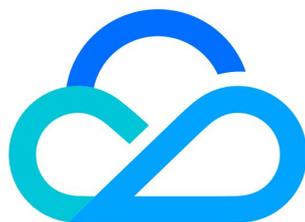


文本审核 SDK 文档



腾讯云

【 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 】



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

C++ SDK

- 快速入门

- 文本审核

.NET(C#) SDK

- 快速入门

- 文本审核

Go SDK

- 快速入门

- 文本审核

COS iOS SDK

- 快速入门

- 快速体验

- 存储桶操作

- 文本审核

Java SDK

- 快速入门

- 文本审核

- 异常处理

JavaScript SDK

- 快速入门

- 文本审核

Node.js SDK

- 快速入门

- 文本审核

PHP SDK

- 快速入门

- 文本审核

Python SDK

- 快速入门

- 文本审核

小程序 SDK

- 快速入门

- 文本审核

SDK 文档

SDK 概览

最近更新时间：2024-10-15 21:04:22

文本审核服务由腾讯云 [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](#)。

C++ SDK

快速入门

最近更新时间：2024-11-29 09:48:13

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 libcos sdk.a，动态库名称为 libcos sdk-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 `cos sdk.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 `libcos sdk.a`，动态库名称为 `libcos sdk-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'
libPocoCrypto.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'
libPocoCrypto.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 libcos sdk.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 libcos sdk。

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.IsSuccess()) {
    // 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> contents = resp.GetContents();
    for (std::vector<qcloud_cos::Content>::const_iterator itr =
contents.begin(); itr != contents.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.IsSuccess()) {
    // 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;
}
}
```

文本审核

最近更新时间：2024-11-29 09:48:13

简介

内容审核功能是由 [数据万象](#)（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

说明

使用内容审核服务需拥有数据万象使用权限：

- 主账号请 [单击此处](#) 进行角色授权。
- 子账号请参见 [授权子账号接入数据万象服务](#) 文档。

本文档提供关于文本审核的 API 概览和 SDK 示例代码。

API	操作描述
提交文本审核任务	用于提交一个文本审核任务
查询文本审核任务结果	用于查询指定的文本审核任务

提交文本审核任务

功能说明

用于提交一个文本审核任务。

方法原型

```
CosResult CosAPI::CreateTextAuditingJob(const CreateTextAuditingJobReq& req,
CreateTextAuditingJobResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "examplebucket-1250000000";

CreateTextAuditingJobReq req(bucket_name);
CreateTextAuditingJobResp resp;

// 添加请求参数 参数详情请见 API 接口文档
// Content、Url 和 Object 不同时生效
// 当使用 Content 时为审核结果同步返回
```

```
// 使用 Object 或 Url 时为审核结果异步返回
req.SetContent(CodecUtil::Base64Encode("content"));
// req.SetObject(test.txt);
req.SetBizType("b81d45f94b91a683255e9a9506f45a11");

// 调用接口, 获取任务响应对象
CosResult result = cos.CreateTextAuditingJob(req, &resp);
if (result.IsSucc()) {
    // 创建审核任务成功, 可以调用 CreateTextAuditingJobResp 的成员函数
} else {
    // 创建审核任务失败, 可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	CreateTextAuditing 操作的请求	CreateTextAuditingJobReq	是
resp	CreateTextAuditing 操作的响应	CreateTextAuditingJobResp	是

CreateTextAuditingJobReq 提供以下成员函数:

```
// 设置执行操作的 bucket
void SetBucketName(const std::string& bucket_name);
// 设置 input 需要审核的视频
void SetInput(const AuditingInput& input);
// 设置审核配置规则
void SetConf(const Conf& conf);

// conf
// 表示审核策略的唯一标识, 您可以通过控制台上的审核策略页面, 配置您希望审核的场景, 如涉黄、广告、违法违规等, 配置指引: 设置公共审核策略。
// 您可以在控制台上获取到 BizType。BizType 填写时, 此条审核请求将按照该审核策略中配置的场景进行审核。
void SetBizType(const std::string& biz_type);
// 回调地址, 以 http://或者https://开头的地址。
void SetCallBack(const std::string& callback);
// 回调内容的结构, 有效值: Simple (回调内容包含基本信息)、Detail (回调内容包含详细信息)。默认为 Simple。
void SetCallBackVersion(const std::string& callback_version);

// input
// 当前 COS 存储桶中的文本文件名称, 例如在目录 test 中的文件 test.txt, 则文件名称为 test/test.txt, 文本文件仅支持 UTF8 编码和 GBK 编码的内容, 且文件大小不得超过1MB。
```

```
// 当使用 Object 时审核结果异步，需要使用查询文本审核任务接口查询
void SetObject(const std::string& object);
// 文本文件的完整链接，例如：https://www.test.com/test.txt。
// 当使用 Url 时审核结果异步，需要使用查询文本审核任务接口查询
void SetUrl(const std::string& url);
// 该字段在审核结果中会返回原始内容，长度限制为512字节。您可以使用该字段对待审核的数据进行唯一业务标识。
void SetDataId(const std::string& data_id);
// 用户业务字段。
void SetUserInfo(const UserInfo& user_info);
// 当传入的内容为纯文本信息，需要先经过 base64 编码，文本编码前的原文长度不能超过10000个 UTF8 编码字符。若超出长度限制，接口将会报错。
// 当使用 Content 时审核结果同步返回
void SetContent(const std::string& content) { m_input.SetContent(content); }
```

CreateTextAuditingJobResp 提供的成员函数如下：

```
// 获取API请求执行返回的任务详情
TextAuditingJobsDetail GetJobsDetail();
// 获取API请求的透传ID
std::string GetRequestId();
```

返回结果说明

- 成功：解析 API 返回的 XML 内容中的审核任务结果到 TextAuditingJobsDetail 结构中，具体返回参数可查看 [提交文本审核任务](#) 文档。
- 失败：发生错误（例如 Bucket 不存在），错误信息则解析在 CosResult 结构体中。详情请参见 [异常处理](#)。

查询文本审核任务结果

功能说明

用于查询文本审核任务执行情况和结果。

方法原型

```
CosResult DescribeTextAuditingJob(const DescribeTextAuditingJobReq& req,
DescribeTextAuditingJobResp* resp);
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
```

```
std::string bucket_name = "examplebucket-1250000000";

DescribeTextAuditingJobReq req(bucket_name);
DescribeTextAuditingJobResp resp;

// 添加请求参数 参数详情请见 API 接口文档
req.SetJobId("aab1ca9fc8a3ed11ea834c525400863904");

// 调用接口, 获取任务响应对象
CosResult result = cos.DescribeTextAuditingJob(req, &resp);
if (result.IsSuccess()) {
    // 查询审核任务成功, 可以调用 DescribeTextAuditingJobResp 的成员函数
} else {
    // 查询审核任务失败, 可以调用 CosResult 的成员函数输出错误信息
}
```

参数说明

参数	参数描述	类型	是否必填
req	DescribeTextAuditingJob 操作的请求	DescribeTextAuditingJobReq	是
resp	DescribeTextAuditingJob 操作的响应	DescribeTextAuditingJobResp	是

DescribeTextAuditingJobReq 提供以下成员函数:

```
// 设置执行操作的 bucket
void SetBucketName(const std::string& bucket_name);
// 设置查询的审核任务ID
void SetJobId(const std::string& job_id);
```

DescribeTextAuditingJobResp 提供的成员函数如下:

```
// 获取 API 请求执行返回的任务详情
TextAuditingJobsDetail GetJobsDetail();
// 获取 API 请求的透传 ID
std::string GetRequestId();
```

返回结果说明

- 成功: 解析 API 返回的 XML 内容中的审核任务结果到 TextAuditingJobsDetail 结构中, 具体返回参数可查看 [查询文本审核任务结果](#) 文档。
- 失败: 发生错误 (例如 Bucket 不存在), 错误信息则解析在 CosResult 结构体中。详情请参见 [异常处理](#)。

