Request	更新队列配置请求体	String	是

返回结果说明

```
{
    'RequestId': 'NjMyMWEzMTFfZWMxxxxxxxxxxxxxxxxxxx',
    'Queue': [{
        'QueueId': 'p4bdf22746b0xxxxxxxxxxxxxxxxx',
```

```
        'Name': 'speech-queue',
        'State': 'Active',
        'NotifyConfig': {
            'Url': 'http://www.demo.callback.com',
            'Event': 'TaskFinish',
            'Type': 'Url',
            'State': 'On',
            'ResultFormat':
            'JSON',
            'MqMode': None,
            'MqName': None,
            'MqRegion': None
        },
        'MaxSize': '10000',
        'MaxConcurrent': '10',
        'CreateTime': '2022-08-11T14:42:01+0800',
        'UpdateTime': '2022-09-14T11:18:08+0800',
        'BucketId': 'test-125xxxxxxxxxx',
        'Category': 'Speeching'
    }]
}
```

参数名称	描述	类型
RequestId	请求的唯一 ID	dict
Queue	队列信息，详情请参见 查询语音识别队列 中的 QueueList	dict

服务和队列配置
开通AI内容识别服务

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于开通AI内容识别服务的相关 API 概览以及 SDK 示例代码。

API	操作描述
开通AI内容识别服务	开通AI内容识别服务

开通AI内容识别服务

功能说明

本接口用于开通AI 内容识别（异步）服务并生成队列。

示例代码

```
def ci_open_ai_bucket():
    # 开通AI内容识别服务
    response, data = client.ci_open_ai_bucket(
        Bucket=bucket_name
    )
    print(response)
    print(data)
    return response, data
```

参数说明

调用 ci_open_ai_bucket 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是

返回结果说明

response 为响应头，data 为开通AI内容识别服务结果，数据类型都为 dict。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

响应体具体数据内容如下：

参数名称	参数描述	类型
RequestId	请求的唯一 ID	String
AiBucket	存储桶信息	Container

AiBucket 节点内容：

参数名称	参数描述	类型
BucketId	存储桶 ID	String
AliasBucketId	存储桶别名	
Name	存储桶名称，同 BucketId	String
Region	所在的地域	String
CreateTime	创建时间	String

查询AI内容识别服务

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于查询AI内容识别服务的相关 API 概览以及 SDK 示例代码。

API	操作描述
查询AI内容识别服务	查询AI内容识别服务

查询AI内容识别服务

功能说明

本接口用于查询已经开通AI 内容识别（异步）服务的存储桶。

示例代码

```
def ci_get_ai_bucket():
    # 查询AI内容识别服务

    data = client.ci_get_ai_bucket(
        Regions = "ap-chongqing",
        BucketNames = "test-12500000",
        BucketName = "test",
        PageNumber = "1",
        PageSize = "1"
    )
    print(data)
    return data
```

参数说明

调用 ci_get_ai_bucket 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Regions	地域信息，例如 ap-shanghai、ap-beijing，若查询多个地域以“,”分隔字符串，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
BucketNa	存储桶名称前缀，前缀搜索	Strin	否

me		g	
PageNumber	第几页	String	否
PageSize	每页个数，大于0且小于等于100的整数	String	否

返回结果说明

data 为查询AI内容识别服务结果，数据类型为 dict。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

响应体具体数据内容如下：

参数名称	参数描述	类型
RequestId	请求的唯一 ID	String
TotalCount	开通AI 内容识别（异步）的 Bucket 总数	Integer
PageNumber	当前页数，同请求中的 pageNumber	Integer
PageSize	每页个数，同请求中的 pageSize	Integer
AiBucketList	AI 内容识别（异步） Bucket 详情	Container Array

AiBucketList 节点内容：

参数名称	参数描述	类型
BucketId	存储桶 ID	String
Name	存储桶名称，同 BucketId	String
Region	所在的地域	String
CreateTime	创建时间	String

关闭AI内容识别服务

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于关闭AI内容识别服务的相关 API 概览以及 SDK 示例代码。

API	操作描述
关闭AI内容识别服务	关闭AI内容识别服务

关闭AI内容识别服务

功能说明

本接口用于关闭AI 内容识别（异步）服务并删除队列。

示例代码

```
def ci_close_ai_bucket():
    # 关闭AI内容识别服务
    response, data = client.ci_close_ai_bucket(
        Bucket=bucket_name
    )
    print(response)
    print(data)
    return response, data
```

参数说明

调用 ci_close_ai_bucket 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是

返回结果说明

response 为响应头，data 为关闭AI内容识别服务结果，数据类型都为 dict。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

响应体具体数据内容如下：

参数名称	参数描述	类型
RequestId	请求的唯一 ID	String
BucketName	存储桶名称	String

查询AI内容识别队列

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于查询AI内容识别队列的相关 API 概览以及 SDK 示例代码。

API	操作描述
查询AI内容识别队列	查询AI内容识别队列

查询AI内容识别队列

功能说明

本接口用于查询AI 内容识别（异步）队列。

示例代码

```
def ci_get_ai_queue():
    # 查询AI内容识别队列
    # demo地址：https://github.com/tencentyun/cos-python-sdk-v5/blob/master/demo/ai_recognition_demo.py#L546
    response = client.ci_get_ai_queue(
        Bucket=bucket_name,
        QueueIds='',
        State='All',
        PageNumber='',
        PageSize=''
    )
    print(response)
    return response
```

参数说明

调用 ci_get_ai_queue 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
QueueIds	队列 ID（开通语音识别服务后自动生成，可通过 万象控制台 在存储桶中查询），以“,”符号分割字符串	String	否

State	• Active 表示队列内的作业会被调度执行 • Paused 表示队列暂停，作业不再会被调度执行，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
PageNumber	第几页，默认值1	String	否
PageSize	每页个数，默认值10	String	否

返回结果说明

response 查询AI内容识别队列结果，数据类型都为 dict。

response内容

由于接口默认返回xml内容，解析结果都为string类型，以下描述为字段的真实类型。具体数据内容如下：

参数名称	参数描述	类型
RequestId	请求的唯一 ID	String
TotalCount	队列总数	Integer
PageNumber	当前页数，同请求中的 pageNumber	Integer
PageSize	每页个数，同请求中的 pageSize	Integer
QueueList	队列数组	Container Array
NonExistPIDs	不存在的队列 ID 列表	String Array

QueueList 节点内容：

参数名称	参数描述	类型
QueueId	队列 ID	String
Name	队列名字	String
State	当前状态，Active 或者 Paused	String
NotifyConfig	回调配置	Container
MaxSize	队列最大长度	Integer

MaxConcurrent	当前队列最大并行执行的任务数	Integer
Category	队列类型	String
UpdateTime	更新时间	String
CreateTime	创建时间	String
BucketId	队列对应的存储桶名称	String

NotifyConfig 节点内容：

参数名称	参数描述	类型
Url	回调地址	String
State	开关状态	String
Type	回调类型	String
Event	回调事件	String
ResultFormat	回调类型	String
MqMode	TDMQ 使用模式	String
MqRegion	TDMQ 所属园区	String
MqName	TDMQ 主题名称	String

更新AI内容识别队列

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于更新AI内容识别队列的相关 API 概览以及 SDK 示例代码。

API	操作描述
更新AI内容识别队列	更新AI内容识别队列

更新AI内容识别队列

功能说明

本接口用于更新AI 内容识别（异步）的队列。

示例代码

```
def ci_update_ai_queue():
    # 更新AI内容识别队列
    # demo地址：https://github.com/tencentyun/cos-python-sdk-v5/blob/master/demo/ai_recognition_demo.py#L559
    body = {
        # 队列名称，仅支持中文、英文、数字、_、-和*，长度不超过 128
        # 是否必传：是
        'Name': "test",
        # Active 表示队列内的作业会被调度执行；Paused 表示队列暂停，作业不再会被调度执行，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
        # 是否必传：是
        'State': "Active",
        # 回调配置
        # 是否必传：是
        'NotifyConfig': {
            # 回调开关OffOn
            # 是否必传：否
            'State': "On",
            # 回调事件，TaskFinish：任务完成；WorkflowFinish：工作流完成
            # 是否必传：否
            'Event': "TaskFinish",
            # 回调格式XMLJSON
            # 是否必传：否
            'ResultFormat': "XML",
            # 回调类型Url TDMQ
            # 是否必传：否
            'Type': "Url",
```

```
            # 回调地址，不能为内网地址。
            # 是否必传：否
            'Url': "http://www.test.cpm",
            # TDMQ 使用模式Topic：主题订阅Queue：队列服务
            # 是否必传：否
            'MqMode': "Topic",
            # TDMQ 所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）
            # 是否必传：否
            'MqRegion': "sh",
            # TDMQ 主题名称
            # 是否必传：否
            'MqName': "testTopic",
        },
    }
    response = client.ci_update_ai_queue(
        Bucket=bucket_name,
        QueueId="queueId",
        Request=body,
        ContentType="application/xml"
    )
    print(response)
    return response
```

参数说明

调用 ci_update_ai_queue 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
QueueId	队列 ID（开通内容识别服务后自动生成，可通过 万象控制台 在存储桶中查询）	String	是
Request	更新AI内容识别队列请求参数	Container	否

Request 节点内容：

参数名称	参数描述	类型	是否必填
Name	队列名称，仅支持中文、英文、数字、_、-和*，长度不超过 128	String	是
State	• Active 表示队列内的作业会被调度执行	String	是

	• Paused 表示队列暂停，作业不再会被调度执行，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		
NotifyConfig	回调配置	Container	是

NotifyConfig 节点内容：

参数名称	参数描述	类型	是否必填
State	回调开关Off/On	String	否
Event	回调事件 • TaskFinish：任务完成 • WorkflowFinish：工作流完成	String	否
ResultFormat	回调格式，XML/JSON	String	否
Type	回调类型，Url/TDMQ	String	否
Url	回调地址，不能为内网地址。	String	否
MqMode	TDMQ 使用模式 • Topic：主题订阅 • Queue: 队列服务	String	否
MqRegion	TDMQ 所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）	String	否
MqName	TDMQ 主题名称	String	否

返回结果说明

response 为更新AI内容识别队列结果，数据类型为 dict。

response内容

由于接口默认返回xml内容，解析结果都为string类型，以下描述为字段的真实类型。具体数据内容如下：

参数名称	参数描述	类型
RequestId	请求的唯一 ID	String
Queue	队列信息	Container

Queue 节点内容：

参数名称	参数描述	类型
QueueId	队列 ID	String
Name	队列名字	String
State	当前状态，Active 或者 Paused	String
NotifyConfig	回调配置，同 Request.NotifyConfig	Container
MaxSize	队列最大长度	Integer
MaxConcurrent	当前队列最大并行执行的任务数	Integer
Category	队列类型	String
UpdateTime	更新时间	String
CreateTime	创建时间	String

NotifyConfig 节点内容：

参数名称	参数描述	类型
State	回调开关 • Off • On	String
Event	回调事件 • TaskFinish：任务完成 • WorkflowFinish：工作流完成	String
ResultFormat	回调格式 • XML • JSON	String
Type	回调类型 • Url • TDMQ	String
Url	回调地址，不能为内网地址。	String
MqMode	TDMQ 使用模式 • Topic：主题订阅 • Queue: 队列服务	String

MqRegion	TDMQ 所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）	String
MqName	TDMQ 主题名称	String

版权所有：腾讯云计算（北京）有限责任公司

图片二维码

最近更新时间：2024-12-27 16:37:07

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
        )
        # 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

返回结果说明

response 为响应头，data 为二维码识别结果，数据类型都为 dict。

```
# response
{
    'Content-Type': 'application/xml',
    'Content-Length': '1094',
    'Connection': 'keep-alive',
    'Date': 'Sat, 00 May 0000 00:00:00 GMT',
    'Etag': '"string"',
    'Server': 'tencent-ci',
    'X-Cos-Hash-Crc64ecma': '00000000000000000',
    'X-Cos-Storage-Class': 'STANDARD',
    'x-cos-request-id': 'request id'
}
# data
{
    'OriginalInfo': {
        'Key': 'example_key',
        'Location': 'test-1250000000.cos.ap-
chongqing.myqcloud.com/example_key',
        'ETag': '"1cb58f788804beac37769b9eb407c954"',
        'ImageInfo': {
            'Format': 'PNG',
            'Width': '400',
            'Height': '400',
            'Quality': '100',
            'Ave': '0xe0e0e0',
            'Orientation': '0',
```

```
            'FrameCount': '1'
        }
    },
    'ProcessResults': {
        'Object': {
            'Key': '1.jpg',
            'Location': 'test-1250000000.cos.ap-
chongqing.myqcloud.com/1.jpg',
            'Format': 'png',
            'Width': '400',
            'Height': '400',
            'Size': '594',
            'Quality': '80',
            'FrameCount': '1',
            'ETag': '"e7e22acd8f0cec7abe858748d641d266"',
            'CodeStatus': '1',
            'QRcodeInfo': {
                'CodeUrl': 'cloud.tencent.com',
                'CodeLocation': {
                    'Point': ['27,371', '27,27', '371,27', '371,371']
                }
            }
        }
    }
}
```

response 内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体 data

响应体具体数据内容如下：

参数名称	参数描述	类型
OriginalInfo	原图信息	Container
ProcessResults	图片处理结果	Container

OriginalInfo 节点内容：

节点名称	参数描述	类型
Key	原图文件名	String
Location	图片路径	String

Etag	图片etag信息	String
ImageInfo	原图图片信息	Container

ImageInfo 节点内容：

节点名称	参数描述	类型
Format	格式	String
Width	图片宽度	String
Height	图片高度	String
Quality	图片质量	String
Ave	图片主色调	String
Orientation	图片旋转角度	String
FrameCount	图片的帧数。静态图为1，动图为对应的帧数。	String

ProcessResults 节点内容：

节点名称	参数描述	类型
Object	每一个图片处理结果	Container

Object 节点内容：

节点名称	参数描述	类型
Key	文件名	String
Location	图片路径	String
Format	图片格式	String
Width	图片宽度	String
Height	图片高度	String
Size	图片大小	String
Quality	图片质量	String
Etag	图片etag信息	String
FrameCount	图片的帧数。静态图为1，动图为对应的帧数。	String
CodeStatus	二维码识别结果。0表示未识别到二维码，1表示识别到二维码	String

版权所有：腾讯云计算（北京）有限责任公司

QRcodeInfo	二维码识别结果，可能有多个	Container

QRcodeInfo 节点内容：

节点名称	参数描述	类型
CodeUrl	二维码的内容。可能识别不出	String
CodeLocation	图中识别到的二维码位置坐标	Container

codelocation 节点内容：

节点名称	参数描述	类型
Point	二维码坐标点	String

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,
    BarType=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
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`	String	是

	中，对象键为 doc/pic.jpg		
Cover	二维码覆盖功能。可为0或1。 • 0表示不开启二维码覆盖 • 1表示开启二维码覆盖 功能开启后，将对识别出的二维码覆盖上马赛克，默认值0	Int	是
BarType	二维码/条形码识别功能，将对识别出的二维码/条形码 覆盖马赛克。取值有0，1，2 。 • 0表示都识别 • 1表示识别二维码 • 2表示识别条形码 • 默认值0	Int	否

返回结果说明

response 为响应头，data 为二维码识别结果，数据类型都为 dict。

```
# response
{
    'Content-Type': 'application/xml',
    'Content-Length': '1094',
    'Connection': 'keep-alive',
    'Date': 'Sat, 00 May 0000 00:00:00 GMT',
    'Etag': '"string"',
    'Server': 'tencent-ci',
    'X-Cos-Hash-Crc64ecma': '00000000000000000',
    'X-Cos-Storage-Class': 'STANDARD',
    'x-cos-request-id': 'request id'
}
# data
{
    'CodeStatus': '1',
    'QRcodeInfo': {
        'CodeUrl': 'www.tencent.com',
        'CodeLocation': {
            'Point': ['23,375', '23,23', '375,23', '375,375']
         }
     }
}
```

response 内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体 data

响应体具体数据内容如下：

节点名称（关键字）	参数描述	类型
CodeStatus	二维码识别结果。0表示未识别到二维码，1表示识别到二维码	String
QRcodeInfo	二维码识别结果，可能有多个	Container
ResultImage	处理后的图片 base64数据，请求参数 cover 为1时返回	String

QRcodeInfo 节点内容：

节点名称（关键字）	参数描述	类型
CodeUrl	二维码的内容。可能识别不到内容	String
CodeLocation	图中识别到的二维码位置坐标	Container

CodeLocation 节点内容：

节点名称（关键字）	参数描述	类型
Point	二维码坐标点（X坐标,Y坐标）	String

版权所有：腾讯云计算（北京）有限责任公司

图片主体检测

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于图像主体检测的相关 API 概览以及 SDK 示例代码。