.NET(C#) SDK

快速入门

最近更新时间：2025-05-06 11:31:01

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());
}
```

文本审核

最近更新时间：2024-11-29 09:48:13

简介

内容审核功能是由 [数据万象](#)（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

说明

使用内容审核服务需拥有数据万象使用权限：

- 主账号请 [单击此处](#) 进行角色授权。
- 子账号请参见 [授权子账号接入数据万象服务](#) 文档。

本文档提供关于文本审核的 API 概览和 SDK 示例代码。

API	操作描述
提交文本审核任务	用于提交一个文本审核任务
查询文本审核任务结果	用于查询指定的文本审核任务

提交文本审核任务

功能说明

本接口用于提交一个文本审核任务。您可以通过提交文本审核任务审核您的文本文件，然后通过查询文本审核任务接口查询审核结果。

下面示例展示了如何提交一个文本审核任务，然后通过 JobId 查询文本审核任务结果。

说明

- 此操作需要存储桶开通数据万象相关功能。
- 文本审核接口从 5.4.24 版本开始支持，下载新版 SDK 前往 [Releases](#) 或参见 [快速入门](#)。
- 查看版本 Changelog 请前往 [GitHub](#)。

示例代码

```
[/]: # ".cssg-snippet-SubmitTextCensorJobModel"
```

```
using COSXML.Model.CI;
using COSXML.Auth;
using System;
using System.Threading;
using COSXML;
```

```
namespace COSSnippet
{
    public class SubmitTextCensorJobModel {

        private CosXml cosXml;

        SubmitTextCensorJobModel() {
            CosXmlConfig config = new CosXmlConfig.Builder()
                .SetRegion("COS_REGION") // 设置默认的地域, COS 地域的简称请参照
https://cloud.tencent.com/document/product/436/6224
                .Build();

            string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请
参照 https://console.cloud.tencent.com/cam/capi
            string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥
请参照 https://console.cloud.tencent.com/cam/capi
            long durationSecond = 600; // 每次请求签名有效时长, 单位为秒
            QCloudCredentialProvider qCloudCredentialProvider = new
DefaultQCloudCredentialProvider(secretId,
                secretKey, durationSecond);

            this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
        }

        /// 提交文本审核任务
        public string SubmitTextCensorJob()
        {
            // 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考
https://console.cloud.tencent.com/developer
            string bucket = "examplebucket-1250000000"; // 注意: 此操作需要 bucket 开
通内容审核相关功能
            SubmitTextCensorJobRequest request = new
SubmitTextCensorJobRequest(bucket);
            request.SetCensorObject("text.txt"); // 媒体文件的对象键, 需要替换成桶内存在
的媒体文件的对象键
            // 审核的场景类型, 有效值: Porn (涉黄)、Ads (广告), 可以传入多种类型, 不同类型以
逗号分隔, 例如: Porn,Ads
            request.SetDetectType("Porn,Ads");
            // 执行请求
            SubmitCensorJobResult result = cosXml.SubmitTextCensorJob(request);
            Console.WriteLine(result.GetResultInfo());
            Console.WriteLine(result.censorJobsResponse.JobsDetail.JobId);
            Console.WriteLine(result.censorJobsResponse.JobsDetail.State);
            Console.WriteLine(result.censorJobsResponse.JobsDetail.CreationTime);
            return result.censorJobsResponse.JobsDetail.JobId;
        }
    }
}
```

```
        // .cssg-snippet-body-end
    }

    static void Main(string[] args)
    {
        SubmitTextCensorJobModel m = new SubmitTextCensorJobModel();
        /// 提交审核任务
        string JobId = m.SubmitTextCensorJob();
        /// 打印唯一标识了此次审核任务的 JobId
        Console.WriteLine("JobId : " + JobId);
    }
}
}
```

ⓘ 说明

更多完整示例，请前往 [GitHub](#) 查看。

查询文本审核任务

功能说明

用于查询文本审核任务执行情况和结果。

示例代码

```
using COSXML.Model.CI;
using COSXML.Auth;
using System;
using System.Threading;
using COSXML;

namespace COSSnippet
{
    public class SubmitTextCensorJobModel {

        private CosXml cosXml;

        SubmitTextCensorJobModel() {
            CosXmlConfig config = new CosXmlConfig.Builder()
                .SetRegion("COS_REGION") // 设置默认的地域，COS 地域的简称请参照
                https://cloud.tencent.com/document/product/436/6224
                .Build();
        }
    }
}
```

```
string secretId = "SECRET_ID"; // 云 API 密钥 SecretId, 获取 API 密钥请
参照 https://console.cloud.tencent.com/cam/capi
string secretKey = "SECRET_KEY"; // 云 API 密钥 SecretKey, 获取 API 密钥
请参照 https://console.cloud.tencent.com/cam/capi
long durationSecond = 600; //每次请求签名有效时长, 单位为秒
QCloudCredentialProvider qCloudCredentialProvider = new
DefaultQCloudCredentialProvider(secretId,
    secretKey, durationSecond);

this.cosXml = new CosXmlServer(config, qCloudCredentialProvider);
}

/// 查询文本审核任务结果
public void GetTextCensorJobResult(string JobId)
{
    // 存储桶名称, 此处填入格式必须为 bucketname-APPID, 其中 APPID 获取参考
    https://console.cloud.tencent.com/developer
    string bucket = "examplebucket-1250000000"; // 注意: 此操作需要 bucket 开
    通内容审核相关功能
    GetTextCensorJobRequest request = new GetTextCensorJobRequest(bucket,
    JobId);
    // 执行请求
    GetTextCensorJobResult result = cosXml.GetTextCensorJob(request);
    Console.WriteLine(result.GetResultInfo());

    // 读取审核结果
    Console.WriteLine(result.resultStruct.JobsDetail.JobId);
    Console.WriteLine(result.resultStruct.JobsDetail.State);
    Console.WriteLine(result.resultStruct.JobsDetail.CreationTime);
    Console.WriteLine(result.resultStruct.JobsDetail.Object);
    Console.WriteLine(result.resultStruct.JobsDetail.SectionCount);
    Console.WriteLine(result.resultStruct.JobsDetail.Result);

    Console.WriteLine(result.resultStruct.JobsDetail.PornInfo.HitFlag);
    Console.WriteLine(result.resultStruct.JobsDetail.PornInfo.Count);
    Console.WriteLine(result.resultStruct.JobsDetail.AdsInfo.HitFlag);
    Console.WriteLine(result.resultStruct.JobsDetail.AdsInfo.Count);

    // 文本节选Section信息
    for(int i = 0; i < result.resultStruct.JobsDetail.Section.Count; i++)
    {

    Console.WriteLine(result.resultStruct.JobsDetail.Section[i].StartByte);

    Console.WriteLine(result.resultStruct.JobsDetail.Section[i].PornInfo.HitFlag);
```

```
Console.WriteLine(result.resultStruct.JobsDetail.Section[i].PornInfo.Score);

Console.WriteLine(result.resultStruct.JobsDetail.Section[i].PornInfo.Keywords)
;
    }
}

static void Main(string[] args)
{
    SubmitTextCensorJobModel m = new SubmitTextCensorJobModel();
    // 输入提交文本审核任务时获取到的 JobId
    string JobId = "xxx";
    // 查询审核任务结果
    m.GetTextCensorJobResult(JobId);
}
}
```

📌 说明

更多完整示例，请前往 [GitHub](#) 查看。

Go SDK

快速入门

最近更新时间：2024-11-29 09:48:13

下载与安装

相关资源

- 对象存储 COS 的 XML Go SDK 源码下载地址：[XML Go SDK](#)。
- 示例 Demo 下载地址：[COS XML Go SDK 示例](#)。
- 更多信息请参见 [COS Go SDK API 文档](#)。
- SDK 文档中的所有示例代码请参见 [SDK 代码示例](#)。
- SDK 更新日志请参见 [ChangeLog](#)。
- SDK 常见问题请参见：[Go SDK 常见问题](#)。

! 说明

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

环境依赖

Golang：用于下载和安装 Go 编译运行环境，请前往 [Golang 官网](#) 进行下载。

安装 SDK

执行以下命令安装 COS Go SDK：

```
go get -u github.com/tencentyun/cos-go-sdk-v5
```

开始使用

下面为您介绍如何使用 COS Go SDK 完成一个基础操作，如初始化客户端、创建存储桶、查询存储桶列表、上传对象、查询对象列表、下载对象和删除对象。

初始化

⚠ 注意

- 建议用户 [使用临时密钥](#) 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)
}
}
```

文本审核

最近更新时间：2024-11-29 09:48:13

简介

内容审核功能是由 [数据万象](#)（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

说明

使用内容审核服务需拥有数据万象使用权限：

- 主账号请 [单击此处](#) 进行角色授权。
- 子账号请参见 [授权子账号接入数据万象服务](#) 文档。

本文档提供关于文本审核的相关的 API 概览以及 SDK 示例代码。

API	操作描述
创建任务	用于提交一个文本审核任务
查询任务	用于查询指定的文本审核任务

创建任务

PutTextAuditingJob 接口用来提交一个文本审核任务。当审核文本内容时，响应体直接返回审核结果，当审核文本对象时，返回任务摘要，您可以通过主动设置回调地址接收审核信息，也可以通过 JobId 进行查询。

方法原型

```
func (s *CIService) PutTextAuditingJob(ctx context.Context, opt
*PutTextAuditingJobOptions) (*PutTextAuditingJobResult, *Response, error)
```

请求示例

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

```

    }
})
opt := &cos.PutTextAuditingJobOptions{
    InputObject: "test.txt",
    Conf: &cos.TextAuditingJobConf{
    },
}
res, _, err := c.CI.PutTextAuditingJob(context.Background(), opt)

```

参数说明

```

type PutTextAuditingJobOptions struct {
    InputObject    string
    InputUrl       string
    InputContent   string
    InputDataId   string
    InputUserInfo *UserExtraInfo
    Conf           *TextAuditingJobConf
}
type UserExtraInfo struct {
    TokenId  string
    Nickname string
    DeviceId string
    AppId    string
    Room     string
    IP       string
    Type     string
}
type TextAuditingJobConf struct {
    Callback          string
    CallbackVersion  string
    BizType           string
}

```

参数名称	参数描述	类型	是否必填
InputObject	当前 COS 存储桶中的文本文件名称，例如在目录 test 中的文件 test.txt，则文件名称为 test/test.txt，文本文件仅支持UTF8编码和 GBK 编码的内容，且文件大小不得超过1MB。Object，Url 和 Content 只能选择其中一种。	String	否
InputContent	当传入的内容为纯文本信息，需要先经过 base64 编码，文本编码前的原文长度不能超过10000个 utf8 编码字符。若超出长度限制，接口将会报错。Object，Url 和 Content 只能选择其中一种。	String	否

InputUrl	文本文件的链接地址，例如 <code>http://examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/test.txt</code> 。Object, Url 和 Content 只能选择其中一种。	String	否
InputDataId	文本标识，该字段在结果中返回原始内容，长度限制为512字节。	String	否
InputUserInfo	用户业务字段。	Object	否
Conf	审核规则配置。	Struct	是
BizType	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置审核策略 。	String	否
Callback	审核结果可以回调形式发送至您的回调地址，支持以 <code>http://</code> 或者 <code>https://</code> 开头的地址，例如： <code>http://www.callback.com</code> 。当 Input 使用 Content 时，该参数不生效，结果会直接返回。	String	否
CallbackVersion	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。	String	否

结果说明

调用 PutTextAuditingJob 函数，会解析 api 返回的 xml 内容到 PutTextAuditingJobResult 结构，具体返回参数可查看 [提交文本审核任务](#) 文档。

查询任务

GetTextAuditingJob 接口用来查询指定的文本审核任务。您可以根据文本审核任务的 JobId 来查询文本审核结果。

方法原型

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

请求示例

```
// 将 examplebucket-1250000000 和 COS_REGION 修改为真实的信息
// CI 任务需要提供 CIURL
bu, _ := url.Parse("https://examplebucket-1250000000.cos.COS_REGION.myqcloud.com")
cu, _ := url.Parse("https://examplebucket-1250000000.ci.COS_REGION.myqcloud.com")
b := &cos.BaseURL{BucketURL: bu, CIURL: cu}
```

```
c := cos.NewClient(b, &http.Client{
    Transport: &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
})
jobId := "stce25f391a72e11eb99f*****"
res, _, err := c.CI.GetTextAuditingJob(context.Background(), jobId)
```

参数说明

参数名称	参数描述	类型
jobId	任务 ID。	String

结果说明

调用 `GetTextAuditingJob` 函数，会解析 api 返回的 xml 内容到 `GetTextAuditingJobResult` 结构，具体返回参数可查看 [查询文本审核任务结果](#) 文档。

COS iOS SDK

快速入门

最近更新时间：2024-11-29 09:48:13

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](#)。

① 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：

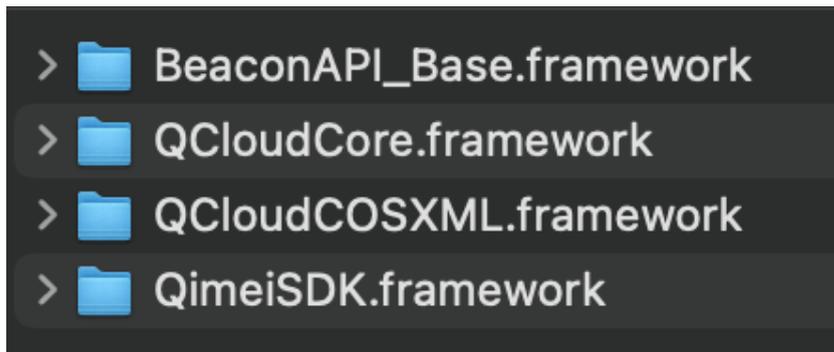
```
pod 'QCloudCOSXML/Transfer'
```

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
```



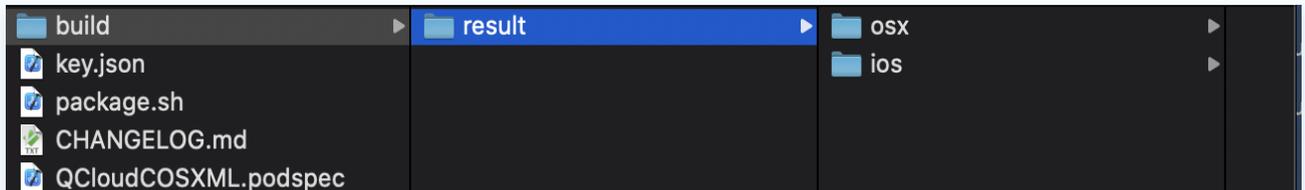
! 说明

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*) fields
    request:(QCloudBizHTTPRequest*) request
    urlRequest:(NSMutableURLRequest*) urlRequest
```

```
compelete:
(QCloudHTTPAuthenticationContinueBlock)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

    QCloudAuthenticationV5Creator* creator = [[QCloudAuthenticationV5Creator
    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 fields: QCloudSignatureFields!,
              request: QCloudBizHTTPRequest!,
              urlRequest urlRequest: NSMutableURLRequest!,
              complete continueBlock: QCloudHTTPAuthenticationContinueBlock!) {

    //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 = QCloudAuthenticationV5Creator.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 `QCloudCredentialFenceQueue` 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`
- `QCloudCredentialFenceQueueDelegate`

完整示例代码请参考如下：

Objective-c

```
//AppDelegate.m
//AppDelegate 需遵循 QCloudSignatureProvider 与
```

```
//QCloudCredentialFenceQueueDelegate 协议

@interface AppDelegate()<QCloudSignatureProvider,
QCloudCredentialFenceQueueDelegate>

// 一个脚手架实例
@property (nonatomic) QCloudCredentialFenceQueue* 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 = [QCloudCredentialFenceQueue new];
    self.credentialFenceQueue.delegate = self;

    return YES;
}

- (void) fenceQueue:(QCloudCredentialFenceQueue *)queue
requestCreatorWithContinue:(QCloudCredentialFenceQueueContinue) 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

QCloudAuthenticationV5Creator* creator = [[QCloudAuthenticationV5Creator 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*) fields
                    request:(QCloudBizHTTPRequest*) request
                    urlRequest:(NSMutableURLRequest*) urlRequest
                    complete:(QCloudHTTPAuthenticationContinueBlock) continueBlock
{
    [self.credentialFenceQueue performAction:^(QCloudAuthenticationCreator *creator,
        NSError *error) {
        if (error) {
            continueBlock(nil, error);
        } else {
            // 注意 这里不要对urlRequest 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
//QCloudCredentialFenceQueueDelegate 协议

class AppDelegate: UIResponder, UIApplicationDelegate,
    QCloudSignatureProvider, QCloudCredentialFenceQueueDelegate {

    var credentialFenceQueue:QCloudCredentialFenceQueue?;

    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 = QCloudCredentialFenceQueue.init();
    }
}
```