API	操作描述
图像主体检测	图像主体检测

图像主体检测

功能说明

腾讯云数据万象通过 AIObjectDetect 接口实现对图片内不同主体的识别，返回图片中识别到主体 的名称、坐标、置信度分值。

示例代码

```
from qcloud_cos import AIRecognitionClient
from qcloud_cos import CosConfig

# 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.cloud.tencent.com/cam/capi
secret_id = os.environ["SECRETID"]
# 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.cloud.tencent.com/cam/capi
secret_key = os.environ["SECRETKEY"]
# 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
region = 'ap-chongqing'
# COS支持的所有region列表参见https://www.qcloud.com/document/product/436/6224
# 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://cloud.tencent.com/document/product/436/14048
token = None
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key,
Token=token)
client = AIRecognitionClient(config)

def cos_create_ai_object_detect_job():
    # 图像主体检测
    # demo地址: https://github.com/tencentyun/cos-python-sdk-
v5/blob/master/demo/ai_recognition_demo.py#L57
    response, data = client.cos_create_ai_object_detect_job(
```

```
        Bucket=bucket_name,
        ObjectKey = "objectKey",
    )
    print(response)
    print(data)
    return response, data
```

参数说明

调用 cos_create_ai_object_detect_job 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
Object Key	设置对象名称	String	否
Detect Url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey。detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`	String	否

返回结果说明

response 为响应头，data 为图像主体检测结果，数据类型都为 dict。

response 内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体 data

响应体具体数据内容如下：

参数名称	参数描述	类型
Status	主体识别结果状态。0表示未识别到，1表示识别到。	Integer
DetectMultiObj	主体识别结果，可能有多个。	Container Array

DetectMultiObj 节点内容：

参数名称	参数描述	类型

Name	识别到主体的名称。	String
Confidence	识别到主体的置信度，取值范围为[0-100]。值越高概率越大。	Integer
Location	图中识别到主体的坐标。	Container

Location 节点内容：

参数名称	参数描述	类型
X	识别主体框左上角横坐标。	Integer
Y	识别主体框左上角纵坐标。	Integer
Width	识别主体框高度。	Integer
Height	识别主体框高度。	Integer

版权所有：腾讯云计算（北京）有限责任公司

图片上色

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于图片上色的相关 API 概览以及 SDK 示例代码。

API	操作描述
图片上色	图片上色

图片上色

功能说明

腾讯云数据万象通过 AIImageColoring 接口对黑白图像进行上色

示例代码

```
def cos_ai_image_coloring():
    # 图片上色
    response, data = client.cos_ai_image_coloring(
        Bucket=bucket_name,
        ObjectKey="test.jpeg",
        # DetectUrl="https://test-125000000.cos.ap-
chongqing.myqcloud.com/test.jpeg"
        Stream=True
    )
    data.get_stream_to_file('result.jpg')
    print(response)
    return response, data
```

参数说明

调用 cos_ai_image_coloring 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
ObjectKey	数据万象处理能力，图片上色参固定为AIImageColoring。	String	否
Detect	待上色图片url，与ObjectKey二选其一，如果同时存在，则默认以	Strin	否

Url	ObjectKey为准	g	
Stream	是否以流式方式接收响应内容，默认为True	Boolean	否

返回结果说明

response 为响应头， 数据类型为 dict。 data 为图片上色处理后的图片。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

[pic-content]

图片标签

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于图片标签的相关 API 概览以及 SDK 示例代码。

API	操作描述
图片标签	图片标签

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。

示例代码

```
def ci_image_detect_label():
    # 图片标签
    # demo地址: https://github.com/tencentyun/cos-python-sdk-v5/blob/master/demo/ci_image.py#L586
    response = client.ci_image_detect_label(
        Bucket=bucket_name,
        Key='test.jpg',
        Scenes='camera',
        # DetectUrl='https://test-125000000.cos.ap-chongqing.myqcloud.com/test.jpeg',
    )
    print(response)
    return response
```

参数说明

调用 cos_detect_label 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
Key	设置对象名称。Key与DetectUrl必选一个	String	否

Scenes	本次调用支持的识别场景，可选值如下： • web，针对网络图片优化； • camera，针对手机摄像头拍摄图片优化； • album，针对手机相册、网盘产品优化； • news，针对新闻、资讯、广电等行业优化； 如果不传此参数，则默认为 camera。 支持多场景（scenes）一起检测，以，分隔。例如，使用 scenes=web，camera 即对一张图片使用两个模型同时检测，输出两套识别结果	String	否
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey。detect-url 示例：`http://www.example.com/abc.jpg` 。Key与DetectUrl必选一个	String	否

返回结果说明

response 为响应头，data 为图片标签结果，数据类型都为 dict。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

由于接口默认返回xml内容，解析结果都为string类型，以下描述为字段的真实类型。具体数据内容如下：

参数名称	参数描述	类型
WebLabels	Web 网络版标签结果数组。如未选择 web 场景，则不存在该项。此字段可能为空，表示取不到有效值	Container
CameraLabels	Camera 摄像头版标签结果数组。如未选择 camera 场景，则不存在该项。此字段可能为空，表示取不到有效值	Container
AlbumLabels	Album 相册版标签结果数组。如未选择 album 场景，则不存在该项。此字段可能为空，表示取不到有效值	Container
NewsLabels	News 新闻版标签结果数组。如未选择 news 场景，则不存在该项。此字段可能为空，表示取不到有效值	Container

WebLabels，CameraLabels，AlbumLabels，NewsLabels 节点内容：

参数名称	参数描述	类型
Labels	标签信息	Container Array

Labels节点内容：

参数名称	参数描述	类型
Confidence	该标签的置信度分数，分数越高则该标签准确度越高	Integer
Name	识别出的图片标签	String
FirstCategory	标签的一级分类	String
SecondCategory	标签的二级分类	String

图像增强

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于图像增强的相关 API 概览以及 SDK 示例代码。

API	操作描述
图像增强	图像增强

图像增强

功能说明

腾讯云数据万象通过 AIEnhanceImage 接口对图像进行增强处理。

示例代码

```
def cos_ai_enhance_image():
    # 图像增强

    response, data = client.cos_ai_enhance_image(
        Bucket=bucket_name,
        ObjectKey="test.jpg",
        Denoise=3,
        Sharpen=3,
        # DetectUrl="https://test-125000000.cos.ap-
chongqing.myqcloud.com/test.jpeg",
        IgnoreError=0,
        Stream=True
    )
    data.get_stream_to_file('result.jpg')
    print(response)
    return response, data
```

参数说明

调用 cos_ai_enhance_image 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是

ObjectKey	设置对象名称。ObjectKey与DetectUrl必选一个	String	否
Denoise	去噪强度值，取值范围为 0 - 5 之间的整数，值为 0 时不进行去噪操作，默认值为3	Integer	否
Sharpen	锐化强度值，取值范围为 0 - 5 之间的整数，值为 0 时不进行锐化操作，默认值为3	Integer	否
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了detect-url 时，后台会处理 detect-url链接，无需再填写 ObjectKey ，detect-url 示例：`http://www.example.com/abc.jpg` 。ObjectKey与DetectUrl必选一个	String	否
IgnoreError	当此参数为1时，针对文件过大等导致处理失败的场景，会直接返回原图而不报错	Integer	否
Stream	是否以流式方式接收响应内容，默认为 True	Boolean	否

返回结果说明

response 为响应头， 数据类型为 dict。 data 为图像增强处理后的图片。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

[pic-content]

图像智能裁剪

最近更新时间：2025-06-17 20:30:52

简介

本文档提供关于图像智能裁剪的相关 API 概览以及 SDK 示例代码。

API	操作描述
图像智能裁剪	图像智能裁剪

功能说明

腾讯云数据万象通过 AIImageCrop 接口对图像进行智能裁剪，支持持久化、云上处理及下载时处理。

示例代码

```
from qcloud_cos import AIRecognitionClient
from qcloud_cos import CosConfig

# 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.cloud.tencent.com/cam/capi
secret_id = os.environ["SECRETID"]
# 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.cloud.tencent.com/cam/capi
secret_key = os.environ["SECRETKEY"]
# 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
region = 'ap-chongqing'
# COS 支持的所有 region 列表参见 https://www.qcloud.com/document/product/436/6224
# 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://cloud.tencent.com/document/product/436/14048
token = None
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key,
Token=token)
client = AIRecognitionClient(config)
bucket_name = 'test-1250000000'

def cos_ai_image_crop():
    # 图像智能裁剪
    response, data = client.cos_ai_image_crop(
        Bucket=bucket_name,
        ObjectKey="test.jpg",
        # DetectUrl="https://test-125000000.cos.ap-
chongqing.myqcloud.com/test.jpeg"
```

```
        Width=100,
        Height=100,
        Fixed=1,
        IgnoreError=0
    )
    data.get_stream_to_file('result.jpg')
    print(response)
    return response, data
```

参数说明

调用 cos_ai_image_crop 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
ObjectKey	设置对象名称	String	否
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey detect-url 示例：`http://www.example.com/abc.jpg`	String	否
Width	• 需要裁剪区域的宽度，与 height 共同组成所需裁剪的图片宽高比例 • 输入数字请大于0、小于图片宽度的像素值	Integer	是
Height	• 需要裁剪区域的高度，与 width 共同组成所需裁剪的图片宽高比例 • 输入数字请大于0、小于图片高度的像素值 • width : height 建议取值在[1, 2.5]之间，超过这个范围可能会影响效果	Integer	是
Fixed	是否严格按照 width 和 height 的值进行输出。 • 取值为0时，宽高比例（width : height）会简化为最简分数，即如果width输入10、height输入20，会简化为1：2 • 取值为1时，输出图片的宽度等于 width，高度等于 height；默认值为0	Integer	否
IgnoreError	当此参数为1时，针对文件过大等导致处理失败的场景，会直接返回原图而不报错	Integer	否

返回结果说明

response 为响应头， 数据类型为 dict。data 为图像智能裁剪后的图片。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

[pic-content]

图像修复

最近更新时间：2024-12-27 16:37:07

简介

本文档提供关于图像修复的相关 API 概览以及 SDK 示例代码。

API	操作描述
图像修复	图像修复

图像修复

功能说明

腾讯云数据万象通过 ImageRepair 接　能检测并擦除图片中常见的标志,并对擦除部分进行智能修复，此功能需携带签名。

示例代码

```
def cos_image_repair():
    # 图像修复
    # demo地址: https://github.com/tencentyun/cos-python-sdk-v5/blob/master/demo/ai_recognition_demo.py#L111
    mask_pic = 'https://test-12500000.cos.ap-chongqing.myqcloud.com/mask.jpg'
    mask_poly = '[[[100, 200], [1000, 200], [1000, 400], [100, 400]]]'
    if len(mask_pic) != 0:
        mask_pic = base64.b64encode(mask_pic.encode('utf-8')).decode('utf-8')
    if len(mask_poly) != 0:
        mask_poly = base64.b64encode(mask_poly.encode('utf-8')).decode('utf-8')
    response, data = client.cos_image_repair(
        Bucket=bucket_name,
        ObjectKey="xiufu.jpg",
        # MaskPic=mask_pic,
        MaskPoly=mask_poly
    )
    data.get_stream_to_file('result.jpg')
    print(response)
    return response, data
```

参数说明

调用 cos_image_repair 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
ObjectKey	设置对象名称	String	否
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey。detect-url 示例：`http://www.example.com/abc.jpg`	String	否
MaskPic	遮罩（图像中需要去除的标志位置）图片地址，私有图片需携带签名，需要经过 URL 安全的 Base64 编码 。例如，遮罩图片为 `http://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/shuiyin_2.png` ，则该处编码后的字符串为 `aHR0cDovL2V4YW1wbGVidWNrZXQtMTI1MDAwMDAwMC5jb3MuYXAtZ3Vhbmd6aG91Lm15cWNsb3VkLmNvbS9zaHVpeWluXzIucG5n`	String	否
MaskPoly	例如： [[[608, 794], [1024, 794], [1024, 842], [608, 842]],[[1295, 62], [1295, 30], [1597, 32],[1597,64]]] ，顺时针输　多边形的每个点的坐标,每个多边形: [[x1, y1], [x2, y2]...] , 形式为三维矩阵（多个多边形： [多边形1,多边形2] ）或　维矩阵（单个多边形），且需要经过 URL 安全的 Base64 编码。MaskPoly 同时与 MaskPic 填写时，优先采　MaskPic 的值。	String	否

返回结果说明

response 为响应头， 数据类型为 dict。 data 为图像修复结果图片。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

[pic-content]

智能抠图
通用抠图

最近更新时间：2025-06-17 20:30:52

简介

本文档提供关于通用抠图的相关 API 概览以及 SDK 示例代码。

API	操作描述
通用抠图	通用抠图

功能说明

数据万象通用抠图 AIPicMatting 接口检测图片中的主体信息，自动分割图片背景，生成只包含主体信息的图片，支持持久化、云上处理及下载时处理。

示例代码

```
from qcloud_cos import AIRecognitionClient
from qcloud_cos import CosConfig

# 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.cloud.tencent.com/cam/capi
secret_id = os.environ["SECRETID"]
# 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.cloud.tencent.com/cam/capi
secret_key = os.environ["SECRETKEY"]
# 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
region = 'ap-chongqing'
# COS 支持的所有 region 列表参见 https://www.qcloud.com/document/product/436/6224
# 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://cloud.tencent.com/document/product/436/14048
token = None
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key,
Token=token)
client = AIRecognitionClient(config)
bucket_name = 'test-1250000000'

def cos_ai_pic_matting():
    # 通用抠图
    response, data = client.cos_ai_pic_matting(
```

```
        Bucket=bucket_name,
        ObjectKey="heibai.jpeg",
        # DetectUrl="https://test-125000000.cos.ap-
chongqing.myqcloud.com/test.jpeg"
        CenterLayout=1,
        PaddingLayout="10x10",
        Stream=True
    )
    data.get_stream_to_file('result.jpg')
    print(response)
    return response, data
```

参数说明

调用 cos_ai_pic_matting 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
ObjectKey	设置对象名称，ObjectKey 与 DetectUrl 中必传一个	String	否
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey。detect-url 示例：`http://www.example.com/abc.jpg`，ObjectKey 与 DetectUrl 中必传一个	String	否
CenterLayout	抠图主体居中显示；值为1时居中显示，值为0不做处理，默认为0	Integer	否
PaddingLayout	将处理后的图片四边进行留白，形式为 padding-layout = <dx> x <dy>，左右两边各进行 dx 像素的留白，上下两边各进行 dy 像素的留白，例如：padding-layout = 20 x 10，默认不进行留白操作，dx、dy 最大值为1000像素。	String	否
Stream	是否以流式方式接收响应内容，默认为 True	Boolean	否

返回结果说明

response 为响应头， 数据类型为 dict。 data 为通用抠图结果。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

[pic-content]

版权所有：腾讯云计算（北京）有限责任公司

商品抠图

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于商品抠图的相关 API 概览以及 SDK 示例代码。

API	操作描述
商品抠图	商品抠图

商品抠图

功能说明

腾讯云数据万象通过 GoodsMatting 接口检测图片中的商品信息，生成只包含商品信息的图片，支持持久化、云上处理及下载时处理。

示例代码

```
def cos_goods_matting():
    # 商品抠图
    # demo 地址：https://github.com/tencentyun/cos-python-sdk-v5/blob/master/demo/ai_recognition_demo.py#L69
    response, data = client.cos_goods_matting(
        Bucket=bucket_name,
        ObjectKey = "objectKey",
        DetectUrl = "https://test-125000000.cos.ap-chongqing.myqcloud.com/test.jpeg",
        CenterLayout=1,
        PaddingLayout='300x300',
        Stream=True
    )
    data.get_stream_to_file('test.jpg')
    print(response)
    return response, data
```

参数说明

调用 cos_goods_matting 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是

ObjectKey	设置对象名称。ObjectKey与DetectUrl必选一个	String	否
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey。detect-url 示例：`http://www.example.com/abc.jpg`，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg` 。ObjectKey与DetectUrl必选一个	String	否
CenterLayout	抠图商品居中显示； 值为1时居中显示，值为0时不作处理，默认为0	Int	否
PaddingLayout	将处理后的图片四边进行留白，形式为 padding-layout=<dx>x<dy>，左右两边各进行 dx 像素的留白，上下两边各进行 dy 像素的留白，例如：padding-layout=20x10，默认不进行留白操作，dx、dy最大值为1000像素	String	否
Stream	是否以流式方式接收响应内容，默认为 True	Boolean	否

返回结果说明

response 为响应头， 数据类型为 dict。 data 为商品抠图结果。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

[pic-content]

人像抠图

最近更新时间：2025-06-17 20:30:52

简介

本文档提供关于人像抠图的相关 API 概览以及 SDK 示例代码。

API	操作描述
人像抠图	人像抠图

功能说明

腾讯云数据万象通过 AIPortraitMatting 接口检测图片中的人像主体信息，智能分割图像背景，生成只包含人像主体信息的图片，支持持久化、云上处理及下载时处理。

示例代码

```
from qcloud_cos import AIRecognitionClient
from qcloud_cos import CosConfig

# 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.cloud.tencent.com/cam/capi
secret_id = os.environ["SECRETID"]
# 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.cloud.tencent.com/cam/capi
secret_key = os.environ["SECRETKEY"]
# 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
region = 'ap-chongqing'
# COS 支持的所有 region 列表参见 https://www.qcloud.com/document/product/436/6224
# 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://cloud.tencent.com/document/product/436/14048
token = None
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key,
Token=token)
client = AIRecognitionClient(config)
bucket_name = 'test-1250000000'

def cos_ai_portrait_matting():
    # 人像抠图
    response, data = client.cos_ai_portrait_matting(
        Bucket=bucket_name,
        ObjectKey="relian.jpeg",
        # DetectUrl="https://test-125000000.cos.ap-
```

```
chongqing.myqcloud.com/test.jpeg"
        CenterLayout=1,
        PaddingLayout="10x10",
        Stream=True
    )
    data.get_stream_to_file('result.jpg')
    print(response)
    return response, data
```

参数说明

调用 cos_ai_portrait_matting 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
ObjectKey	设置对象名称，ObjectKey 与 DetectUrl 中必传一个	String	否
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey。detect-url 示例：`http://www.example.com/abc.jpg`，ObjectKey 与 DetectUrl 中必传一个	String	否
CenterLayout	抠图主体居中显示；值为1时居中显示，值为0不做处理，默认为0	Integer	否
PaddingLayout	将处理后的图片四边进行留白，形式为 padding-layout = <dx> x <dy>，左右两边各进行 dx 像素的留白，上下两边各进行 dy 像素的留白，例如：padding-layout = 20 x 10。默认不进行留白操作，dx、dy 最大值为1000像素。	String	否
Stream	是否以流式方式接收响应内容，默认为 True	Boolean	否

返回结果说明

response 为响应头， 数据类型为 dict。data 为人像抠图结果。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

[pic-content]

版权所有：腾讯云计算（北京）有限责任公司

人脸核身
身份证识别

最近更新时间：2025-06-17 20:30:52

简介

本文档提供关于身份证识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
身份证识别	身份证识别

功能说明

支持中国大陆居民二代身份证正反面所有字段的识别，包括姓名、性别、民族、出生日期、住址、公民身份证号、签发机关、有效期限；具备身份证照片、人像照片的裁剪功能和翻拍、PS、复印件告警功能，以及边框和框内遮挡告警、临时身份证告警和身份证有效期不合法告警等扩展功能。

示例代码

```
from qcloud_cos import AIRecognitionClient
from qcloud_cos import CosConfig

# 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.cloud.tencent.com/cam/capi
secret_id = os.environ["SECRETID"]
# 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.cloud.tencent.com/cam/capi
secret_key = os.environ["SECRETKEY"]
# 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
region = 'ap-chongqing'
# COS 支持的所有 region 列表参见
https://cloud.tencent.com/document/product/436/6224
# 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://cloud.tencent.com/document/product/436/14048
token = None
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key,
Token=token)
client = AIRecognitionClient(config)
bucket_name = 'test-1250000000'

def cos_ai_id_card_ocr():
    # 身份证识别
```

```
    response, data = client.cos_ai_id_card_ocr(
        Bucket=bucket_name,
        ObjectKey="test.jpeg",
        # CardSide="FRONT",
        Config='{"CropIdCard":true,"CropPortrait":true}'
    )
    print(response)
    print(data)
    return response, data
```

参数说明

调用 cos_aiid_card_ocr 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
ObjectKey	设置对象名称	String	是
CardSide	• FRONT：身份证有照片的一面（人像面） • BACK：身份证有国徽的一面（国徽面） • 该参数如果不填，将为您自动判断身份证正反面	String	否
Config	以下可选字段均为 bool 类型，默认 false： • CropIdCard，身份证照片裁剪（去掉证件外多余的边缘、自动矫正拍摄角度） • CropPortrait，人像照片裁剪（自动抠取身份证头像区域） • CopyWarn，复印件告警 • BorderCheckWarn，边框和框内遮挡告警 • ReshootWarn，翻拍告警 • DetectPsWarn，PS 检测告警 • TempIdWarn，临时身份证告警 • InvalidDateWarn，身份证有效日期不合法告警 • Quality，图片质量分数（评价图片的模糊程度） • MultiCardDetect，是否开启多卡证检测 参数设置方式参考：Config = {"CropIdCard":true,"CropPortrait":true}	String	否

返回结果说明

response 为响应头，data 为身份证识别结果，数据类型都为 dict。

response 内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体 data

响应体具体数据内容如下：

参数名称	参数描述	类型
IdInfo	身份证识别信息	Container
AdvancedInfo	扩展信息，不请求则不返回	Container

AdvancedInfo 节点内容：

参数名称	参数描述	类型
IdCard	裁剪后身份证照片的 Base64编码，设置 Config.CropIdCard 为 true 时返回	String
Portrait	身份证头像照片的 Base64编码，设置 Config.CropPortrait 为 true 时返回	String
Quality	图片质量分数，设置 Config.Quality 为 true 时返回（取值范围：0~100，分数越低越模糊，建议阈值≥50）	String
BorderCodeValue	身份证边框不完整告警阈值分数，设置 Config.BorderCheckWarn 为 true 时返回（取值范围：0~100，分数越低边框遮挡可能性越低，建议阈值≥50）	String
WarnInfos	告警信息，仅当存在异常时返回，Code 告警码列表和释义： • 9100身份证有效日期不合法告警 • 9101身份证边框不完整告警 • 9102身份证复印件告警 • 9103身份证翻拍告警 • 9104临时身份证告警 • 9105身份证框内遮挡告警 • 9106身份证 PS 告警 可能存在多个 WarnInfos	String

IdInfo 节点内容：

参数名称	参数描述	类型
Name	姓名（人像面）	String
Sex	性别（人像面）	String

Nation	民族（人像面）	String
Birth	出生日期（人像面）	String
Address	地址（人像面）	String
IdNum	身份证号（人像面）	String
Authority	发证机关（国徽面）	String
ValidDate	证件有效期（国徽面）	String

获取动作顺序

最近更新时间：2025-06-17 20:30:52

简介

本文档提供关于获取动作顺序的相关 API 概览以及 SDK 示例代码。

API	操作描述
获取动作顺序	获取动作顺序

功能说明

使用动作活体检测模式前，需调用本接口获取动作顺序。

示例代码

```
from qcloud_cos import AIRecognitionClient
from qcloud_cos import CosConfig

# 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.cloud.tencent.com/cam/capi
secret_id = os.environ["SECRETID"]
# 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.cloud.tencent.com/cam/capi
secret_key = os.environ["SECRETKEY"]
# 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
region = 'ap-chongqing'
# COS 支持的所有 region 列表参见 https://www.qcloud.com/document/product/436/6224
# 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://cloud.tencent.com/document/product/436/14048
token = None
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key,
Token=token)
client = AIRecognitionClient(config)
bucket_name = 'test-1250000000'

def cos_get_action_sequence():
    # 获取动作顺序
    response, data = client.cos_get_action_sequence(
        Bucket=bucket_name,
    )
    print(response)
    print(data)
```

```
    return response, data
```

参数说明

调用 cos_get_action_sequence 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是

返回结果说明

response 为响应头，data 为获取动作顺序结果，数据类型都为 dict。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

响应体具体数据内容如下：

参数名称	参数描述	类型
ActionSequence	动作顺序（2，1 or 1，2），1代表张嘴，2代表闭眼	String

获取数字验证码

最近更新时间：2025-06-17 20:30:52

简介

本文档提供关于获取数字验证码的相关 API 概览以及 SDK 示例代码。

API	操作描述
获取数字验证码	获取数字验证码

功能说明

使用数字活体检测模式前，需调用本接口获取数字验证码。

示例代码

```
from qcloud_cos import AIRecognitionClient
from qcloud_cos import CosConfig

# 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.cloud.tencent.com/cam/capi
secret_id = os.environ["SECRETID"]
# 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.cloud.tencent.com/cam/capi
secret_key = os.environ["SECRETKEY"]
# 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
region = 'ap-chongqing'
# COS 支持的所有 region 列表参见 https://www.qcloud.com/document/product/436/6224
# 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://cloud.tencent.com/document/product/436/14048
token = None
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key,
Token=token)
client = AIRecognitionClient(config)
bucket_name = 'test-1250000000'

def cos_get_live_code():
    # 获取数字验证码
    response, data = client.cos_get_live_code(
        Bucket=bucket_name,
    )
    print(response)
    print(data)
```

```
    return response, data
```

参数说明

调用 cos_get_live_code 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是

返回结果说明

response 为响应头，data 为获取数字验证码结果，数据类型都为 dict。

response 内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体 data

响应体具体数据内容如下：

参数名称	参数描述	类型
LiveCode	数字验证码，如：1234	String

活体人脸核身

最近更新时间：2025-06-17 20:30:52

简介

本文档提供关于活体人脸核身的相关 API 概览以及 SDK 示例代码。

API	操作描述
活体人脸核身	活体人脸核身

功能说明

集成了活体检测和跟权威库进行比对的能力，传入一段视频和姓名、身份证号信息即可进行验证。对录制的自拍视频进行活体检测，从而确认当前用户为真人，可防止照片、视频、静态3D建模等各种不同类型的攻击。检测为真人后，再判断该视频中的人与权威库的证件照是否属于同一个人，实现用户身份信息核实。

示例代码

```
from qcloud_cos import AIRecognitionClient
from qcloud_cos import CosConfig

# 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.cloud.tencent.com/cam/capi
secret_id = os.environ["SECRETID"]
# 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.cloud.tencent.com/cam/capi
secret_key = os.environ["SECRETKEY"]
# 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
region = 'ap-chongqing'
# COS 支持的所有 region 列表参见 https://www.qcloud.com/document/product/436/6224
# 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://cloud.tencent.com/document/product/436/14048
token = None
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key,
Token=token)
client = AIRecognitionClient(config)
bucket_name = 'test-1250000000'

def cos_liveness_recognition():
    # 活体人脸核身
    response, data = client.cos_liveness_recognition(
        Bucket=bucket_name,
        ObjectKey="objectKey",
```

```
        IdCard="123456",
        Name="测试",
        LivenessType="LIP",
        ValidateData="1234",
        BestFrameNum=5
    )
    print(response)
    print(data)
    return response, data
```

参数说明

调用 cos_liveness_recognition 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
ObjectKey	设置对象名称	String	是
IdCard	身份证号	String	是
Name	姓名。中文请使用 UTF-8编码	String	是
LivenessType	• 活体检测类型，取值：LIP/ACTION/SILENT • LIP 为数字模式，ACTION 为动作模式，SILENT 为静默模式，三种模式选择一种传入	String	是
ValidateData	• 数字模式传参：数字验证码（1234），需先调用接口获取数字验证码 • 动作模式传参：传动作顺序（2，1 or 1，2），需先调用接口获取动作顺序 • 静默模式传参：空	String	否
BestFrameNum	需要返回多张最佳截图，取值范围1 - 10，不设置默认返回一张最佳截图	Integer	否

返回结果说明

response 为响应头，data 为活体人脸核身结果，数据类型都为 dict。

response 内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体 data

响应体具体数据内容如下：

参数名称	参数描述	类型
BestFrameBase64	验证通过后的视频最佳截图照片，照片为 BASE64 编码后的值，JPG 格式 注意：此字段可能返回 null，表示取不到有效值	String
Sim	相似度，取值范围 [0.00, 100.00]。推荐相似度大于等于70时可判断为同一人，可根据具体场景自行调整阈值（阈值70的误通过率为千分之一，阈值80的误通过率是万分之一） 检测成功返回	float
BestFrameList	最佳截图列表，仅在配置了返回多张最佳截图时返回 注意：此字段可能返回 null，表示取不到有效值	String Array

人体识别

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于人体识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
人体识别	人体识别

人体识别

功能说明

腾讯云数据万象通过 AIBodyRecognition 接口识别并输出画面中人体，输出其位置（矩形框）和置信度。图片人体识别请求包属于 GET 请求，请求时需要携带签名。

示例代码

```
def cos_ai_body_recognition():
    # 人体识别
    # demo地址：https://github.com/tencentyun/cos-python-sdk-v5/blob/master/demo/ai_recognition_demo.py#L99
    response, data = client.cos_ai_body_recognition(
        Bucket=bucket_name,
        ObjectKey="test.jpg",
        DetectUrl="https://test-125000000.cos.ap-chongqing.myqcloud.com/test.jpeg"
    )
    print(response)
    print(data)
    return response, data
```

参数说明

调用 cos_ai_body_recognition 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
Object	设置对象名称。ObjectKey与DetectUrl必选一个	Stri	否

Key		ng	
Detect Url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey，detect-url 示例：`http://www.example.com/abc.jpg`，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg` 。ObjectKey与DetectUrl必选一个	String	否

返回结果说明

response 为响应头，data 为人体识别结果，数据类型都为 dict。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

由于接口默认返回xml内容，解析结果都为string类型，以下描述为字段的真实类型。具体数据内容如下：

参数名称	参数描述	类型
Status	人体识别结果。0表示未识别到，1表示识别到	Integer
Pedestrian Info	人体识别结果，可能有多个	Container Array

PedestrianInfo 节点内容：

参数名称	参数描述	类型
Name	识别类型，人体识别默认：person	String
Score	人体的置信度，取值范围为[0-100]。值越高概率越大。	String
Location	图中识别到人体的坐标	Container Array

Location 节点内容：

参数名称	参数描述	类型
Point	人体坐标点（X坐标,Y坐标）	String Array

人脸识别
人脸检测

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于人脸检测的相关 API 概览以及 SDK 示例代码。

API	操作描述
人脸检测	人脸检测

人脸检测

功能说明

人脸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人脸位置。该接口属于 GET 请求。

示例代码

```
def cos_ai_detect_face():
    # 人脸检测
    # https://github.com/tencentyun/cos-python-sdk-v5/blob/master/demo/ai_recognition_demo.py#L111
    response, data = client.cos_ai_detect_face(
        Bucket=bucket_name,
        ObjectKey = "objectKey",
        MaxFaceNum = 3
    )
    print(response)
    print(data)
    return response, data
```

参数说明

调用 cos_ai_detect_face 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
ObjectKey	设置对象名称	String	是

MaxFaceNum	最多处理的人脸数目。默认值为1（仅检测图片中面积最大的那张人脸），最大值为120。此参数用于控制处理待检测图片中的人脸个数，值越小，处理速度越快。	Integer	否

返回结果说明

response 为响应头，data 为人脸检测结果，数据类型都为 dict。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

由于接口默认返回xml内容，解析结果都为string类型，以下描述为字段的真实类型。具体数据内容如下：

参数名称	参数描述	类型
ImageWidth	请求的图片宽度。	Integer
ImageHeight	请求的图片高度。	Integer
FaceModelVersion	人脸识别所用的算法模型版本。	String
RequestId	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。	String
FaceInfos	人脸信息列表。包含人脸坐标信息。	Container
Status	是否识别到人脸	String

FaceInfos 节点内容：

参数名称	参数描述	类型
X	人脸框左上角横坐标。人脸框包含人脸五官位置并在此基础上进行一定的扩展，若人脸框超出图片范围，会导致坐标负值。	Integer
Y	人脸框左上角纵坐标。人脸框包含人脸五官位置并在此基础上进行一定的扩展，若人脸框超出图片范围，会导致坐标负值。	Integer
Width	人脸框宽度。	Integer
Height	人脸框高度。	Integer

版权所有：腾讯云计算（北京）有限责任公司

人脸特效

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于人脸特效的相关 API 概览以及 SDK 示例代码。

API	操作描述
人脸特效	人脸特效

人脸特效

功能说明

人脸特效，提供人脸美颜、人像变换、人像分割功能。

示例代码

```
def cos_ai_face_effect():
    # 人脸特效
    # demo地址：https://github.com/tencentyun/cos-python-sdk-v5/blob/master/demo/ai_recognition_demo.py#L164
    response, data = client.cos_ai_face_effect(
        Bucket=bucket_name,
        ObjectKey="relian.jpeg",
        # DetectUrl="https://test-125000000.cos.ap-chongqing.myqcloud.com/test.jpeg",
        Type="face-beautify",
        Whitening=30,
        Smoothing=10,
        FaceLifting=70,
        EyeEnlarging=70,
    )
    print(response)
    print(data)
    return response, data
```