```
self.credentialFenceQueue?.delegate = self;

return true
}

func fenceQueue(_ queue: QCloudCredentialFenceQueue!,
requestCreatorWithContinue continueBlock:
QCloudCredentialFenceQueueContinue!) {
//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 = QCloudAuthenticationV5Creator.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 urlRequist: NSMutableURLRequest!,
complete continueBlock: QCloudHTTPAuthenticationContinueBlock!) {
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*) fields
    request:(QCloudBizHTTPRequest*) request
    urlRequst:(NSMutableURLRequest*) urlRequst
    compelete:
(QCloudHTTPAuthenticationContinueBlock) 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
QCloudAuthenticationV5Creator* creator = [[QCloudAuthenticationV5Creator
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 fields: QCloudSignatureFields!,
              request: QCloudBizHTTPRequest!,
              urlRequest urlRequst: NSMutableURLRequest!,
              compelete continueBlock: QCloudHTTPAuthenticationContinueBlock!)
{
    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 = QCloudAuthenticationV5Creator.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 `QCloudCredentialFenceQueueDelegate` 协议。

Objective-C

```
- (void) signatureWithFields:(QCloudSignatureFields*) fields
                    request:(QCloudBizHTTPRequest*) request
                    urlRequest:(NSMutableURLRequest*) urlRequst
                    compelete:
(QCloudHTTPAuthenticationContinueBlock) 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 fields: QCloudSignatureFields!,
               request: QCloudBizHTTPRequest!,
               urlRequest urlRequest: NSMutableURLRequest!,
               complete continueBlock: QCloudHTTPAuthenticationContinueBlock!)
{
    // 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 URLWithString:@"Local File Path"];

//监听下载结果
[request setFinishBlock:^(id responseObject, NSError *error) {
    //responseObject 包含所有的响应 http 头部
    NSDictionary* info = (NSDictionary *) responseObject;
}];

//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 defaultManager]
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(request);
```

! 说明

更多完整示例，请前往 [GitHub](#) 查看。

快速体验

最近更新时间：2024-11-29 09:48:13

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：2024-11-29 09:48:13

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 responseObject, NSError* error) {  
    // 可以从 responseObject 中获取服务器返回的 header 信息  
    NSDictionary* info = (NSDictionary *) responseObject;  
}];  
[[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 responseObject, NSError* error) {
    // 可以从 responseObject 中获取服务器返回的 header 信息
    NSDictionary * result = (NSDictionary *)responseObject;
}];

[[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，请确保持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 responseObject, NSError*error) {
    // 可以从 responseObject 中获取服务器返回的 header 信息
    NSDictionary* info = (NSDictionary *) responseObject;
}];

[[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。

文本审核

最近更新时间：2024-11-29 09:48:13

简介

内容审核功能是由 [数据万象](#)（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

说明

使用内容审核服务需拥有数据万象使用权限：

- 主账号请 [单击此处](#) 进行角色授权。
- 子账号请参见 [授权子账号接入数据万象服务](#) 文档。

本文档提供关于文本审核的相关的 API 概览以及 SDK 示例代码。

API	操作描述
提交文本审核任务	用于提交一个文本审核任务
查询文本审核任务结果	用于查询指定的文本审核任务
文本审核结果反馈	<ul style="list-style-type: none">您可通过本接口反馈与预期不符的审核结果，例如色情文本被审核判定为正常或正常文本被判定为色情时可通过该接口直接反馈本接口不会直接修改审核结果，您反馈的错误审核结果将在后台进行确认，并在后续的审核任务中生效

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 [SDK API](#)。

创建任务

功能说明

QCloudPostTextRecognitionRequest 接口用于提交一个文本审核任务。您可以通过主动设置回调地址接收审核信息，也可以通过 JobId 进行查询。

注意

COS iOS SDK 版本需要大于等于 v6.0.9。

请求示例

Objective-C

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

// content:纯文本信息
// object: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"
// url:文本文件的完整链接
// 单次请求只能使用 Object、Content、Url 中的一个。
// 当选择 Object、Url 时,审核结果为异步返回,可通过 查询文本审核任务结果 API 接口获取返回
结果。
// 当选择 Content 时,审核结果为同步返回,可通过 响应体 查看返回结果。
request.content = @"文本内容";

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

// 文件所在地域
request.regionName = @"regionName";

// 审核策略,不带审核策略时使用默认策略。具体查看
https://cloud.tencent.com/document/product/460/56345
request.bizType = @"bizType";

request.finishBlock = ^(QCloudPostTextRecognitionResult * outputObject,
NSError *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
    // QCloudPostTextRecognitionResult 类;
};

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

❗ 说明

更多完整示例,请前往 [GitHub](#) 查看。

Swift

```
let request = QCloudPostTextRecognitionRequest();

// content:纯文本信息
// object: COS 存储桶中的文本文件名称
// url:文本文件的完整链接
// 单次请求只能使用 Object、Content、Url 中的一个。
// 当选择 Object、Url 时,审核结果为异步返回,可通过 查询文本审核任务结果 API 接口获取返回
结果。
// 当选择 Content 时,审核结果为同步返回,可通过 响应体 查看返回结果。
```

```
request.content = "文本内容";

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

// 审核策略, 不带审核策略时使用默认策略。具体查看
https://cloud.tencent.com/document/product/460/56345
request.bizType = "bizType";

// 文件所在地域
request.regionName = "regionName";

// 审核策略, 不带审核策略时使用默认策略。具体查看
https://cloud.tencent.com/document/product/460/56345
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
    // QCloudPostTextRecognitionResult 类;
}
QCloudCOSXMLService.defaultCOSXML().postTextRecognition(request);
```

❗ 说明

更多完整示例, 请前往 [GitHub](#) 查看。

查询任务

功能说明

`QCloudGetTextRecognitionRequest` 接口用于查询指定的文本审核任务。您可以根据文本审核任务的 `JobId` 来查询文本审核结果。

⚠ 注意

COS iOS SDK 版本需要大于等于 v6.0.9。

请求示例

Objective-C

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

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

// 文件所在地域
```

```
request.regionName = @"regionName";

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

request.finishBlock = ^(QCloudTextRecognitionResult * 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
    // QCloudTextRecognitionResult 类;
};

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

📌 说明

更多完整示例, 请前往 [GitHub](#) 查看。

Swift

```
let request = QCloudGetTextRecognitionRequest();

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

// 文件所在地域
request.regionName = "regionName";

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

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
    // QCloudTextRecognitionResult 类;
};

QCloudCOSXMLService.defaultCOSXML().getTextRecognition(request);
```

📌 说明

更多完整示例, 请前往 [GitHub](#) 查看。

文本审核结果反馈

功能说明

您可通过本接口反馈与预期不符的审核结果，例如色情文本被审核判定为正常或正文本被判定为色情时可通过该接口直接反馈。

⚠ 注意:

COS iOS SDK 版本需要大于等于 v6.2.5。

示例代码

Objective-C

```
QCloudPostTextAuditReportRequest * request = [QCloudPostTextAuditReportRequest
new];
// 存储桶名称, 格式为 BucketName-APPID
request.bucket = @"examplebucket-1250000000";
// 文件所在地域
request.regionName = @"regionName";
QCloudPostTextAuditReport * input = [QCloudPostTextAuditReport new];
input.ContentType = 1;
input.Label = @"Label";
input.SuggestedLabel = @"Normal";
request.input = input;
[request setFinishBlock:^(QCloudPostTextAuditReportResult * _Nullable result,
NSError * _Nullable error) {
    /// result 文本审核结果反馈, 详细字段请查看 API 文档或者 SDK 源码
}];
[[QCloudCOSXMLService defaultCOSXML] PostTextAuditReport:request];
```

📌 说明:

更多完整示例, 请前往 [GitHub](#) 查看。

Swift

```
let request = QCloudPostTextAuditReportRequest()
// 存储桶名称, 格式为 BucketName-APPID
request.bucket = "examplebucket-1250000000"
// 文件所在地域
request.regionName = "regionName"
let input = QCloudPostTextAuditReport()
input.contentType = 1
input.label = "Label"
input.suggestedLabel = "Normal"
request.input = input
request.finishBlock = { result, error in
    /// result 文本审核结果反馈, 详细字段请查看 API 文档或者 SDK 源码
```

```
}  
QCloudCOSXMLService.defaultCOSXML().postTextAuditReport(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：2024-11-29 09:48:13

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();
```

文本审核

最近更新时间：2025-02-28 20:43:42

简介

内容审核功能是由 [数据万象](#)（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

说明

使用内容审核服务需拥有数据万象使用权限：

- 主账号请 [单击此处](#) 进行角色授权。
- 子账号请参见 [授权子账号接入数据万象服务](#) 文档。

本文档提供关于文本审核的 API 概览和 SDK 示例代码。

API	操作描述
提交文本审核任务	用于提交一个文本审核任务。
查询文本审核任务结果	用于查询指定的文本审核任务。

提交文本审核任务

注意

- 不可同时输入 Object 和 Content。
- 当选择 Input 资源为 Object 或 URL 时，审核结果为异步返回，可通过 [查询文本审核任务结果](#) API 接口获取返回结果。
- 当选择 Input 资源为 Content 时，审核结果为同步返回，可通过直接通过请求的响应体 TextAuditingResponse 查看审核结果。
- 目前仅支持中文、英文、阿拉伯数字的检测与审核。

功能说明

用于提交一个文本审核任务。

方法原型

```
TextAuditingResponse createAuditingTextJobs(TextAuditingRequest request);
```

请求示例

```
//1. 创建任务请求对象
```

```
TextAuditingRequest request = new TextAuditingRequest ();
//2.添加请求参数 参数详情请见 API 接口文档
request.setBucketName ("examplebucket-12500000008");
//2.1.1设置请求内容, 文本内容的Base64编码
request.getInput ().setContent ("Base64Str");
//或是cos中的设置对象地址 不可同时设置
//request.getInput ().setObject ("1.txt");
//2.2设置审核模板 (可选)
//request.getConf ().setBizType ("aa3e9d84a6a079556b0109a935c*****");
//3.调用接口, 获取任务响应对象
TextAuditingResponse response = client.createAuditingTextJobs (request);
```

参数说明

Request 中的具体数据描述如下:

节点名称 (关键字)	父节点	描述	类型	是否必选
Request	无	文本审核的具体配置项。	Container	是

Container 类型 Request 的具体数据描述如下:

节点名称 (关键字)	父节点	描述	类型	是否必选
BucketName	Request	COS 的存储桶名称。	String	是
Input	Request	需要审核的内容。	Container	是
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下:

节点名称 (关键字)	父节点	描述	类型	是否必选
Object	Request.Input	当前 COS 存储桶中的文本文件名称, 例如在目录 test 中的文件 1.txt, 则文件名称为 test/1.txt, 文本文件仅支持 UTF8 编码和 GBK 编码的内容, 且文件大小不得超过 1MB。	String	否
Content	Request.Input	当传入的内容为纯文本信息, 需要先经过 base64 编码, 文本编码前的原文长度不能超过10000个 utf8 编码字符。若超出长度限制, 接口将会报错。	String	否

Container 类型 Conf 的具体数据描述如下:

节点名称（关键字）	父节点	描述	类型	是否必选
BizType	Request.Conf	审核策略，不填写则使用默认策略。可在控制台进行配置，详情请参见 设置公共审核策略 。	String	否
Callback	Request.Conf	审核结果可以回调形式发送至您的回调地址，支持以 <code>http://</code> 或者 <code>https://</code> 开头的地址，例如： <code>http://www.callback.com</code> 。	String	否

返回结果说明

- 成功：成功则返回 `TextAuditingResponse` 文本审核任务结果对象。具体响应参数同 [文本内容同步审核的 Response](#)。
- 失败：发生错误（例如 Bucket 不存在），抛出异常 `CosClientException` 或者 `CosServiceException`。详情请参见 [异常处理](#)。

查询文本审核任务

功能说明

用于查询文本审核任务执行情况和结果。

方法原型

```
TextAuditingResponse describeAuditingTextJob(TextAuditingRequest request);
```

请求示例

```
//1. 创建任务请求对象
TextAuditingRequest request = new TextAuditingRequest();
//2. 添加请求参数 参数详情请见 API 接口文档
request.setBucketName("examplebucket-1250000000");
request.setJobId("st68d08596f35011eb9324525400*****");
//3. 调用接口, 获取任务响应对象
TextAuditingResponse response = client.describeAuditingTextJob(request);
```

参数说明

参数名称	描述	类型	是否必选
bucketName	Bucket 的命名规则为 BucketName-APPID，详情请参见 存储桶概述 。	String	是

jobId	需要查询的任务 ID。	String	是
-------	-------------	--------	---

返回结果说明

- 成功：返回 TextAuditingResponse 审核任务结果对象。具体响应参数同 [查询文本审核任务的 Response](#)。
- 失败：发生错误（例如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。详情请参见 [异常处理](#)。

异常处理

最近更新时间：2024-11-29 09:48:13

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 RuntimeException。

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
traceld	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。

COS 自助诊断工具	<> 点击自助诊断
输入 RequestId 进行智能诊断，获取请求基本信息、帮助指引和诊断提示，快速定位请求错误。	

3. 在顶部的 RequestId 输入框中，输入待诊断的 RequestId，并单击[开始诊断](#)，请您耐心等待几分钟，便能看到相应的智能诊断结果。

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 19:14:32

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。

添加跨域访问CORS规则

来源 Origin *

http://a.qcloud.com
https://b.qcloud.com

域名以 http://或 https:// 开头，每行一个，一行最多一个通配符 *

操作 Methods *

PUT GET POST DELETE HEAD

Allow-Headers

*

在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，例如：x-cos-meta-md5

Expose-Headers

ETag
Content-Length
x-cos-request-id

Expose-Header 里返回的是 COS 的常用Header

超时 Max-Age *

5 s

OPTIONS 请求得到结果的有效期，需为正整数

返回 Vary: Origin

设置是否返回 Vary: Origin Header。如果浏览器同时存在 CORS 和非 CORS 请求，请启用该选项否则会出现跨域问题。勾选 Vary: Origin 后可能会造成浏览器访问或者 CDN 回源增加

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  
        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);
});
```

文本审核

最近更新时间：2024-11-29 09:48:13

简介

内容审核功能是由 [数据万象](#)（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

说明：

使用内容审核服务需拥有数据万象使用权限：

- 主账号请 [单击此处](#) 进行角色授权。
- 子账号请参见 [授权子账号接入数据万象服务](#) 文档。

本文档提供关于文本审核的 API 概览和 SDK 示例代码。

注意：

COS Javascript SDK 版本需要大于等于 v1.3.1。

API	操作描述
文本内容审核	用于进行一次文本内容审核
提交文本文件审核任务	用于提交一个文本文件审核任务
查询文本审核任务结果	用于查询指定的文本审核任务

文本内容审核

功能说明

用于进行一次文本内容审核，会同步返回审核结果。

请求示例

```
// sdk的引入及初始化cos请参考
https://cloud.tencent.com/document/product/436/11459
function postTextContentAuditing() {
  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，比如ap-beijing，必须
  };
  const host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
  const key = 'text/auditing'; // 固定值，必须
```

```
const url = `https://${host}/${key}`;
const body = COS.util.json2xml({
  Request: {
    Input: {
      // 使用 COS.util.encodeBase64 方法需要sdk版本至少为1.4.19
      Content: COS.util.encodeBase64('hello'), /* 需要审核的文本内容 */
    },
    Conf: {
      BizType: '',
    }
  }
});
cos.request({
  Method: 'POST', // 固定值, 必须
  Url: url, // 请求的url, 必须
  Key: key, // 固定值, 必须
  ContentType: 'application/xml', // 固定值, 必须
  Body: body // 请求体参数, 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);
  }
});
}
postTextContentAuditing();
```

参数说明

cos.request 方法参数说明:

参数名称	描述	类型	是否必选
Method	请求方法, 固定值	String	是
Url	请求url, 固定值	String	是
Key	固定值	String	是
ContentType	固定值	String	是
Body	请求体详情请参见 文本内容审核	Container	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 文本内容审核	Object

相关链接

[API 文档](#)

[功能指南](#)

提交文本文件审核任务

功能说明

用于提交一个文本文件内容审核任务。

请求示例

```
// sdk的引入及初始化cos请参考
https://cloud.tencent.com/document/product/436/11459
function postTextAuditing() {
  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，比如ap-beijing，必须
  };
  const host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
```