参数说明

调用 cos_ai_face_effect 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填

Bucket	存储桶名称	String	是
ObjectKey	设置对象名称。ObjectKey与DetectUrl必选一个	String	否
DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey。detect-url 示例：`http://www.example.com/abc.jpg`，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg` 。ObjectKey与DetectUrl必选一个	String	否
Type	人脸特效类型 • 人脸美颜：face-beautify • 人脸性别转换：face-gender-transformation • 人脸年龄变化：face-age-transformation • 人像分割：face-segmentation	String	是
Whitening	type 为 face-beautify 时生效，美白程度，取值范围[0,100]。0不美白，100代表最高程度。默认值30	Integer	否
Smoothing	type 为 face-beautify 时生效，磨皮程度，取值范围[0,100]。0不磨皮，100代表最高程度。默认值10	Integer	否
FaceLifting	type 为 face-beautify 时生效，瘦脸程度，取值范围[0,100]。0不瘦脸，100代表最高程度。默认值70	Integer	否
EyeEnlarging	type 为 face-beautify 时生效，大眼程度，取值范围[0,100]。0不大眼，100代表最高程度。默认值70	Integer	否
Gender	type 为 face-gender-transformation 时生效，选择转换方向，0：男变女，1：女变男。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。	Integer	否
Age	type为 face-age-transformation 时生效，变化到的人脸年龄，[10,80]。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。	Integer	否

返回结果说明

response 为响应头，data 为人脸特效结果，数据类型都为 dict。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

响应体具体数据内容如下：

参数名称	参数描述	类型
ResultImage	处理后的图片 base64 数据	String
ResultMask	人像分割输出参数，一个通过 Base64 编码的文件，解码后文件由 Float 型浮点数组成。这些浮点数代表原图从左上角开始的每一行的每一个像素点，每一个浮点数的值是原图相应像素点位于人体轮廓内的置信度（0-1）转化的灰度值（0-255）	String

宠物识别

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于宠物识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
宠物识别	宠物识别

宠物识别

功能说明

腾讯云数据万象通过 detect-pet 接口识别并输出画面中宠物，输出其位置（矩形框）和置信度。图片宠物识别请求包属于 GET 请求，请求时需要携带签名。

示例代码

```
def cos_ai_detect_pet():
    # 宠物识别
    # demo 地址: https://github.com/tencentyun/cos-python-sdk-v5/blob/master/demo/ai_recognition_demo.py#L123
    response, data = client.cos_ai_detect_pet(
        Bucket=bucket_name,
        ObjectKey = "objectKey",
    )
    print(response)
    print(data)
    return response, data
```

参数说明

调用 cos_ai_detect_pet 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
Object Key	设置对象名称	String	是

返回结果说明

response 为响应头，data 为宠物识别结果，数据类型都为 dict。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

由于接口默认返回xml内容，解析结果都为string类型，以下描述为字段的真实类型。具体数据内容如下：

参数名称	参数描述	类型
ResultInfo	保存结果的容器	Container Array

ResultInfo 节点内容：

参数名称	参数描述	类型
Score	分数，范围在0-100之间，值越高，表示目标为相应结果的可能性越高	Integer
Name	宠物名称	String
Location	保存宠物位置的容器	Container

Location 节点内容：

参数名称	参数描述	类型
X	x轴坐标	Integer
Y	y轴坐标	Integer
Height	(x,y)坐标距离高度	Integer
Width	(x,y)坐标距离长度	Integer

游戏场景识别

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于游戏场景识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
游戏场景识别	游戏场景识别

游戏场景识别

功能说明

游戏标签功能实现游戏图片场景的识别，返回图片中置信度较高的游戏类别标签。游戏标签识别请求包属于 GET 请求，请求时需要携带签名。

示例代码

```
def cos_ai_game_rec():
    # 游戏场景识别
    # demo地址: https://github.com/tencentyun/cos-python-sdk-v5/blob/master/demo/ai_recognition_demo.py#L181
    response, data = client.cos_ai_game_rec(
        Bucket=bucket_name,
        ObjectKey="youxi.png",
        # DetectUrl="https://test-125000000.cos.ap-chongqing.myqcloud.com/test.jpeg"
    )
    print(response)
    print(data)
    return response, data
```

参数说明

调用 cos_ai_game_rec 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
ObjectKey	图片地址。ObjectKey与DetectUrl必选一个	String	否

DetectUrl	您可以通过填写 detect-url 对任意公网可访问的图片进行游戏场景识别。不填写 detect-url 时，后台会默认处理 objectkey；填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 objectkey， detect-url 示例：`http://www.example.com/abc.jpg`，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`。ObjectKey与DetectUrl必选一个	String	否

返回结果说明

response 为响应头，data 为游戏场景识别结果，数据类型都为 dict。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

由于接口默认返回xml内容，解析结果都为string类型，以下描述为字段的真实类型。具体数据内容如下：

参数名称	参数描述	类型
GameLabels	游戏标签信息	Container

GameLabels 节点内容：

参数名称	参数描述	类型
Confidence	该标签的置信度分数，分数越高则该标签准确度越高	Integer
FirstCategory	一级标签	String
SecondCategory	二级标签	String
GameName	识别出的游戏标签名称	String

卡证识别

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于卡证识别的相关 API 概览以及 SDK 示例代码。

API	操作描述
卡证识别	卡证识别

卡证识别

功能说明

本接口支持中国大陆居民二代身份证正面（暂不支持背面）、驾驶证主页（暂不支持副页）所有字段的自动定位，暂不支持文本识别，用于对特定字段的抹除、屏蔽，以及进一步的文本识别。

示例代码

```
def cos_ai_license_rec():
    # 卡证识别

    response, data = client.cos_ai_license_rec(
        Bucket=bucket_name,
        ObjectKey="shenfenzheng.jpeg",
        # DetectUrl="https://test-125000000.cos.ap-
chongqing.myqcloud.com/test.jpeg"
        CardType="IDCard"
    )
    print(response)
    print(data)
    return response, data
```

参数说明

调用 cos_ai_license_rec 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
ObjectKey	设置对象名称	String	是

DetectUrl	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey，detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`	String	否
CardType	卡证识别类型，有效值为 IDCard，DriverLicense。 • IDCard 表示身份证 • DriverLicense 表示驾驶证 • 默认：DriverLicense	String	是

返回结果说明

response 为响应头，data 为卡证识别结果，数据类型都为 dict。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

由于接口默认返回xml内容，解析结果都为string类型，以下描述为字段的真实类型。具体数据内容如下：

参数名称	参数描述	类型
Status	卡证识别结果 • 1：识别到结果 • 0：未识别到结果	Integer
IdInfo	卡证识别信息。	Container Array

IdInfo 节点内容：

参数名称	参数描述	类型
Name	字段的名称	String
DetectedText	字段的值	String
Score	字段的置信度，取值范围为[0-100]，值越高概率越大	Integer
Location	驾驶证中识别到字段的坐标	Container Array

Location 节点内容：

参数名称	参数描述	类型
Point	驾驶证信息 坐标点（X 坐标，Y 坐标）	String

版权所有：腾讯云计算（北京）有限责任公司

实时文字翻译

最近更新时间：2025-06-17 20:30:52

简介

本文档提供关于实时文字翻译的相关 API 概览以及 SDK 示例代码。

API	操作描述
实时文字翻译	实时文字翻译

功能说明

腾讯云数据万象通过 AutoTranslationBlock 接口对文字块进行翻译，请求时需要携带签名。

示例代码

```
from qcloud_cos import AIRecognitionClient
from qcloud_cos import CosConfig

# 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.cloud.tencent.com/cam/capi
secret_id = os.environ["SECRETID"]
# 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.cloud.tencent.com/cam/capi
secret_key = os.environ["SECRETKEY"]
# 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
region = 'ap-chongqing'
# COS 支持的所有 region 列表参见 https://www.qcloud.com/document/product/436/6224
# 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://cloud.tencent.com/document/product/436/14048
token = None
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key,
Token=token)
client = AIRecognitionClient(config)
bucket_name = 'test-1250000000'

def cos_auto_translation_block():
    # 实时文字翻译

    response, data = client.cos_auto_translation_block(
        Bucket=bucket_name,
        InputText="测试",
        SourceLang="zh",
```

```
        TargetLang="en",
        TextDomain="general",
        TextStyle="sentence"
    )
    print(response)
    print(data)
    return response, data
```

参数说明

调用 cos_auto_translation_block 函数，具体请求参数如下：

参数名称	参数描述	类型	是否必填
Bucket	存储桶名称	String	是
CiProcess	数据万象处理能力，文本块翻译固定为 AutoTranslationBlock。	String	是
InputText	待翻译的文本	String	是
SourceLang	输入语言，如 "zh"	String	是
TargetLang	输出语言，如 "en"	String	是
TextDomain	文本所属业务领域，如: "ecommerce", 缺省值为 general	String	否
TextStyle	文本类型，如: "title", 缺省值为 sentence	String	否

返回结果说明

response 为响应头，data 为实时文字翻译结果，数据类型都为 dict。

response内容

返回公共响应头部，详情请参见 公共响应头部 文档。

响应体data

响应体具体数据内容如下：

参数名称	参数描述	类型

TranslationResult	翻译结果	String

小程序 SDK
快速入门

最近更新时间：2025-03-14 19:19:32

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

- 该 SDK 只适用于微信小程序环境。
- 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 。
- 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。

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：

- cos.postObject 使用 wx.uploadFile 方法。
- 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：

- 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` 。
- 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。

```
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://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.md
        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.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

版权所有：腾讯云计算（北京）有限责任公司

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
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
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（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);
});
```

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。请参见 授权指引 。
常见上传错误排查，请参见 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);
});
```

内容识别
服务和队列配置
开通 AI 内容识别服务

最近更新时间：2024-12-27 16:37:08

简介

开通AI内容识别服务功能是由 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。
本文档提供关于开通AI内容识别服务的相关 API 概览以及 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
开通AI内容识别服务	本接口用于开通AI 内容识别（异步）服务并生成队列。

开通AI内容识别服务

功能说明

本接口用于开通AI 内容识别（异步）服务并生成队列。

使用示例

```
function openAIBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'ai_bucket'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
```

```
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
openAIBucket();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Url	请求url，固定值	String	是
Key	固定值：ai_bucket	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 开通AI内容识别服务	Object

相关参考

API 文档

查询 AI 内容识别服务

最近更新时间：2024-12-27 16:37:08

简介

查询AI内容识别服务功能是由 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。
本文档提供关于查询AI内容识别服务的相关 API 概览以及 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
查询AI内容识别服务	本接口用于查询已经开通AI 内容识别（异步）服务的存储桶。

查询AI内容识别服务

功能说明

本接口用于查询已经开通AI 内容识别（异步）服务的存储桶。

使用示例

```
function getAIBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'ai_bucket'; // 固定值
  const host = `ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;

  cos.request(
    {
      Method: 'GET', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Query: {
        // 地域信息，例如 ap-shanghai、ap-beijing，若查询多个地域以“,”分隔字符串，详情
请参见 地域与域名;是否必传：否
        // regions:  '',
```

```
        // 存储桶名称，以“,”分隔，支持多个存储桶，精确搜索;是否必传：否
        // bucketNames:  '',
        // 存储桶名称前缀，前缀搜索;是否必传：否
        // bucketName:  '',
        // 第几页;是否必传：否
        // pageNumber: 1,
        // 每页个数，大于0且小于等于100的整数;是否必传：否
        // pageSize: 10,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
getAIBucket();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：GET	String	是
Url	请求url，固定值	String	是
Key	固定值：ai_bucket	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
regions	地域信息，例如 ap-shanghai、ap-beijing，若查询多个地域以“,”分隔字符串，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	Integer	否
pageSize	每页个数，大于0且小于等于100的整数	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果 详情请参见 查询AI内容识别服务	Object

相关链接

API 文档

关闭 AI 内容识别服务

最近更新时间：2024-12-27 16:37:08

简介

关闭AI内容识别服务功能是由 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。
本文档提供关于关闭AI内容识别服务的相关 API 概览以及 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
关闭AI内容识别服务	本接口用于关闭AI 内容识别（异步）服务并删除队列。

关闭AI内容识别服务

功能说明

本接口用于关闭AI 内容识别（异步）服务并删除队列。

使用示例

```
function closeAIBucket() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'ai_bucket'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;

  cos.request(
    {
      Method: 'DELETE', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
```

```
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
closeAIBucket();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：DELETE	String	是
Url	请求url，固定值	String	是
Key	固定值：ai_bucket	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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

- Response	响应结果 详情请参见 关闭AI内容识别服务	Object

相关参考

API 文档

查询 AI 内容识别队列

最近更新时间：2024-12-27 16:37:08

简介

查询AI内容识别队列功能是由 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。
本文档提供关于查询AI内容识别队列的相关 API 概览以及 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
查询AI内容识别队列	本接口用于查询AI 内容识别（异步）队列。

查询AI内容识别队列

功能说明

本接口用于查询AI 内容识别（异步）队列。

使用示例

```
function getAIQueue() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'ai_queue'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;

  cos.request(
    {
      Method: 'GET', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Query: {
        // 队列 ID，以“,”符号分割字符串;是否必传：否
        // queueIds:  "",
```

```
        // Active 表示队列内的作业会被调度执行Paused 表示队列暂停，作业不再会被调度执行，
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;是否必传：否
        state: 'Active',
        // 第几页，默认值1;是否必传：否
        // pageNumber: 0,
        // 每页个数，默认值10;是否必传：否
        // pageSize: 0,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
getAIQueue();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：GET	String	是
Url	请求url，固定值	String	是
Key	固定值：ai_queue	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
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调度执行Paused 表示队列暂停，作业不再会被调度执行，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
pageNumber	第几页，默认值1。	Integer	否

pageSize	每页个数，默认值10。	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果 详情请参见 查询AI内容识别队列	Object

相关参考

API 文档

更新 AI 内容识别队列

最近更新时间：2024-12-27 16:37:08

简介

更新AI内容识别队列功能是由 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。
本文档提供关于更新AI内容识别队列的相关 API 概览以及 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
更新AI内容识别队列	本接口用于更新 AI 内容识别（异步）的队列。

更新AI内容识别队列

功能说明

本接口用于更新 AI 内容识别（异步）的队列。

使用示例

```
function updateAIQueue() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const queueId = 'xxxx'; // 队列id
  const key = `ai_queue/${queueId}`; // 固定格式
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 队列名称，仅支持中文、英文、数字、_、-和*，长度不超过 128;是否必传：是
      Name: 'my_queue',
      // Active 表示队列内的作业会被调度执行Paused 表示队列暂停，作业不再会被调度执行，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;是否必传：是
      State: 'Active',
      // 回调配置;是否必传：是
      NotifyConfig: {
```

```
          // 回调开关OffOn;是否必传：否
          State: 'On',
          // 回调事件TaskFinish：任务完成WorkflowFinish：工作流完成;是否必传：否
          Event: 'TaskFinish',
          // 回调格式XMLJSON;是否必传：否
          // ResultFormat: '',
          // 回调类型UrlTDMQ;是否必传：否
          Type: 'Url',
          // 回调地址，不能为内网地址。;是否必传：否
          Url: 'http://example.com',
          // TDMQ 使用模式Topic：主题订阅Queue：队列服务;是否必传：否
          // MqMode: '',
          // TDMQ 所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、
hk（中国香港）;是否必传：否
          // MqRegion: '',
          // TDMQ 主题名称;是否必传：否
          // MqName: '',
        },
      },
    });

    cos.request(
      {
        Method: 'PUT', // 固定值，必须
        Key: key, // 必须
        Url: url, // 请求的url，必须
        Body: body, // 请求体参数，必须
        ContentType: 'application/xml', // 固定值，必须
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
updateAIQueue();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：PUT	String	是
Url	请求 url，固定值	String	是
Key	ai_queue/{queueId}	String	是
ContentType	固定值	String	是
Body	请求体详情请参见 更新AI内容识别队列	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 更新AI内容识别队列	Object

相关参考

API 文档

图片上色

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图片上色的 API 概览以及 SDK 示例代码。

> **注意：**
> - COS 小程序 SDK 版本需要使用 v1.1.1及以上版本。
> - 待处理的图片，原图大小不超过5MB。

API	说明
图片上色	腾讯云数据万象通过 AIImageColoring 接口对黑白图像进行上色

上传时处理

下面示例展示了如何在上传图片时自动实现图片上色。
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

```
<view>
  <button type="primary" bindtap="button">上传</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) {
            // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
            cos.putObject(
              {
                Bucket: 'examplebucket-1250000000', // Bucket 格式：test-
1250000000，必填
                Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
```

```
                Key: file.name, // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），
必填
                Body: res.data, // 上传文件对象，必填
                Headers: {
                  // 通过 AIImageColoring 接口使用图片上色功能
                  'Pic-Operations': JSON.stringify({
                    is_pic_info: 1,
                    rules: [{ fileid: 'test.jpg', rule: 'ci-
process=AIImageColoring' }]
                  }),
                },
              },
              (err, data) => {
                if (err) {
                  // 处理请求失败
                  console.log(err);
                } else {
                  // 处理请求成功
                  console.log(data);
                }
              },
            );
          },
          fail: (err) => console.error(err),
        });
      },
      fail: (err) => console.error(err),
    });
  },
});
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容，一般传入选择的文件 file 对象	File/Blob	是

Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AIImageColoring	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 图片上色	Object

对云上数据进行图片上色

下面示例展示了如何在对已存储在 COS 的图片进行图片上色操作，并将结果存入到 COS。

示例代码

```
<view>
  <button type="primary" bindtap="button">对云上数据处理</button>
</view>
```

```
Page({
  button: function () {
```

```
    // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
    cos.request(
      {
        Bucket: 'examplebucket-1250000000', // Bucket 格式：test-1250000000，必
填
        Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
        Key: '1.png', // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必填
        Method: 'POST', // 固定值
        Action: 'image_process', // 固定值
        Headers: {
            // 通过 AIImageColoring 接口使用图片上色功能
            'Pic-Operations': JSON.stringify({
              is_pic_info: 1,
              rules: [{ fileid: 'test.jpg', rule: 'ci-process=AIImageColoring'
}]
            }),
        },
      },
      (err, data) => {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      },
    );
  },
});
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是

Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AIImageColoring	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图片上色	Object

下载时使用图片上色

下面示例展示了如何在下载图片时实现图片上色。

示例代码

```
<<view>
  <button type="primary" bindtap="button">下载时处理</button>
</view>
```

```
Page({
  button: function () {
    // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
    cos.request(
      {
```

```
        Bucket: 'examplebucket-1250000000', // Bucket 格式：test-1250000000，必
填
        Region: 'COS_REGION', // Bucket 所在地域，比如ap-beijing，必填
        Method: 'GET',  // 固定值
        Key: '1.png', // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），与 detect-
url二选一传递
        Query: {
          'ci-process': 'AIImageColoring',
          // 'detect-url': '', // 待处理图片 url，与 Key 二选一传递
        }
      },
      (err, data) => {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      },
    );
  },
});
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	否
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必选

ci-process	数据万象处理能力，图片上色固定为 AIImageColoring	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接 detect-url 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图片上色	Object

相关链接

- API 文档
- 功能指南

图像超分

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图像超分的 API 概览以及 SDK 示例代码。

> **注意：**
> - COS 小程序 SDK 版本需要使用 v1.1.1及以上版本。
> - 待处理的图片，原图大小不超过5MB，分辨率不得超过1920px * 1080px。

API	说明
图像超分	腾讯云数据万象通过 AISuperResolution 接口对图像进行超分辨率处理，当前默认超分为宽高的2倍

上传时处理

下面示例展示了如何在上传图片时自动实现图像超分。
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

```
<view>
  <button type="primary" bindtap="button">上传</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) {
            // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
            cos.putObject(
              {
                Bucket: 'examplebucket-1250000000', // Bucket 格式：test-
1250000000，必填
                Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
```

```
                Key: file.name, // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），
必填
                Body: res.data, // 上传文件对象，必填
                Headers: {
                    // 通过 AISuperResolution 接口使用图像超分功能
                    'Pic-Operations': JSON.stringify({
                      is_pic_info: 1,
                      rules: [{ fileid: 'test.jpg', rule: 'ci-
process=AISuperResolution' }]
                    }),
                  },
              },
              (err, data) => {
                if (err) {
                  // 处理请求失败
                  console.log(err);
                } else {
                  // 处理请求成功
                  console.log(data);
                }
              },
            );
          },
          fail: (err) => console.error(err),
        });
      },
      fail: (err) => console.error(err),
    });
  },
});
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容，一般传入选择的文件 file 对象	File/Blob	是

Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AISuperResolution	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图片超分	Object

对云上数据进行图像超分

下面示例展示了如何对已存储在 COS 的图片进行图像超分操作，并将结果存入到 COS。

示例代码

```
<view>
  <button type="primary" bindtap="button">对云上数据处理</button>
</view>
```

```
Page({
  button: function () {
    // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
    cos.request(
      {
```

```
        Bucket: 'examplebucket-1250000000', // Bucket 格式：test-1250000000，必
填
        Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
        Key: '1.png', // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必填
        Method: 'POST', // 固定值
        Action: 'image_process', // 固定值
        Headers: {
          // 通过 AISuperResolution 接口使用图像超分功能
          'Pic-Operations': JSON.stringify({
            is_pic_info: 1,
            rules: [{ fileid: 'test.jpg', rule: 'ci-process=AISuperResolution'
}]
          }),
        },
      },
      (err, data) => {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      },
    );
  },
});
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AISuperResolution	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 图片超分	Object

下载时使用图像超分

下面示例展示了如何下载图片时实现图像超分。

示例代码

```
<<view>
  <button type="primary" bindtap="button">下载时处理</button>
</view>
```

```
Page({
  button: function () {
    // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
    cos.request(
      {
```

```
        Bucket: 'examplebucket-1250000000', // Bucket 格式：test-1250000000，必
填
        Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
        Method: 'GET',  // 固定值
        Key: '1.png', // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），与 detect-
url二选一传递
        Query: {
          'ci-process': 'AISuperResolution',  // 固定值
          // 'detect-url': '', // 待处理图片 url，与 Key 二选一传递
        },
      },
      (err, data) => {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      },
    );
  },
});
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	否
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必选

ci-process	数据万象处理能力，图像超分固定为 AISuperResolution	String	是
detect-url	处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接 detect-url 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 图像超分	Object

相关链接

- API 文档
- 功能指南

图像增强

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图像增强的 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> - COS 小程序 SDK 版本需要使用 v1.1.1及以上版本。
> - 待处理的图片，原图大小不超过5MB。

API	说明
图像增强	腾讯云数据万象通过 AIEnhanceImage 接口对图像进行增强处理

上传时处理

下面示例展示了如何在上传图片时自动实现图像增强。
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

```
<view>
  <button type="primary" bindtap="button">上传</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) {
            // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
            cos.putObject(
              {
                Bucket: 'examplebucket-1250000000', // Bucket 格式：test-
1250000000，必填
                Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
```

```
                Key: file.name, // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），
必填
                Body: res.data, // 上传文件对象，必填
                Headers: {
                    // 通过 AIEnhanceImage 接口使用图像增强功能：指定图片锐化强度值5，
去噪强度值3
                    'Pic-Operations': JSON.stringify({
                      is_pic_info: 1,
                      rules: [{ fileid: 'test.jpg', rule: 'ci-
process=AIEnhanceImage' }]
                    }),
                  },
              },
              (err, data) => {
                if (err) {
                  // 处理请求失败
                  console.log(err);
                } else {
                  // 处理请求成功
                  console.log(data);
                }
              },
            );
          },
          fail: (err) => console.error(err),
        });
      },
      fail: (err) => console.error(err),
    });
  },
});
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容，一般传入选择的文件 file 对象	File/Blob	是

版权所有：腾讯云计算（北京）有限责任公司

Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AIEnhanceImage	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图像增强	Object

对云上数据进行图像增强

下面示例展示了如何在对已存储在 COS 的图片进行图像增强操作，并将结果存入到 COS。

示例代码

```
<view>
  <button type="primary" bindtap="button">对云上数据处理</button>
</view>
```

```
Page({
  button: function () {
    // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
    cos.request(
      {
```

```
        Bucket: 'examplebucket-1250000000', // Bucket 格式：test-1250000000，必
填
        Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
        Key: '1.png', // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必填
        Method: 'POST', // 固定值
        Action: 'image_process', // 固定值
        Headers: {
          // 通过 AIEnhanceImage 接口使用图像增强功能：指定图片锐化强度值5，去噪强度值3
          'Pic-Operations': JSON.stringify({
            is_pic_info: 1,
            rules: [{ fileid: 'test.jpg', rule: 'ci-process=AIEnhanceImage' }]
          }),
        },
      },
      (err, data) => {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      },
    );
  },
});
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是
Pic-Operations	只需填写 fileid，代表处理后的图片路径，如'test.jpg'。rule 固定为 ci-process=AIEnhanceImage	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果，详情请参见 图像增强	Object

下载时使用图像增强

下面示例展示了如何在下载图片时实现图像增强。

示例代码

```
<<view>
  <button type="primary" bindtap="button">下载时处理</button>
</view>
```

```
Page({
  button: function () {
    // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
    cos.request(
      {
        Bucket: 'examplebucket-1250000000', // Bucket 格式：test-1250000000，必
填
        Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
```

```
        Method: 'GET',  // 固定值
        Key: '1.png', // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），与 detect-url二选一传递
        Query: {
          'ci-process': 'AIEnhanceImage',  // 固定值
          denoise: 5, // 非必需，去噪强度值，取值范围为0 - 5之间的整数，值为0时不进行去噪操作，默认值为3
          sharpen: 5, // 非必需，锐化强度值，取值范围为0 - 5之间的整数，值为0时不进行锐化操作，默认值为3
          'detect-url': 'http://www.example.com/abc.jpg'  // 非必须，可以通过填写 detect-url处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，detect-url 需要进行 UrlEncode
        },
      },
      (err, data) => {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      },
    );
  },
});
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	否
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必选
ci-process	数据万象处理能力，图像增强固定为 AIEnhanceImage	String	是
denoise	去噪强度值，取值范围为0 - 5之间的整数，值为0时不进行去噪操作，默认值为3	Integer	否
sharpen	锐化强度值，取值范围为0 - 5之间的整数，值为0时不进行锐化操作，默认值为3	Integer	否
detect-url	处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接 detect-url 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图像增强	Object

相关链接

- API 文档

- 功能指南

图像智能裁剪

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图像智能裁剪的 API 概览以及 SDK 示例代码。

> ⚠ **注意：**
> - COS 小程序 SDK 版本需要使用 v1.1.1及以上版本。
> - 待处理的图片，原图大小不超过5MB。

API	说明
图像智能裁剪	腾讯云数据万象通过 AIImageCrop 接口对图像进行智能裁剪

上传时处理

下面示例展示了如何在上传图片时自动实现图像智能裁剪。
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

```
<view>
  <button type="primary" bindtap="button">上传</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) {
            // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
            cos.putObject(
              {
                Bucket: 'examplebucket-1250000000', // Bucket 格式: test-
1250000000，必填
```

```
                Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
                Key: file.name, // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），
必填
                Body: res.data, // 上传文件对象，必填
                Headers: {
                    // 通过 AIImageCrop 接口使用图像智能裁剪功能：指定图片剪区域的宽度
1，高度2
                    'Pic-Operations': JSON.stringify({
                      is_pic_info: 1,
                      rules: [{ fileid: 'test.jpg', rule: 'ci-
process=AIImageCrop&width=1&height=2' }]
                    }),
                  },
                },
                (err, data) => {
                  if (err) {
                    // 处理请求失败
                    console.log(err);
                  } else {
                    // 处理请求成功
                    console.log(data);
                  }
                },
              );
            },
            fail: (err) => console.error(err),
          });
        },
        fail: (err) => console.error(err),
      });
    },
});
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Body	文件内容，一般传入选择的文件 file 对象	File/Blob	是

Pic-Operations	详情请参见 图像智能裁剪处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 图像智能裁剪	Object

对云上数据进行图像智能裁剪

下面示例展示了如何对已存储在 COS 的图片进行图像智能裁剪操作，并将结果存入到 COS。

示例代码

```
<view>
  <button type="primary" bindtap="button">对云上数据处理</button>
</view>
```

```
Page({
  button: function () {
```

```
    // sdk 的引入及初始化 cos 请参考 https://cloud.tencent.com/document/product/436/31953
    cos.request(
      {
        Bucket: 'examplebucket-1250000000', // Bucket 格式：test-1250000000，必填
        Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
        Key: '1.png', // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必填
        Method: 'POST', // 固定值
        Action: 'image_process', // 固定值
        Headers: {
          // 通过 AIImageCrop 接口使用图像智能裁剪功能：指定图片剪区域的宽度1，高度2
          'Pic-Operations': JSON.stringify({
            is_pic_info: 1,
            rules: [{ fileid: 'test.jpg', rule: 'ci-process=AIImageCrop&width=1&height=2' }]
          }),
        },
      },
      (err, data) => {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      },
    );
  },
});
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是
Method	固定值：POST	String	是
Action	固定值：image_process	String	是

Pic-Operations	详情请参见 图像智能裁剪处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 图像智能裁剪	Object

下载时使用图像智能裁剪

下面示例展示了如何在下载图片时实现图像智能裁剪。

示例代码

```
<<view>
  <button type="primary" bindtap="button">下载时处理</button>
</view>
```

```
Page({
  button: function () {
```

```
    // sdk 的引入及初始化 cos 请参考 https://cloud.tencent.com/document/product/436/31953
    cos.request(
      {
        Bucket: 'examplebucket-1250000000', // Bucket 格式：test-1250000000，必填
        Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
        Method: 'GET',  // 固定值
        Key: '1.png', // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），与 detect-url二选一传递
        Query: {
          'ci-process': 'AIImageCrop',  // 固定值
          width: 5, // 必需，需要裁剪区域的宽度
          height: 5, // 必需，需要裁剪区域的高度
          //fixed:1, // 非必需，是否严格按照 width 和 height 的值进行输出
          //detect-url: ''  // 非必须，可以通过填写 detect-url处理任意公网可访问的图片链接。不填写 detect-url时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，detect-url 需要进行 UrlEncode */
        },
      },
      (err, data) => {
        console.log(err || data);
      },
    );
  },
});
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	否
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必选
ci-process	数据万象处理能力，智能裁剪固定为 AIImageCrop	String	是
width	需要裁剪区域的宽度，与 height 共同组成所需裁剪的图片宽高比例；输入数字请大于0、小于图片宽度的像素值	Integer	是
height	需要裁剪区域的高度，与 width 共同组成所需裁剪的图片宽高比例；输入数字请大于0、小于图片高度的像素值；width : height 建议取值在[1, 2.5]之间，超过这个范围可能会影响效果	Integer	是
fixed	是否严格按照 width 和 height 的值进行输出。 • 取值为0时，宽高比例（width : height）会简化为最简分数，即如果 width 输入10、height 输入20，会简化为1：2 • 取值为1时，输出图片的宽度等于 width，高度等于 height 默认值为0	Integer	否
detect-url	处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接 detect-url 示例：`http://www.example.com/abc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-header	请求返回的头部信息	Object

s		
– Response	响应结果，详情请参见 图像智能裁剪	Object

相关链接

- API 文档
- 功能指南

图像修复

最近更新时间：2025-02-11 15:37:23

简介

本文档提供关于数据万象图像修复的 API 概览以及 SDK 示例代码。

> **注意：**
> - COS 小程序 SDK 版本需要使用 v1.1.1及以上版本。
> - 待处理的图片，原图大小不超过32MB。
> - 宽高不超过50000像素且总像素不超过2.5亿像素，处理结果图宽高设置不超过9999像素。

API	说明
图像修复	腾讯云数据万象通过 ImageRepair 接口检测图片中的水印并将其擦除

上传时处理

下面示例展示了如何在上传图片时自动实现图像修复。
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

```
<view>
  <button type="primary" bindtap="button">上传</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) {
            // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
            cos.putObject(
              {
```

```
                Bucket: 'examplebucket-1250000000', // Bucket 格式：test-
1250000000，必填
                Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
                Key: file.name, // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），
必填
                Body: res.data, // 上传文件对象，必填
                Headers: {
                  // 通过 ImageRepair 接口使用图像修复功能：指定遮罩图片地址 URL 安全
的 Base64编码：aHR0c...XXX
                  'Pic-Operations': JSON.stringify({
                    is_pic_info: 1,
                    rules: [{ fileid: 'test.jpg', rule: 'ci-
process=ImageRepair&MaskPic=aHR0c...XXX' }]
                  }),
                },
              },
              (err, data) => {
                if (err) {
                  // 处理请求失败
                  console.log(err);
                } else {
                  // 处理请求成功
                  console.log(data);
                }
              },
            );
          },
          fail: (err) => console.error(err),
        });
      },
      fail: (err) => console.error(err),
    });
  },
});
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是

Body	文件内容，一般传入选择的文件 file 对象	File/Blob	是
Pic-Operations	详情请参见 图像修复处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 图像修复	Object

对云上数据进行图像修复

下面示例展示了如何对已存储在 COS 的图片进行图像修复操作，并将结果存入到 COS。

示例代码