```
const key = 'text/auditing'; // 固定值, 必须
const url = `https://${host}/${key}`;
const body = COS.util.json2xml({
  Request: {
    Input: {
      // Url与Object二选一传入
      Object: 'test.txt', // 文件存储在cos的路径, 支持传入中文, 比如: 中文.txt
      // Url: `www.test.com/${encodeURIComponent('中文.txt')}`, // 文本文件url
      // 注意中文路径需要encode
    },
    Conf: {
      BizType: '',
    }
  }
});
cos.request({
  Method: 'POST', // 固定值, 必须
  Url: url, // 请求的url, 必须
  Key: key, // 固定值, 必须
  ContentType: 'application/xml', // 固定值, 必须
  Body: body // 请求体参数, 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);
  }
});
}
postTextAuditing();
```

参数说明

cos.request 方法参数说明:

参数名称	描述	类型	是否必选
Method	请求方法, 固定值	String	是
Url	请求url, 固定值	String	是
Key	固定值	String	是
ContentType	固定值	String	是

Body	请求体详情请参见 文本文件审核	Container	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 文本文件审核	Object

相关链接

[API 文档](#)

[功能指南](#)

查询文本审核任务

功能说明

用于查询文本审核任务执行情况和结果。

请求示例

```
// sdk的引入及初始化cos请参考  
https://cloud.tencent.com/document/product/436/11459  
function getTextAuditingResult () {  
  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, 比如ap-beijing, 必须
};
const jobId = 'st8d88c664aff511ecb23352540078cxxx'; // jobId 可以通过提交文本审核任务返回 (Input传入Object)
const host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
const key = `text/auditing/${jobId}`; // 固定值, 必须
const url = `https://${host}/${key}` + jobId;
cos.request({
  Method: 'GET', // 固定值, 必须
  Url: url, // 请求的url, 必须
  Key: key, // 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);
  }
});
}
getTextAuditingResult();

```

参数说明

cos.request 方法参数说明:

参数名称	描述	类型	是否必选
Method	请求方法, 固定值	String	是
Url	请求url, 固定值	String	是
Key	固定值: text/auditing/ 要查询的jobId	String	是

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID	String	是

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 查询文本审核任务	Object

相关链接

[API 文档](#)

[功能指南](#)

Node.js SDK

快速入门

最近更新时间：2025-03-14 19:14:32

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 <code>https:</code> 、 <code>http:</code> ，默认判断当前页面是 <code>http:</code> 时使用 <code>http:</code> ，否则使用 <code>https:</code>	String	否

ServiceDomain	调用 getService 方法时, 请求的域名, 例如 <code>service.cos.myqcloud.com</code>	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 <code>"{Bucket}.cos.{Region}.myqcloud.com"</code> , 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: <code>http://127.0.0.1:8080</code>	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);
});
```

文本审核

最近更新时间：2024-11-29 09:48:13

简介

内容审核功能是由 [数据万象](#)（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

说明：

使用内容审核服务需拥有数据万象使用权限：

- 主账号请 [单击此处](#) 进行角色授权。
- 子账号请参见 [授权子账号接入数据万象服务](#) 文档。

本文档提供关于文本审核的 API 概览和 SDK 示例代码。

注意：

COS Node.js SDK 版本需要大于等于 v2.11.2。

API	操作描述
文本内容审核	用于进行一次文本内容审核
提交文本文件审核任务	用于提交一个文本文件审核任务
查询文本审核任务结果	用于查询指定的文本审核任务

文本内容审核

功能说明

用于进行一次文本内容审核，会同步返回审核结果。

请求示例

```
// sdk的引入及初始化cos请参考
https://cloud.tencent.com/document/product/436/8629
function postTextContentAuditing() {
  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，比如ap-beijing，必须
  };
  const host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
  const key = 'text/auditing'; // 固定值，必须
```

```
const url = `https://${host}/${key}`;
const body = COS.util.json2xml({
  Request: {
    Input: {
      // 使用 COS.util.encodeBase64 方法需要sdk版本至少为1.4.19
      Content: COS.util.encodeBase64('hello'), /* 需要审核的文本内容 */
    },
    Conf: {
      BizType: '',
    }
  }
});
cos.request({
  Method: 'POST', // 固定值, 必须
  Url: url, // 请求的url, 必须
  Key: key, // 固定值, 必须
  ContentType: 'application/xml', // 固定值, 必须
  Body: body // 请求体参数, 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);
  }
});
}
postTextContentAuditing();
```

参数说明

cos.request 方法参数说明:

参数名称	描述	类型	是否必选
Method	请求方法, 固定值	String	是
Url	请求 url, 固定值	String	是
Key	固定值	String	是
ContentType	固定值	String	是
Body	请求体详情请参见 文本内容审核	Container	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 文本内容审核	Object

相关链接

[API 文档](#)

[功能指南](#)

提交文本文件审核任务

功能说明

用于提交一个文本文件内容审核任务。

请求示例

```
// sdk的引入及初始化cos请参考
https://cloud.tencent.com/document/product/436/8629
function postTextAuditing() {
  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，比如ap-beijing，必须
  };
  const host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
```

```
const key = 'text/auditing'; // 固定值, 必须
const url = `https://${host}/${key}`;
const body = COS.util.json2xml({
  Request: {
    Input: {
      // Url与Object二选一传入
      Object: 'test.txt', // 文件存储在cos的路径, 支持传入中文, 比如: 中文.txt
      // Url: `www.test.com/${encodeURIComponent('中文.txt')}`, // 文本文件url
      // 注意中文路径需要encode
    },
    Conf: {
      BizType: '',
    }
  }
});
cos.request({
  Method: 'POST', // 固定值, 必须
  Url: url, // 请求的url, 必须
  Key: key, // 固定值, 必须
  ContentType: 'application/xml', // 固定值, 必须
  Body: body // 请求体参数, 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);
  }
});
}
postTextAuditing();
```

参数说明

cos.request 方法参数说明:

参数名称	描述	类型	是否必选
Method	请求方法, 固定值	String	是
Url	请求 url, 固定值	String	是
Key	固定值	String	是
ContentType	固定值	String	是

Body	请求体详情请参见 文本文件审核	Container	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 文本文件审核	Object

相关链接

[API 文档](#)

[功能指南](#)

查询文本审核任务

功能说明

用于查询文本审核任务执行情况和结果。

请求示例

```
// sdk的引入及初始化cos请参考  
https://cloud.tencent.com/document/product/436/8629  
function getTextAuditingResult () {  
  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, 比如ap-beijing, 必须
  };
  const jobId = 'st8d88c664aff511ecb23352540078cxxx'; // jobId 可以通过提交文本审核任务返回 (Input传入Object)
  const host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
  const key = `text/auditing/${jobId}`; // 固定值, 必须
  const url = `https://${host}/${key}` + jobId;
  cos.request({
    Method: 'GET', // 固定值, 必须
    Url: url, // 请求的url, 必须
    Key: key, // 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);
    }
  });
}
getTextAuditingResult();

```

参数说明

cos.request 方法参数说明:

参数名称	描述	类型	是否必选
Method	请求方法, 固定值	String	是
Url	请求 url, 固定值	String	是
Key	固定值: text/auditing/要查询的jobId	String	是

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID	String	是

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 查询文本审核任务	Object

相关链接

[API 文档](#)

[功能指南](#)

PHP SDK

快速入门

最近更新时间：2025-06-10 10:11:11

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。

① 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);
}
```

文本审核

最近更新时间：2025-06-10 10:11:11

简介

本文档提供关于文本审核的 API 概览和 SDK 示例代码。

⚠ 注意：

需要 COS PHP SDK v2.4.4 及以上版本。旧版本可能存在 bug，使用时建议升级到 [最新版本](#)。

API	操作描述
提交文本审核任务	用于提交一个文本审核任务
查询文本审核任务结果	用于查询指定的文本审核任务

提交文本审核任务

功能说明

用于提交一个文本审核任务。

方法原型

```
public Guzzle\Service\Resource\Model detectText(array $args = array());
```

请求示例

示例一：文本内容审核

```
<?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', // 审核时必须为https
```

```
'credentials'=> array(
    'secretId' => $secretId ,
    'secretKey' => $secretKey));

try {
    $content = '敏感词';
    $result = $cosClient->detectText(array(
        'Bucket' => 'examplebucket-1250000000', //存储桶名称, 由 BucketName-
Appid 组成, 可以在 COS 控制台查看 https://console.cloud.tencent.com/cos5/bucket
        'Input' => array(
            'Content' => base64_encode($content), // 文本需 base64_encode
//            'DataId' => '', // 选填, 该字段在审核结果中会返回原始内容, 长度限制为512
字节。您可以使用该字段对待审核的数据进行唯一业务标识。
        ),
//        'Conf' => array(
//            'BizType' => '',
//        ), // 非必选, 在BizType 不传的情况下, 走默认策略及默认审核场景。
    ));
    // 请求成功
    print_r($result);
} catch (\Exception $e) {
    // 请求失败
    echo($e);
}
```

示例二：存储桶文本文件审核