```
<view>
  <button type="primary" bindtap="button">对云上数据处理</button>
</view>
```

```
Page({
  button: function () {
    // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
    cos.request(
      {
        Bucket: 'examplebucket-1250000000', // Bucket 格式：test-1250000000，必
填
        Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
        Key: '1.png', // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必填
        Method: 'POST', // 固定值
        Action: 'image_process', // 固定值
        Headers: {
          // 通过 ImageRepair 接口使用图像修复功能：指定遮罩图片地址 URL 安全的 Base64
编码：aHR0c...XXX
          'Pic-Operations': JSON.stringify({
            is_pic_info: 1,
            rules: [{ fileid: 'test.jpg', rule: 'ci-
process=ImageRepair&MaskPic=aHR0c...XXX' }]
          }),
        },
      },
      (err, data) => {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      },
    );
  },
});
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg	String	是

Method	固定值：POST	String	是
Action	固定值：image_process	String	是
Pic-Operations	详情请参见 图像修复处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 图像修复	Object

下载时使用图像修复

下面示例展示了如何在下载图片时实现图像修复。

示例代码

```
<<view>
  <button type="primary" bindtap="button">下载时处理</button>
</view>
```

```
Page({
  button: function () {
    // sdk 的引入及初始化 cos 请参考
https://cloud.tencent.com/document/product/436/31953
    cos.request(
      {
        Bucket: 'examplebucket-1250000000', // Bucket 格式：test-1250000000，必
填
        Region: 'COS_REGION', // Bucket 所在地域，比如 ap-beijing，必填
        Method: 'GET',  // 固定值
        Key: '1.png', // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），必填
        Query: {
          'ci-process': 'ImageRepair',  // 固定值
          MaskPic: 'aHR0c...XXX' /*非必需，遮罩（白色区域为需要去除的水印位置）图片地
址，需要经过 URL 安全的 Base64编码*/,
          //MaskPoly:'[[[608, 794], [1024, 794]],[[1295, 62],[1295, 30]]]' /*
非必需，顺时针输入多边形的每个点的坐标,MaskPoly 同时与 MaskPic 填写时，优先采用 MaskPic
的值*/,
        },
      },
      (err, data) => {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      },
    );
  },
});
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是

Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必选
ci-process	数据万象处理能力，图像修复固定为 ImageRepair	String	是
MaskPic	遮罩（白色区域为需要去除的水印位置）图片地址，需要经过 URL 安全的 Base64编码。例如，遮罩图片为 `http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/shuiyin_2.png`，则该处编码后的字符串为 `aHR0cDovL2V4YW1wbGVidWNrZXQtMTI1MDAwMDAwMC5jb3MuYXAtc2hhbmdoYWkubXlxY2xvdWQuY29tL3NodWl5aW5fMi5wbmc`	String	否
MaskPoly	例如：[[[608, 794], [1024, 794], [1024, 842], [608, 842]],[[1295, 62], [1295, 30], [1597, 32],[1597, 64]]]，顺时针输入多边形的每个点的坐标,每个多边形：[[x1, y1], [x2, y2]...]，形式为三维矩阵（多个多边形：[多边形1, 多边形2]）或二维矩阵（单个多边形），且需要经过 URL 安全的 Base64 编码。MaskPoly 同时与 MaskPic 填写时，优先采用 MaskPic 的值	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果，详情请参见 图像修复	Object

相关链接

- API 文档
- 功能指南

图片二维码

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于图片二维码相关的 API 概览以及 SDK 示例代码。

API	说明
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。
二维码生成	二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码。

二维码识别

功能说明

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。

请求示例1：上传时识别

```
<view>
  <button type="primary" bindtap="button">上传</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) {
            // sdk的引入及初始化cos请参考
https://cloud.tencent.com/document/product/436/31953
            cos.putObject(
              {
                Bucket: 'examplebucket-1250000000', // Bucket 格式：test-
1250000000，必填
                Region: 'COS_REGION', // Bucket所在地域，比如ap-beijing，必填
```

```
                Key: file.name, // 存储在桶里的对象键（例如1.jpg，a/b/test.txt），
必填
                Body: res.data, // 上传文件对象，必填
                Headers: {
                  'Pic-Operations': JSON.stringify({
                    is_pic_info: 1,
                    rules: [{ fileid: 'test.jpg', rule: 'QRcode/cover/0' }]
                  }),
                },
              },
              (err, data) => {
                if (err) {
                  // 处理请求失败
                  console.log(err);
                } else {
                  // 处理请求成功
                  console.log(data);
                }
              },
            );
          },
          fail: (err) => console.error(err),
        });
      },
      fail: (err) => console.error(err),
    });
  },
});
```

参数说明

参数名称	描述	类型	是否必选
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	对象文件名，例如：folder/document.jpg。	String	是
Body	文件内容，一般传入选择的文件file对象	File/Blob	是
Pic-Operations	详情请参见 图像修复处理参数	JsonString	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果 详情请参见 二维码识别	Object

请求示例2：下载时识别

```
function identifyQrcode_get() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  cos.request({
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Key: '1/jpg', // 对象文件名，例如：folder/document.jpg。
      Query: {
        'ci-process': 'QRcode'  // 固定值
      },
  },
  function(err, data){
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
```

```
        }
    });
}
identifyQrcode_get();
```

参数说明

cos.request 方法参数说明：

参数名称	描述	类型	是否必选
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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg。	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	描述	类型	是否必填
ci-process	万象处理能力，二维码识别固定为 QRcode	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果 详情请参见 二维码识别	Object

二维码生成

功能说明

二维码生成功能可根据用户指定的文本信息（URL 或文本），生成对应的二维码或条形码。

示例代码

```
function generateQrcode() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/11459
  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 */
  };
  cos.request({
      // 需要替换成您自己的存储桶信息
      Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
      Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
      Method: 'GET',  // 固定值
      Query: {
        'ci-process': 'qrcode-generate', /* 必须，对象存储处理能力，二维码生成参数为qrcode-generate */
        'qrcode-content': '二维码文案', /* 必须，可识别的二维码文本信息     */
        // mode: 0, /* 非必须，生成的二维码类型，可选值：0或1。0为二维码，1为条形码，默认值为0  */
        width: 200, /* 必须，指定生成的二维码或条形码的宽度，高度会进行等比压缩   */
      },
  },
  function(err, data){
    if (!err) {
      // 获得二维码 base64
      var imgBase64 = data.Response.ResultImage;
      // 比如可拼接前缀直接展示在 img 里
      // document.querySelector('#img').src = '' + imgBase64;
    }
  });
```

```
}
generateQrcode();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，二维码生成参数为 qrcode-generate	String	是
qrcode-content	可识别的二维码文本信息	String	是
mode	生成的二维码类型，可选值：0或1。0为二维码，1为条形码，默认值为0	Integer	否
width	指定生成的二维码或条形码的宽度，高度会进行等比压缩	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-statusC	请求返回的 HTTP 状态码，例如200、403、404等	Number

ode		
– 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
– Response	响应结果 详情请参见 图片二维码生成	Object

相关链接

API 文档

图片标签

最近更新时间：2025-06-09 19:23:32

简介

本文档提供关于图片标签的相关 API 概览以及 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
图片标签	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。

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。

使用示例

```
function detectLabel() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/dog.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 固定值detect-label;是否必传：是
        'ci-process': 'detect-label',
        // 本次调用支持的识别场景，可选值如下：web，针对网络图片优化；camera，针对手机摄像头拍摄图片优化；album，针对手机相册、网盘产品优化；news，针对新闻、资讯、广电等行业优化；如果不传此参数，则默认为camera。支持多场景（scenes）一起检测，以，分隔。例如，使用scenes=web，camera 即对一张图片使用两个模型同时检测，输出两套识别结果。;是否必传：否
        // scenes: '',
```

```
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url
时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写
Key detect-url 示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后为
http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg;是否必传：否
        // 'detect-url': '',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
detectLabel();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-proces	固定值 detect-label	String	是

s			
scenes	本次调用支持的识别场景，如果不传此参数，则默认为 camera。可选值如下： • web，针对网络图片优化； • camera，针对手机摄像头拍摄图片优化； • album，针对手机相册、网盘产品优化； • news，针对新闻、资讯、广电等行业优化； 支持多场景（scenes）一起检测，以，分隔。例如，使用 scenes = web，camera 即对一张图片使用两个模型同时检测，输出两套识别结果。	String	否
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。 detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果详情请参见 图片标签	Object

相关链接

- API 文档
- 功能指南

版权所有：腾讯云计算（北京）有限责任公司

图片质量评分

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于数据万象图片质量评分的 API 概览以及 SDK 示例代码。

注意：

- COS 小程序 SDK 版本需要大于等于 v1.1.1。
- 图片支持格式：PNG、JPG、JPEG。
- 图片大小：所下载图片经 Base64 编码后不超过 4MB。
- 图片像素：建议大于 50*50 像素，否则影响识别效果。
- 长宽比：建议长边与短边的比例小于 5:1。

API	说明
图片质量评分	图片质量评分功能为同步请求方式，您可以通过本接口对图片文件进行检测，从多方面评估，最终给出综合可观的清晰度评分和主观的美观度评分。

图片质量评分

功能说明

图片质量评分功能为同步请求方式，您可以通过本接口对图片文件进行检测，从多方面评估，最终给出综合可观的清晰度评分和主观的美观度评分。该接口属于 GET 请求。

使用示例

```
function assessQuality() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/dog.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，图像质量检测固定为 AssessQuality。;是否必传：是
```

```
        'ci-process': 'AssessQuality',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
assessQuality();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，必须，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，图像质量检测固定为 AssessQuality。	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果 详情请参见 图片质量评分	Object

相关链接

API 文档
功能指南

图片文字识别（OCR）

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于图片文字识别(OCR)的相关 API 概览以及 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
图片文字识别(OCR)	通用文字识别功能（Optical Character Recognition，OCR）基于行业前沿的深度学习技术，将图片上的文字内容，智能识别为可编辑的文本，可应用于随手拍扫描、纸质文档电子化、电商广告审核等多种场景，大幅提升信息处理效率。

图片文字识别(OCR)

功能说明

通用文字识别功能（Optical Character Recognition，OCR）基于行业前沿的深度学习技术，将图片上的文字内容，智能识别为可编辑的文本，可应用于随手拍扫描、纸质文档电子化、电商广告审核等多种场景，大幅提升信息处理效率。

使用示例

```
function cOSOCR() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/0.jpg', // 对象文件名，例如：folder/document.jpg；与detect-url二选一
传递
      Query: {
        // 数据万象处理能力，图片文字识别固定为OCR;是否必传：是
        'ci-process': 'OCR',
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url
时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写
```

```
Key。detect-url 示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后为http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg;是否必传：否
        // 'detect-url': '',
        // ocr的识别类型，有效值为general，accurate，efficient，fast，handwriting。general表示通用印刷体识别；accurate表示印刷体高精度版；efficient表示印刷体精简版；fast表示印刷体高速版；handwriting表示手写体识别。默认值为general。;是否必传：否
        type: 'general',
        // type值为general时有效，表示识别语言类型。支持自动识别语言类型，同时支持自选语言种类，默认中英文混合(zh)，各种语言均支持与英文混合的文字识别。可选值：zh：中英混合zh_rare：支持英文、数字、中文生僻字、繁体字，特殊符号等auto：自动mix：混合语种jap：日语kor：韩语spa：西班牙语fre：法语ger：德语por：葡萄牙语vie：越语may：马来语rus：俄语ita：意大利语hol：荷兰语swe：瑞典语fin：芬兰语dan：丹麦语nor：挪威语hun：匈牙利语tha：泰语hi：印地语ara：阿拉伯语;是否必传：否
        'language-type': 'zh',
        // type值为general，fast时有效，表示是否开启PDF识别，有效值为true和false，默认值为false，开启后可同时支持图片和PDF的识别。;是否必传：否
        ispdf: false,
        // type值为general，fast时有效，表示需要识别的PDF页面的对应页码，仅支持PDF单页识别，当上传文件为PDF且ispdf参数值为true时有效，默认值为1。;是否必传：否
        // 'pdf-pagenumber': 0,
        // type值为general，accurate时有效，表示识别后是否需要返回单字信息，有效值为true和false，默认为false;是否必传：否
        isword: false,
        // type值为handwriting时有效，表示是否开启单字的四点定位坐标输出，有效值为true和false，默认值为false。;是否必传：否
        'enable-word-polygon': false,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
cOSOCR();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，图片文字识别固定为 OCR	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。 detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`	String	否
type	OCR的识别类型，默认值为general。有效值有： • general：表示通用印刷体识别。 • accurate：表示印刷体高精度版。 • efficient：表示印刷体精简版。 • fast：表示印刷体高速版。 • handwriting：表示手写体识别。	String	否
languageType	type值为general时有效，表示识别语言类型。支持自动识别语言类型，同时支持自选语言种类，默认中英文混合(zh)。 各种语言均支持与英文混合的文字识别。可选值有： • zh：中英混合 • zh_rare：支持英文、数字、中文生僻字、繁体字，特殊符号等 • auto：自动 • mix：混合语种 • jap：日语	String	否

	• kor：韩语 • spa：西班牙语 • fre：法语 • ger：德语 • por：葡萄牙语 • vie：越语 • may：马来语 • rus：俄语 • ita：意大利语 • hol：荷兰语 • swe：瑞典语 • fin：芬兰语 • dan：丹麦语 • nor：挪威语 • hun：匈牙利语 • tha：泰语 • hi：印地语 • ara：阿拉伯语		
ispdf	type 值为 general，fast 时有效，表示是否开启 PDF 识别，有效值为 true 和 false，默认值为 false，开启后可同时支持图片和 PDF 的识别。	boolean	否
pdfPagenumber	type 值为 general，fast 时有效，表示需要识别的 PDF 页面的对应页码，仅支持 PDF 单页识别，当上传文件为 PDF 且 ispdf 参数值为 true 时有效，默认值为1。	Integer	否
isword	type 值为 general，accurate 时有效，表示识别后是否需要返回单字信息，有效值为 true 和false，默认为 false。	boolean	否
enableWordPolygon	type 值为 handwriting 时有效，表示是否开启单字的四点定位坐标输出，有效值为 true 和false，默认值为 false。	boolean	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 图片文字识别(OCR)	Object

相关链接

API 文档
功能指南

实时文字翻译

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于实时文字翻译的相关 API 概览以及 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
实时文字翻译	腾讯云数据万象通过 AutoTranslationBlock 接口对文字块进行翻译，请求时需要携带签名。

实时文字翻译

功能说明

腾讯云数据万象通过 AutoTranslationBlock 接口对文字块进行翻译，请求时需要携带签名。

使用示例

```
function autoTranslationBlock() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Query: {
        // 数据万象处理能力，文本块翻译固定为 AutoTranslationBlock。;是否必传：是
        'ci-process': 'AutoTranslationBlock',
        // 待翻译的文本;是否必传：是
        InputText: '我是张三',
        // 输入语言，如 "zh";是否必传：是
        SourceLang: 'zh',
        // 输出语言，如 "en";是否必传：是
        TargetLang: 'en',
        // 文本所属业务领域，如：  "ecommerce", //缺省值为 general;是否必传：否
        // TextDomain: '',
```

```
          // 文本类型，如: "title", //缺省值为 sentence;是否必传：否
          // TextStyle: '',
        },
      },
      function (err, data) {
        if (err) {
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data);
        }
      }
    );
}
autoTranslationBlock();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，文本块翻译固定为 AutoTranslationBlock。	String	是
InputText	待翻译的文本	String	是
SourceLang	输入语言，如 "zh"	String	是

TargetLang	输出语言，如 "en"	String	是
TextDomain	文本所属业务领域，如: "ecommerce", 缺省值为 general	String	否
TextStyle	文本类型，如: "title", 缺省值为 sentence	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果，详情请参见 实时文字翻译	Object

相关链接

API 文档

人脸识别
人脸检测

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于人脸检测的相关 API 概览以及 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
人脸检测	人脸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人脸位置。该接口属于 GET 请求。

人脸检测

功能说明

人脸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人脸位置。该接口属于 GET 请求。

使用示例

```
function aIDetectFace() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/1.jpg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，人脸特效固定为 DetectFace。;是否必传：是
        'ci-process': 'DetectFace',
        // 最多处理的人脸数目。默认值为1（仅检测图片中面积最大的那张人脸），最大值为120。此参数用于控制处理待检测图片中的人脸个数，值越小，处理速度越快。;是否必传：否
        'max-face-num': 1,
      },
    },
```

```
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
aIDetectFace();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg。	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，人脸特效固定为 DetectFace。	String	是
max-face-num	最多处理的人脸数目。默认值为1（仅检测图片中面积最大的那张人脸），最大值为120。此参数用于控制处理待检测图片中的人脸个数，值越小，处理速度越快。	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 人脸检测	Object

相关链接

API 文档

人脸特效

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于人脸特效的相关 API 概览以及 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
人脸特效	人脸特效，提供人脸美颜、人像变换、人像分割功能。

人脸特效

功能说明

人脸特效，提供人脸美颜、人像变换、人像分割功能。

使用示例

```
function aIFaceEffect() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/1.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 万象处理能力，人脸特效固定为face-effect;是否必传：是
        'ci-process': 'face-effect',
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 ObjectKey ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 ObjectKey detect-url 示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后为http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg。;是否必传：否
        // 'detect-url': '',
```

```
        // 人脸特效类型，人脸美颜：face-beautify；人脸性别转换：face-gender-transformation；人脸年龄变化：face-age-transformation；人像分割：face-segmentation;是否必传：是
        type: 'face-beautify',
        // type为face-beautify时生效，美白程度，取值范围[0,100]。0不美白，100代表最高程度。默认值30;是否必传：否
        whitening: 50,
        // type为face-beautify时生效，磨皮程度，取值范围[0,100]。0不磨皮，100代表最高程度。默认值10;是否必传：否
        smoothing: 50,
        // type为face-beautify时生效，瘦脸程度，取值范围[0,100]。0不瘦脸，100代表最高程度。默认值70;是否必传：否
        faceLifting: 50,
        // type为face-beautify时生效，大眼程度，取值范围[0,100]。0不大眼，100代表最高程度。默认值70;是否必传：否
        eyeEnlarging: 50,
        // type为face-gender-transformation时生效，选择转换方向，0：男变女，1：女变男。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。;是否必传：否
        // gender: 0,
        // type为face-age-transformation时生效，变化到的人脸年龄,[10,80]。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。;是否必传：否
        // age: 0,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
aIFaceEffect();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	万象处理能力，人脸特效固定为 face-effect。	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg` 。	String	否
type	人脸特效类型，可选值有： • face-beautify：人脸美颜； • face-gender-transformation：人脸性别转换 • face-age-transformation：人脸年龄变化 • face-segmentation：人像分割	String	是
whitening	type 为 face-beautify 时生效，美白程度，取值范围[0,100]。0不美白，100代表最高程度。默认值30	Integer	否
smoothing	type 为 face-beautify 时生效，磨皮程度，取值范围[0,100]。0不磨皮，100代表最高程度。默认值10	Integer	否
faceLifting	type 为 face-beautify 时生效，瘦脸程度，取值范围[0,100]。0不瘦脸，100代表最高程度。默认值70	Integer	否
eyeEnlarging	type 为 face-beautify 时生效，大眼程度，取值范围[0,100]。0不大眼，100代表最高程度。默认值70	Integer	否
gender	type 为 face-gender-transformation 时生效，选择转换方向，0：男变女，1：女变男。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。	Integer	否

age	type为 face-age-transformation 时生效，变化到的人脸年龄，[10,80]。无默认值，为必选项。限制：仅对图片中面积最大的人脸进行转换。	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 人脸特效	Object

相关链接

API 文档
功能指南

人体识别

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于人体识别的相关 API 概览以及 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
人体识别	腾讯云数据万象通过 AIBodyRecognition 接口识别并输出画面中人体，输出其位置（矩形框）和置信度。图片人体识别请求包属于 GET 请求，请求时需要携带签名。

人体识别

功能说明

腾讯云数据万象通过 AIBodyRecognition 接口识别并输出画面中人体，输出其位置（矩形框）和置信度。图片人体识别请求包属于 GET 请求，请求时需要携带签名。

使用示例

```
function aIBodyRecognition() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/1.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，人体识别固定为AIBodyRecognition;是否必传：是
        'ci-process': 'AIBodyRecognition',
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url
时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写
Key。 detect-url 示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后
为http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg;是否必传：否
        // 'detect-url': '',
```

```
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
aIBodyRecognition();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，人体识别固定为 AIBodyRecognition	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接，无需再填写 Key。	String	否

	detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`		

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 人体识别	Object

相关链接

API 文档

宠物识别

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于宠物识别的相关 API 概览以及 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
宠物识别	腾讯云数据万象通过 detect-pet 接口识别并输出画面中宠物，输出其位置（矩形框）和置信度。图片宠物识别请求包属于 GET 请求，请求时需要携带签名。

宠物识别

功能说明

腾讯云数据万象通过 detect-pet 接口识别并输出画面中宠物，输出其位置（矩形框）和置信度。图片宠物识别请求包属于 GET 请求，请求时需要携带签名。

使用示例

```
function aIDetectPet() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'dog.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，宠物识别固定为 detect-pet;是否必传：是
        'ci-process': 'detect-pet',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
```

```
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
aIDetectPet();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，宠物识别固定为 detect-pet	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果，详情请参见 宠物识别	Object

相关参考

API 文档

汽车识别

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于车辆车牌识别的相关 API 概览以及 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
车辆车牌识别	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。

车辆车牌识别

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。

使用示例

```
function aIDetectCar() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'car.png', // 对象文件名，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，车辆识别固定为 DetectCar;是否必传：是
        'ci-process': 'DetectCar',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
```

```
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
aIDetectCar();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，车辆识别固定为 DetectCar	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 车辆车牌识别	Object

相关链接

API 文档
功能指南

卡证识别

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于卡证识别的相关 API 概览以及 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
卡证识别	本接口支持中国大陆居民二代身份证正面（暂不支持背面）、驾驶证主页（暂不支持副页）所有字段的自动定位，暂不支持文本识别，用于对特定字段的抹除、屏蔽，以及进一步的文本识别。

卡证识别

功能说明

本接口支持中国大陆居民二代身份证正面（暂不支持背面）、驾驶证主页（暂不支持副页）所有字段的自动定位，暂不支持文本识别，用于对特定字段的抹除、屏蔽，以及进一步的文本识别。

使用示例

```
function aILicenseRec() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'card.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，卡证识别固定为AILicenseRec;是否必传：是
        'ci-process': 'AILicenseRec',
        // 您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url
时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。detect-url
示例：http://www.example.com/abc.jpg ，需要进行 UrlEncode，处理后为
http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg;是否必传：否
        // 'detect-url': '',
```

```
        // 卡证识别类型，有效值为IDCard，DriverLicense。<br>IDCard表示身份证；
DriverLicense表示驾驶证，默认：DriverLicense;是否必传：是
        CardType: 'IDCard',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
aILicenseRec();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg。	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，卡证识别固定为AILicenseRec	String	是

detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 、Key ，填写了 detect-url 时，后台会处理 detect-url 链接 。 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为 `http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg` 。	String	否
CardType	卡证识别类型，默认为DriverLicense。有效值为： • IDCard：表示身份证。 • DriverLicense：表示驾驶证。	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
_Response	响应结果 详情请参见 卡证识别	Object

相关链接

API 文档

游戏场景识别

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于游戏场景识别的相关 API 概览以及 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
游戏场景识别	游戏标签功能实现游戏图片场景的识别，返回图片中置信度较高的游戏类别标签。游戏标签识别请求包属于 GET 请求，请求时需要携带签名。

游戏场景识别

功能说明

游戏标签功能实现游戏图片场景的识别，返回图片中置信度较高的游戏类别标签。游戏标签识别请求包属于 GET 请求，请求时需要携带签名。

使用示例

```
function aIGameRec() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/game.jpeg', // 对象文件名，例如：folder/document.jpg；与detect-url
二选一传递
      Query: {
        // 数据万象处理能力，游戏场景识别固定为 AIGameRec;是否必传：是
        'ci-process': 'AIGameRec',
        // 待检查图片url，需要进行urlencode;与Key二选一传递
        // 'detect-url': '',
      },
    },
```

```
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
aIGameRec();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，游戏场景识别固定为 AIGameRec。	String	是
detect-url	处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接。 detect-url 示例：`http://www.example.com/abc.jpg` ，需要进行 UrlEncode，处理后为	String	否

	`http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`。		

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果 详情请参见 游戏场景识别	Object

相关参考

API 文档

人脸核身
身份证识别

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于身份证识别的相关 API 概览以及 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
身份证识别	支持中国大陆居民二代身份证正反面所有字段的识别，包括姓名、性别、民族、出生日期、住址、公民身份证号、签发机关、有效期限；具备身份证照片、人像照片的裁剪功能和翻拍、PS、复印件告警功能，以及边框和框内遮挡告警、临时身份证告警和身份证有效期不合法告警等扩展功能。

身份证识别

功能说明

支持中国大陆居民二代身份证正反面所有字段的识别，包括姓名、性别、民族、出生日期、住址、公民身份证号、签发机关、有效期限；具备身份证照片、人像照片的裁剪功能和翻拍、PS、复印件告警功能，以及边框和框内遮挡告警、临时身份证告警和身份证有效期不合法告警等扩展功能。

使用示例

```
function aIIDCardOCR() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/card.jpeg', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，身份证识别固定为 IDCardOCR;是否必传：是
        'ci-process': 'IDCardOCR',
```

```
        // FRONT：身份证有照片的一面（人像面）BACK：身份证有国徽的一面（国徽面）该参数如果不填，将为您自动判断身份证正反面;是否必传：否
        CardSide: 'FRONT',
        // 以下可选字段均为 bool 类型，默认 false：CropIdCard，身份证照片裁剪（去掉证件外多余的边缘、自动矫正拍摄角度）CropPortrait，人像照片裁剪（自动抠取身份证头像区域）CopyWarn，复印件告警BorderCheckWarn，边框和框内遮挡告警ReshootWarn，翻拍告警DetectPsWarn，PS 检测告警TempIdWarn，临时身份证告警InvalidDateWarn，身份证有效日期不合法告警Quality，图片质量分数（评价图片的模糊程度）MultiCardDetect，是否开启多卡证检测参数设置方式参考：Config = {"CropIdCard":true,"CropPortrait":true};是否必传：否
        // Config: {},
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
aIIDCardOCR();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，身份证识别固定为 IDCardOCR	String	是
CardSide	FRONT：身份证有照片的一面（人像面） BACK：身份证有国徽的一面（国徽面） 该参数如果不填，将为您自动判断身份证正反面	String	否
Config	以下可选字段均为 bool 类型，默认 false： • CropIdCard，身份证照片裁剪（去掉证件外多余的边缘、自动矫正拍摄角度） • CropPortrait，人像照片裁剪（自动抠取身份证头像区域） • CopyWarn，复印件告警 • BorderCheckWarn，边框和框内遮挡告警 • ReshootWarn，翻拍告警 • DetectPsWarn，PS 检测告警 • TempIdWarn，临时身份证告警 • InvalidDateWarn，身份证有效日期不合法告警 • Quality，图片质量分数（评价图片的模糊程度） • MultiCardDetect，是否开启多卡证检测 参数设置方式参考： Config = {"CropIdCard":true,"CropPortrait":true}	String	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Respon se	响应结果 详情请参见 身份证识别	Object

相关参考

API 文档

活体人脸核身

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于活体人脸核身的相关 API 概览以及 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
活体人脸核身	集成了活体检测和跟权威库进行比对的能力，传入一段视频和姓名、身份证号信息即可进行验证。对录制的自拍视频进行活体检测，从而确认当前用户为真人，可防止照片、视频、静态3D建模等各种不同类型的攻击。检测为真人后，再判断该视频中的人与权威库的证件照是否属于同一个人，实现用户身份信息核实。

活体人脸核身

功能说明

集成了活体检测和跟权威库进行比对的能力，传入一段视频和姓名、身份证号信息即可进行验证。对录制的自拍视频进行活体检测，从而确认当前用户为真人，可防止照片、视频、静态3D建模等各种不同类型的攻击。检测为真人后，再判断该视频中的人与权威库的证件照是否属于同一个人，实现用户身份信息核实。

使用示例

```
function livenessRecognition() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/tf.mp4', // 对象文件名，必须，例如：folder/document.jpg
      Query: {
        // 数据万象处理能力，人脸核身固定为 LivenessRecognition;是否必传：是
        'ci-process': 'LivenessRecognition',
        // 身份证号;是否必传：是
        IdCard: 'xxxxxxx',
```

```
        // 姓名。中文请使用 UTF-8编码;是否必传：是
        Name: '张三',
        // 活体检测类型，取值：LIP/ACTION/SILENTLIP 为数字模式，ACTION 为动作模式，
SILENT 为静默模式，三种模式选择一种传入;是否必传：是
        LivenessType: 'SILENT',
        // 数字模式传参：数字验证码（1234），需先调用接口获取数字验证码动作模式传参：传动作
顺序（2，1 or 1，2），需先调用接口获取动作顺序静默模式传参：空;是否必传：否
        ValidateData: '',
        // 需要返回多张最佳截图，取值范围1 - 10，不设置默认返回一张最佳截图;是否必传：否
        BestFrameNum: 1,
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
livenessRecognition();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填

ci-process	数据万象处理能力，人脸核身固定为 LivenessRecognition	String	是
IdCard	身份证号	String	是
Name	姓名。中文请使用 UTF-8编码	String	是
LivenessType	活体检测类型，取值：LIP/ACTION/SILENTLIP 为数字模式，ACTION 为动作模式，SILENT 为静默模式，三种模式选择一种传入	String	是
ValidateData	数字模式传参：数字验证码（1234），需先调用接口获取数字验证码动作模式传参：传动作顺序（2，1 or 1，2），需先调用接口获取动作顺序静默模式传参：空	String	否
BestFrameNum	需要返回多张最佳截图，取值范围1 - 10，不设置默认返回一张最佳截图	Integer	否

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果 详情请参见 活体人脸核身	Object

相关参考

API 文档
功能指南

版权所有：腾讯云计算（北京）有限责任公司

获取数字验证码

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于获取数字验证码的相关 API 概览以及 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
获取数字验证码	使用数字活体检测模式前，需调用本接口获取数字验证码。

获取数字验证码

功能说明

使用数字活体检测模式前，需调用本接口获取数字验证码。

使用示例

```
function getLiveCode() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Query: {
        // 数据万象处理能力，获取数字验证码固定为 GetLiveCode;是否必传：是
        'ci-process': 'GetLiveCode',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
```

```
            console.log(data.Response);
        }
    }
  );
}
getLiveCode();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，获取数字验证码固定为 GetLiveCode	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 获取数字验证码	Object

相关参考

API 文档

获取动作顺序

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于获取动作顺序的相关 API 概览以及 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
获取动作顺序	使用动作活体检测模式前，需调用本接口获取动作顺序。

获取动作顺序

功能说明

使用动作活体检测模式前，需调用本接口获取动作顺序。

使用示例

```
function getActionSequence() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Query: {
        // 数据万象处理能力，获取动作顺序固定为 GetActionSequence;是否必传：是
        'ci-process': 'GetActionSequence',
      },
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
```

```
            console.log(data.Response);
        }
    }
  );
}
getActionSequence();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，获取动作顺序固定为 GetActionSequence	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 获取动作顺序	Object

相关参考

API 文档

商品抠图

最近更新时间：2024-12-27 16:37:08

简介

本文档提供关于商品抠图的相关 API 概览以及 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
商品抠图	腾讯云数据万象通过 GoodsMatting 接口检测图片中的商品信息，生成只包含商品信息的图片，支持持久化、云上处理及下载时处理。

商品抠图

功能说明

腾讯云数据万象通过 GoodsMatting 接口检测图片中的商品信息，生成只包含商品信息的图片，支持持久化、云上处理及下载时处理。

使用示例

```
function goodsMatting() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  cos.request(
    {
      Bucket: config.Bucket,
      Region: config.Region,
      Method: 'GET', // 固定值，必须
      Key: 'ci/goods.jpeg', // 对象文件名，例如：folder/document.jpg；与detect-url
二选一传递
      Query: {
        // ;是否必传：是
        'ci-process': 'GoodsMatting',
        // ;是否必传：否
        // 'detect-url': '',
      },
    },
```