```
<?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', // 审核时必须为 https
        'credentials'=> array(
            'secretId' => $secretId ,
            'secretKey' => $secretKey));
    ));

try {
```

```
$result = $cosClient->detectText(array(
    'Bucket' => 'examplebucket-1250000000', //存储桶名称, 由 BucketName-
Appid 组成, 可以在 COS 控制台查看 https://console.cloud.tencent.com/cos5/bucket
    'Input' => array(
        'Object' => 'test01.txt'
//          'DataId' => '', // 选填, 该字段在审核结果中会返回原始内容, 长度限制为512
字节。您可以使用该字段对待审核的数据进行唯一业务标识。
    ),
//      'Conf' => array(
//          'BizType' => '',
//          'Callback' => '', // 回调URL 选填
//          'CallbackVersion' => 'Detail', // 选填 Detail、Simple 默认为
Simple
//          'Freeze' => array(
//              'PornScore' => 90,
//              'AdsScore' => 90,
//              'IllegalScore' => 90,
//              'AbuseScore' => 90,
//              'PoliticsScore' => 90,
//              'TerrorismScore' => 90,
//          ), // 可通过该字段, 设置根据审核结果给出的不同分值, 对文本文件进行自动冻结,
仅当 input 中审核的文本为 object 时有效。
//      ), // 非必选, 在 BizType 不传的情况下, 走默认策略及默认审核场景。
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
    // 请求失败
    echo($e);
}
```

示例三：文本文件URL审核

```
<?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', // 审核时必须为https
'credentials'=> array(
    'secretId' => $secretId ,
    'secretKey' => $secretKey));

try {
    $result = $cosClient->detectText(array(
        'Bucket' => 'examplebucket-1250000000', //存储桶名称, 由 BucketName-
Appid 组成, 可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
        'Input' => array(
            'Url' => 'http://example.com/test.txt'
//            'DataId' => '', // 选填 该字段在审核结果中会返回原始内容, 长度限制为512
//            字节。您可以使用该字段对待审核的数据进行唯一业务标识。
        ),
//        'Conf' => array(
//            'BizType' => '',
//            'Callback' => '', // 选填 回调URL
//            'CallbackVersion' => 'Detail', // 选填 Detail、Simple 默认为
Simple
//        ), // 非必选, 在BizType不传的情况下, 走默认策略及默认审核场景。
    ));
// 请求成功
print_r($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
Input	Request	需要审核的内容。	Container	是
Conf	Request	审核规则配置。	Container	是

Container 类型 Input 的具体数据描述如下:

节点名称 (关键字)	父节点	描述	类型	是否必选
Object	Request	当前 COS 存储桶中的文本文件名称, 例如在目录 test 中的文件 test.txt, 则文件名称为 test/test.txt, 文本文件仅支持 UTF8 编码和 GBK 编码的内容, 且文件大小不得超过1MB。	String	否

	.Input			
Url	Request .Input	文本文件的完整链接，例如： <code>https://www.test.com/test.txt</code> 。	String	否
DataId	Request .Input	该字段在审核结果中会返回原始内容，长度限制为512字节。您可以使用该字段对待审核的数据进行唯一业务标识。	String	否
UserIn fo	Request .Input	用户业务字段。	Con tai ner	否

Container 节点 UserInfo 的内容：

节点名称（关键字）	描述	类型	是否必选
TokenId	一般用于表示账号信息，长度不超过128字节。	String	否
Nickname	一般用于表示昵称信息，长度不超过128字节。	String	否
DeviceId	一般用于表示设备信息，长度不超过128字节。	String	否
AppId	一般用于表示 App 的唯一标识，长度不超过128字节。	String	否
Room	一般用于表示房间号信息，长度不超过128字节。	String	否
IP	一般用于表示 IP 地址信息，长度不超过128字节。	String	否
Type	一般用于表示业务类型，长度不超过128字节。	String	否
ReceiveTokenId	一般用于表示接收消息的用户账号，长度不超过128字节。	String	否
Gender	一般用于表示性别信息，长度不超过128字节。	String	否

Level	一般用于表示等级信息，长度不超过128字节。	String	否
Role	一般用于表示角色信息，长度不超过128字节。	String	否

Container 类型 Conf 的具体数据描述如下：

节点名称 (关键字)	父节点	描述	类型	是否必选
BizType	Request.Conf	表示审核策略的唯一标识，您可以通过控制台上的审核策略页面，配置您希望审核的场景，如色情、广告、违法等，配置指引： 设置公共审核策略 。您可以在控制台上获取到 BizType。BizType 填写时，此条审核请求将按照该审核策略中配置的场景进行审核。BizType 不填写时，将自动使用默认的审核策略。	String	否
Callback	Request.Conf	审核结果可以回调形式发送至您的回调地址，支持以 <code>http://</code> 或者 <code>https://</code> 开头的地址，例如： <code>http://www.callback.com</code> 。当 Input 使用 Content 时，该参数不生效，结果会直接返回。	String	否
Callback Version	Request.Conf	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。	String	否
Callback Type	Request.Conf	回调片段类型，有效值：1（回调全部文本片段）、2（回调违规文本片段）。默认为 1。	Integer	否
Freeze	Request.Conf	可通过该字段，设置根据审核结果给出的不同分值，对文本文件进行自动冻结，仅当 <code>input</code> 中审核的文本为 <code>object</code> 时有效。	Container	否

Container 类型 Freeze 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
PornScore	Request.Conf.Freeze	取值为[0,100]，表示当色情审核结果大于或等于该分数时，自动进行冻结操作。不填写则表示不自动冻结，默认值为空。	Integer	否
AdScore	Request.Conf.Freeze	取值为[0,100]，表示当广告审核结果大于或等于该分数时，自动进行冻结操作。不填写则表示不自动冻结，默认值为空。	Integer	否

IllegalScore	Request.Config.Freeze	取值为[0,100]，表示当违法审核结果大于或等于该分数时，自动进行冻结操作。不填写则表示不自动冻结，默认值为空。	Integer	否
AbuseScore	Request.Config.Freeze	取值为[0,100]，表示当谩骂审核结果大于或等于该分数时，自动进行冻结操作。不填写则表示不自动冻结，默认值为空。	Integer	否

返回结果示例

响应参数具体含义参考 [文本审核API文档](#)。

示例一：文本内容审核

//当选择 Content 时，审核结果为同步返回，可通过 响应体 查看返回结果。

```
GuzzleHttp\Command\Result Object
(
    [RequestId] => NjEzYjUxMDhfZmNjYTNiMGFfNGM2NF8zNWY4Njg=
    [ContentType] => application/xml
    [ContentLength] => 878
    [JobsDetail] => Array
        (
            [Code] => Success
            [Message] => Array
                (
                )
            [JobId] => st8ffc0551123311eca33f525400f3e40d
            [State] => Success
            [CreationTime] => 2021-09-10T20:35:20+08:00
            [Content] => 5pWP5oSf6K+N
            [Result] => 0
            [SectionCount] => 1
            [PornInfo] => Array
                (
                    [HitFlag] => 0
                    [Count] => 0
                )
            [AdsInfo] => Array
                (
                    [HitFlag] => 0
                    [Count] => 0
                )
            [IllegalInfo] => Array
                (
                )
            )
        )
    )
```

```
[HitFlag] => 0
[Count] => 0
)

[AbuseInfo] => Array
(
    [HitFlag] => 0
    [Count] => 0
)

[Section] => Array
(
    [0] => Array
        (
        )
    )
)

[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-
guangzhou.myqcloud.com/text/auditing
)
```