```
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data);
      }
    }
  );
}
goodsMatting();
```

参数说明

cos.request 方法参数说明：

参数名称	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
Method	固定值：GET	String	是
Key	对象文件名，例如：folder/document.jpg；	String	是
Query	其他请求参数	Container	是

Query 参数说明：

参数名称	参数描述	类型	是否必填
ci-process	数据万象处理能力，商品抠图固定为 GoodsMatting。	String	是
detect-url	您可以通过填写 detect-url 处理任意公网可访问的图片链接。不填写 detect-url 时，后台会默认处理 Key ，填写了 detect-url 时，后台会处理 detect-url 链接 。	String	否

	示例：`http://www.example.com/abc.jpg`，需要进行 UrlEncode，处理后为`http%25253A%25252F%25252Fwww.example.com%25252Fabc.jpg`。		

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果 详情请参见 商品抠图	Object

相关链接

API 文档
功能指南

内容翻译
提交任务

最近更新时间：2024-12-27 16:37:08

简介

内容翻译功能是由 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。
本文档提供关于内容翻译的相关 API 概览以及 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
提交任务	提交一个翻译任务。

提交任务

功能说明

提交一个翻译任务。

使用示例

```
function postTranslation() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'jobs'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 创建任务的 Tag：Translation;是否必传：是
      Tag: 'Translation',
      // 待操作的对象信息;是否必传：是
      Input: {
        // 源文档文件名单文件（docx/xlsx/html/markdown/txt）：不超过800万字符有页数的（pdf/pptx）：不超过300页文本文件（txt）：不超过10MB二进制文件（pdf/docx/pptx/xlsx）：
```

```
不超过60MB图片文件（jpg/jpeg/png/webp）：不超过10MB;是否必传：是
        Object: 'ci/test.txt',
        // 文档语言类型zh：简体中文zh-hk：繁体中文zh-tw：繁体中文zh-tr：繁体中文en：英语
ar：阿拉伯语de：德语es：西班牙语fr：法语id：印尼语it：意大利语ja：日语pt：葡萄牙语ru：俄语
ko：韩语km：高棉语lo：老挝语;是否必传：是
        Lang: 'zh-hk',
        // 文档类型pdfdocxpptxxlsxtxtxmlhtml：只能翻译 HTML 里的文本节点，需要通过 JS
动态加载的不进行翻译markdownjpgjpegpngwebp;是否必传：是
        Type: 'txt',
        // 原始文档类型仅在 Type=pdf/jpg/jpeg/png/webp 时使用，当值为pdf时，仅支持
docx、pptx当值为jpg/jpeg/png/webp时，仅支持txt;是否必传：否
        // BasicType: '',
      },
      // 操作规则;是否必传：是
      Operation: {
        // 翻译参数;是否必传：是
        Translation: {
          // 目标语言类型源语言类型为 zh/zh-hk/zh-tw/zh-tr 时支持：en、ar、de、es、
fr、id、it、ja、it、ru、ko、km、lo、pt源语言类型为 en 时支持：zh、zh-hk、zh-tw、zh-
tr、ar、de、es、fr、id、it、ja、it、ru、ko、km、lo、pt其他类型时支持：zh、zh-hk、zh-
tw、zh-tr、en;是否必传：是
          Lang: 'en',
          // 文档类型，源文件类型与目标文件类型映射关系如下：docx：docxpptx：pptxxlsx：
xlsxtxt：txtxml：xmlhtml：htmlmarkdown：markdownpdf：pdf、docxpng：txtjpg：
txtjpeg：txtwebp：txt;是否必传：是
          Type: 'txt',
        },
        // 结果输出地址，当NoNeedOutput为true时非必选;是否必传：否
        Output: {
          // 存储桶的地域;是否必传：是
          Region: config.Region,
          // 存储结果的存储桶;是否必传：是
          Bucket: config.Bucket,
          // 输出结果的文件名;是否必传：是
          Object: 'ci/trans_test.txt',
        },
        // 透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
        // UserData: '',
        // 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;是否必传：否
        // JobLevel: '',
        // 仅输出结果，不生成结果文件。取值：true/false。该参数原文档类型为图片时有效。默
认值 false;是否必传：否
        // NoNeedOutput: '',
      },
      // 任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
      // CallBackFormat: '',
```

```
      // 任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
      // CallBackType: '',
      // 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;是否必传：否
      // CallBack: '',
      // 任务回调TDMQ配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig;是否必传：否
      // CallBackMqConfig: {
      // 消息队列所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）;是否必传：是
      // MqRegion: '',
      // 消息队列使用模式，默认 Queue ：主题订阅：Topic队列服务：Queue;是否必传：是
      // MqMode: '',
      // TDMQ 主题名称;是否必传：是
      // MqName: '',
      // },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
postTranslation();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填

Method	固定值：POST	String	是
Url	请求url，固定值	String	是
Key	固定值：jobs	String	是
ContentType	固定值	String	是
Body	请求体，详情请参见 提交任务	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 提交任务	Object

相关链接

API 文档

词性分析
提交任务

最近更新时间：2024-12-27 16:37:09

简介

词性分析功能是由 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。
本文档提供关于词性分析的相关 API 概览以及 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
提交任务	提交一个分词任务。

提交任务

功能说明

提交一个分词任务。

使用示例

```
function postWordsGeneralize() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'jobs'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 创建任务的 Tag：WordsGeneralize;是否必传：是
      Tag: 'WordsGeneralize',
      // 待操作的对象信息;是否必传：是
      Input: {
        // 文件路径;是否必传：是
        Object: 'test.txt',
```

```
      },
      // 操作规则;是否必传：是
      Operation: {
        // 指定分词参数;是否必传：是
        WordsGeneralize: {
          // ner 方式，支持 NerBasic 和 DL，默认值 DL;是否必传：否
          NerMethod: 'DL',
          // 分词粒度，支持 SegBasic 和 MIX，默认值 MIX;是否必传：否
          SegMethod: 'MIX',
        },
        // 透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
        // UserData: '',
        // 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;是否必传：否
        // JobLevel: '',
      },
      // 任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
      // CallBackFormat: '',
      // 任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
      // CallBackType: '',
      // 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;是否必传：否
      // CallBack: '',
      // 任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig;是否必传：否
      // CallBackMqConfig: {
      // 消息队列所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）;是否必传：是
      // MqRegion: '',
      // 消息队列使用模式，默认 Queue ：主题订阅：Topic队列服务：Queue;是否必传：是
      // MqMode: '',
      // TDMQ 主题名称;是否必传：是
      // MqName: '',
      // },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
```

```
          // 处理请求失败
          console.log(err);
        } else {
          // 处理请求成功
          console.log(data.Response);
        }
      }
    );
}
postWordsGeneralize();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Key	固定值：jobs	String	是
Url	请求url，固定值	String	是
Body	请求体，详情请参见 提交任务	Object	是
ContentType	固定值	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Respon se	响应结果，详情请参见 提交任务	Object

相关链接

API 文档

视频目标检测
提交任务

最近更新时间：2024-12-27 16:37:09

简介

提交一个视频目标检测任务功能是由 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。
本文档提供关于提交一个视频目标检测任务的相关 API 概览以及 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
提交任务	提交一个视频目标检测任务。

提交任务

功能说明

提交一个视频目标检测任务

使用示例

```
function postVideoTargetRec() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'jobs'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 创建任务的 Tag：VideoTargetRec;是否必传：是
      Tag: 'VideoTargetRec',
      // 操作规则;是否必传：是
      Operation: {
        // 视频目标检测模板 ID;与VideoTargetRec二选一传递
        TemplateId: 'xxxx',
```

```
        // 视频目标检测参数，同创建视频目标检测模板接口中的 Request.VideoTargetRec;与TemplateId二选一传递
        // VideoTargetRec: {
        // 是否开启人体检测，取值 true/false;是否必传：否
        // Body: '',
        // 是否开启宠物检测，取值 true/false;是否必传：否
        // Pet: '',
        // 是否开启车辆检测，取值 true/false;是否必传：否
        // Car: '',
        // },
        // 透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
        // UserData: '',
        // 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;是否必传：否
        // JobLevel: '',
      },
      // 待操作的媒体信息;是否必传：是
      Input: {
        // 媒体文件名;是否必传：否
        Object: 'ci/test.mp4',
      },
      // 任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
      // CallBackFormat: '',
      // 任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
      // CallBackType: '',
      // 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;是否必传：否
      // CallBack: '',
      // 任务回调TDMQ配置，当 CallBackType 为 TDMQ 时必填。详情请参见 CallBackMqConfig;是否必传：否
      // CallBackMqConfig: {
      // 消息队列所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）;是否必传：是
      // MqRegion: '',
      // 消息队列使用模式，默认 Queue ：主题订阅：Topic队列服务：Queue;是否必传：是
      // MqMode: '',
      // TDMQ 主题名称;是否必传：是
      // MqName: '',
      // },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
```

```
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
postVideoTargetRec();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Url	请求url，固定值	String	是
Key	固定值：jobs	String	是
ContentType	固定值	String	是
Body	请求体，详情请参见 提交任务	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
– Response	响应结果，详情请参见 提交任务	Object

相关参考

API 文档
功能指南

创建模板

最近更新时间：2024-12-27 16:37:09

简介

创建视频目标检测模板功能是由 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。
本文档提供关于创建视频目标检测模板的相关 API 概览以及 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
创建模板	创建视频目标检测模板。

创建模板

功能说明

创建视频目标检测模板。

使用示例

```
function postVideoTargetTemplete() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'template'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 模板类型：VideoTargetRec;是否必传：是
      Tag: 'VideoTargetRec',
      // 模板名称，仅支持中文、英文、数字、_、-和*，长度不超过 64;是否必传：是
      Name: 'my_videoTargetRec',
      // 视频目标检测 参数;是否必传：是
      VideoTargetRec: {
        // 是否开启人体检测，取值 true/false;是否必传：否
        Body: 'true',
```

```
        // 是否开启宠物检测，取值 true/false;是否必传：否
        Pet: 'true',
        // 是否开启车辆检测，取值 true/false;是否必传：否
        Car: 'false',
      },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
postVideoTargetTemplete();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Url	请求url，固定值	String	是
Key	固定值：template	String	是
ContentType	固定值	String	是
Body	请求体，详情请参见 创建模板	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
- Response	响应结果，详情请参见 创建模板	Object

相关参考

API 文档
功能指南

更新模板

最近更新时间：2024-12-27 16:37:09

简介

更新视频目标检测模板功能是由 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。
本文档提供关于更新视频目标检测模板的相关 API 概览以及 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
更新模板	更新视频目标检测模板。

更新模板

功能说明

更新视频目标检测模板。

使用示例

```
function updateVideoTargetTemplete() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const templateId = 'xxxxxxx'; // 要更新的模板id
  const key = `template/${templateId}`; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 模板类型：VideoTargetRec;是否必传：是
      Tag: 'VideoTargetRec',
      // 模板名称，仅支持中文、英文、数字、_、-和*，长度不超过 64;是否必传：是
      Name: 'my_videoTargetRec2',
      // 视频目标检测 参数;是否必传：是
      VideoTargetRec: {
        // 是否开启人体检测，取值 true/false;是否必传：否
```

```
        Body: 'false',
        // 是否开启宠物检测，取值 true/false;是否必传：否
        Pet: 'false',
        // 是否开启车辆检测，取值 true/false;是否必传：否
        Car: 'true',
      },
    },
  });

  cos.request(
    {
      Method: 'PUT', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
updateVideoTargetTemplete();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：PUT	String	是
Url	请求url，固定值	String	是
Key	template/{TemplateId}	String	是
ContentType	固定值	String	是
Body	请求体，详情请参见 更新模板	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
-Response	响应结果，详情请参见 更新模板	Object

相关参考

API 文档
功能指南

视频人像分割
提交任务

最近更新时间：2024-12-27 16:37:09

简介

视频人像分割功能是由 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。
本文档提供关于视频人像分割的相关 API 概览以及 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
提交任务	提交一个视频人像抠图任务。

提交任务

功能说明

提交一个视频人像抠图任务。

使用示例

```
function postSegmentVideoBody() {
  // sdk引入以及初始化请参考：https://cloud.tencent.com/document/product/436/31953
  const config = {
    // 需要替换成您自己的存储桶信息
    Bucket: 'examplebucket-1250000000', // 存储桶，必须字段
    Region: 'COS_REGION', // 存储桶所在地域，必须字段 如 ap-beijing
  };
  const key = 'jobs'; // 固定值
  const host = `${config.Bucket}.ci.${config.Region}.myqcloud.com`;
  const url = `https://${host}/${key}`;
  const body = COS.util.json2xml({
    Request: {
      // 创建任务的 Tag：SegmentVideoBody;是否必传：是
      Tag: 'SegmentVideoBody',
      // 待操作的对象信息;是否必传：是
      Input: {
        // 文件路径;是否必传：是
        Object: 'ci/test.mp4',
```

```
        },
        // 操作规则;是否必传：是
        Operation: {
          // 视频人像抠图配置;是否必传：否
          SegmentVideoBody: {
            // 抠图模式 Mask：输出alpha通道结果Foreground：输出前景视频Combination：输出抠图后的前景与自定义背景合成后的视频默认值：Mask;是否必传：否
            Mode: 'Mask',
            // 抠图类型HumanSeg：人像抠图GreenScreenSeg：绿幕抠图SolidColorSeg：纯色背景抠图默认值：HumanSeg;是否必传：否
            // SegmentType: '',
            // mode为 Foreground 时参数生效，背景颜色为红色，取值范围 [0, 255]， 默认值为 0;是否必传：否
            // BackgroundRed: '',
            // mode为 Foreground 时参数生效，背景颜色为绿色，取值范围 [0, 255]，默认值为 0;是否必传：否
            // BackgroundGreen: '',
            // mode为 Foreground 时参数生效，背景颜色为蓝色，取值范围 [0, 255]，默认值为 0;是否必传：否
            // BackgroundBlue: '',
            // 传入背景文件。mode为 Combination 时，此参数必填，背景文件需与源文件在同存储桶下;是否必传：否
            // BackgroundLogoUrl: '',
            // 调整抠图的边缘位置，取值范围为[0, 255]，默认值为 0;是否必传：否
            // BinaryThreshold: '',
            // 纯色背景抠图的背景色（红），当 SegmentType 为 SolidColorSeg 生效，取值范围为 [0, 255]，默认值为 0;是否必传：否
            // RemoveRed: '',
            // 纯色背景抠图的背景色（绿），当 SegmentType 为 SolidColorSeg 生效，取值范围为 [0, 255]，默认值为 0;是否必传：否
            // RemoveGreen: '',
            // 纯色背景抠图的背景色（蓝），当 SegmentType 为 SolidColorSeg 生效，取值范围为 [0, 255]，默认值为 0;是否必传：否
            // RemoveBlue: '',
          },
          // 结果输出配置;是否必传：是
          Output: {
            // 存储桶的地域;是否必传：是
            Region: config.Region,
            // 存储结果的存储桶;是否必传：是
            Bucket: config.Bucket,
            // 输出结果的文件名;是否必传：是
            Object: 'ci/out_test.mp4',
          },
          // 透传用户信息，可打印的 ASCII 码，长度不超过1024;是否必传：否
          // UserData: '',
```

```
        // 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;是否必传：否
        // JobLevel: '',
      },
      // 任务回调格式，JSON 或 XML，默认 XML，优先级高于队列的回调格式;是否必传：否
      // CallBackFormat: '',
      // 任务回调类型，Url 或 TDMQ，默认 Url，优先级高于队列的回调类型;是否必传：否
      // CallBackType: '',
      // 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;是否必传：否
      // CallBack: '',
      // 任务回调 TDMQ 配置，当 CallBackType 为 TDMQ 时必填。详情见 CallBackMqConfig;是否必传：否
      // CallBackMqConfig: {
      // 消息队列所属园区，目前支持园区 sh（上海）、bj（北京）、gz（广州）、cd（成都）、hk（中国香港）;是否必传：是
      // MqRegion: '',
      // 消息队列使用模式，默认 Queue ：主题订阅：Topic队列服务：Queue;是否必传：是
      // MqMode: '',
      // TDMQ 主题名称;是否必传：是
      // MqName: '',
      // },
    },
  });

  cos.request(
    {
      Method: 'POST', // 固定值，必须
      Key: key, // 必须
      Url: url, // 请求的url，必须
      Body: body, // 请求体参数，必须
      ContentType: 'application/xml', // 固定值，必须
    },
    function (err, data) {
      if (err) {
        // 处理请求失败
        console.log(err);
      } else {
        // 处理请求成功
        console.log(data.Response);
      }
    }
  );
}
postSegmentVideoBody();
```

参数说明

cos.request 方法参数说明：

参数名称	参数描述	类型	是否必填
Method	固定值：POST	String	是
Url	请求url，固定值	String	是
Key	固定值：jobs	String	是
ContentType	固定值	String	是
Body	请求体，详情请参见 提交任务	Object	是

回调函数说明

```
function(err, data) { ... }
```

参数名称	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
–Response	响应结果，详情请参见 提交任务	Object

相关参考

API 文档