示例二：存储桶文本文件审核

```
//当选择 Object 时，审核结果为异步返回，可通过 查询文本审核任务接口获取返回结果。
GuzzleHttp\Command\Result Object
(
    [RequestId] => NjEzYjUxY2VfZmNjYTNiMGFfNGM0MF8zNTE0OWI=
    [ContentType] => application/xml
    [ContentLength] => 324
    [JobsDetail] => Array
        (
            [Object] => exampleobject
            [JobId] => st05c02c3c123411ec953452540024deb5
            [State] => Submitted
            [CreationTime] => 2021-09-10T20:38:38+08:00
        )
    [Bucket] => examplebucket-1250000000
    [Location] => examplebucket-1250000000.ci.ap-
guangzhou.myqcloud.com/text/auditing
)
```

示例三：文本文件URL审核

```
//当选择 url 时，审核结果为异步返回，可通过 查询文本审核任务接口获取返回结果。
GuzzleHttp\Command\Result Object
(
    [RequestId] => NjEzYjUxY2VfZmNjYTNiMGFfNGM0MF8zNTE0OWI=
    [ContentType] => application/xml
    [ContentLength] => 324
    [JobsDetail] => Array
        (
            [JobId] => st05c02c3c123411ec953452540024deb5
            [State] => Submitted
            [CreationTime] => 2021-09-10T20:38:38+08:00
        )

    [Bucket] => examplebucket-1250000000
    [Location] => examplebucket-1250000000.ci.ap-
guangzhou.myqcloud.com/text/auditing
)
```

查询文本审核任务

功能说明

用于查询文本审核任务执行情况和结果。

方法原型

```
public Guzzle\Service\Resource\Model getDetectTextResult(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', // 审核时必须为 https
        'credentials'=> array(
            'secretId' => $secretId ,
            'secretKey' => $secretKey));

try {
    $result = $cosClient->getDetectTextResult(array(
        'Bucket' => 'examplebucket-1250000000', //存储桶名称, 由 BucketName-
Appid 组成, 可以在COS控制台查看 https://console.cloud.tencent.com/cos5/bucket
        'Key' => 'examplejobid', // JobId
    ));
    // 请求成功
    print_r($result);
} catch (\Exception $e) {
    // 请求失败
    echo($e);
}
```

参数说明

参数名称	类型	描述	是否必填
Bucket	String	存储桶名称, 格式: BucketName-APPID	是
Key	String	需要查询的任务 ID	是

返回结果示例

响应参数具体含义参考 [文本审核API文档](#)。

```
GuzzleHttp\Command\Result Object
(
    [RequestId] => NjEzYjU0NDhfNzgwYzdkNjRfMThiOV8zNjQ0Y2Q=
    [ContentType] => application/xml
    [ContentLength] => 725
    [JobsDetail] => Array
        (
            [Code] => Success
            [Message] =>
            [JobId] => st7a813188123511ecb6485254009f6100
            [State] => Success
            [CreationTime] => 2021-09-10T20:49:03+08:00
            [Object] => exampleobject
        )
    )
```

```
[SectionCount] => 1
[Result] => 0
[PornInfo] => Array
    (
        [HitFlag] => 0
        [Count] => 0
    )

[AdsInfo] => Array
    (
        [HitFlag] => 0
        [Count] => 0
    )

[Section] => Array
    (
        [0] => Array
            (
            )
        )
    )

)

[Key] => st7a813188123511ecb6485254009f6100
[Bucket] => examplebucket-1250000000
[Location] => examplebucket-1250000000.ci.ap-
guangzhou.myqcloud.com/text/auditing/st7a813188123511ecb6485254009f6100
)
```

Python SDK

快速入门

最近更新时间：2024-12-03 16:44:02

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'
    }
)
```

文本审核

最近更新时间：2024-11-29 09:48:13

简介

内容审核功能是由 [数据万象](#)（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

说明

使用内容审核服务需拥有数据万象使用权限：

- 主账号请 [单击此处](#) 进行角色授权。
- 子账号请参见 [授权子账号接入数据万象服务](#) 文档。

本文档提供文本审核相关的 API 概览以及 SDK 示例代码。

注意

该功能需要 COS Python SDK v5 1.9.10 及以上版本。如果当前您的COS Python SDK v5版本低于 1.9.10，请 [升级版本](#)。

API	操作描述
文本内容审核	用于进行文本内容的直接审核。
提交文本文件审核任务	提交一个文本文件审核任务。
查询文本文件审核任务结果	用来查询指定的文本审核任务。

文本审核

功能说明

本接口可用于进行文本内容的直接审核，请求方式为同步请求，可直接通过接口返回结果获取文本内容审核结果。也可用于提交一个文本文件审核任务，请求方式为异步请求，可以通过查询文本审核任务接口查询文本文件审核结果。

方法原型

```
def ci_auditing_text_submit(self, Bucket, Key=None, DetectType=None, Content=None, Callback=None, BizType=None, Url=None, UserInfo=None, DataId=None, **kwargs):
```

请求示例一：文本内容审核

```
def ci_auditing_text_submit():
    # 用户自定义业务字段
    user_info = {
        'TokenId': '123456', # 一般用于表示账号信息, 长度不超过128字节
        'Nickname': '测试', # 一般用于表示昵称信息, 长度不超过128字节
        'DeviceId': '腾讯云', # 一般用于表示设备信息, 长度不超过128字节
        'AppId': '12500000', # 一般用于表示 App 的唯一标识, 长度不超过128字节
        'Room': '1', # 一般用于表示房间号信息, 长度不超过128字节
        'IP': '127.0.0.1', # 一般用于表示 IP 地址信息, 长度不超过128字节
        'Type': '测试', # 一般用于表示业务类型, 长度不超过128字节
        'ReceiveTokenId': '789123', # 一般用于表示接收消息的用户账号, 长度不超过128
        字节
        'Gender': '男', # 一般用于表示性别信息, 长度不超过128字节
        'Level': '100', # 一般用于表示等级信息, 长度不超过128字节
        'Role': '测试人员', # 一般用于表示角色信息, 长度不超过128字节
    }
    response = client.ci_auditing_text_submit(
        Bucket=bucket_name, # 桶名称
        Content='123456test'.encode("utf-8"), # 需要审核的文本内容
        BizType='', # 表示审核策略的唯一标识
        UserInfo=user_info, # 用户自定义业务字段
        DataId='456456456', # 待审核的数据进行唯一业务标识
    )
    print(response)
```

请求示例二：提交文本文件审核任务

```
def ci_auditing_text_submit():
    # 用户自定义业务字段
    user_info = {
        'TokenId': '123456', # 一般用于表示账号信息, 长度不超过128字节
        'Nickname': '测试', # 一般用于表示昵称信息, 长度不超过128字节
        'DeviceId': '腾讯云', # 一般用于表示设备信息, 长度不超过128字节
        'AppId': '12500000', # 一般用于表示 App 的唯一标识, 长度不超过128字节
        'Room': '1', # 一般用于表示房间号信息, 长度不超过128字节
        'IP': '127.0.0.1', # 一般用于表示 IP 地址信息, 长度不超过128字节
        'Type': '测试', # 一般用于表示业务类型, 长度不超过128字节
        'ReceiveTokenId': '789123', # 一般用于表示接收消息的用户账号, 长度不超过128
        字节
        'Gender': '男', # 一般用于表示性别信息, 长度不超过128字节
        'Level': '100', # 一般用于表示等级信息, 长度不超过128字节
        'Role': '测试人员', # 一般用于表示角色信息, 长度不超过128字节
    }
    # 对cos文本文件进行审核
    response = client.ci_auditing_text_submit(
```

```
Bucket=bucket_name, # 桶名称
Key='shenhe1.txt', # 对象文件名
BizType='', # 表示审核策略的唯一标识
UserInfo=user_info, # 用户自定义业务字段
DataId='456456456', # 待审核的数据进行唯一业务标识
)
print(response)

# 对url进行审核
response = client.ci_auditing_text_submit(
    Bucket=bucket_name, # 桶名称
    Url='https://www.test.com/test.txt', # 文本文件的完整链接
    BizType='', # 表示审核策略的唯一标识
    UserInfo=user_info, # 用户自定义业务字段
    DataId='456456456', # 待审核的数据进行唯一业务标识
)
print(response)
```

参数说明

调用 `ci_auditing_text_submit` 函数，具体请求参数如下：

⚠ 注意：

- 不可同时输入 Key/Url 和 Content。
- 当选择 Key、Url 时，审核结果为异步返回，且当两者都传入时，仅会对Url传入内容进行处理。可通过 [查询文本审核任务结果](#) API 接口获取返回结果。
- 当选择 Content 时，审核结果为同步返回，可通过 [响应体](#) 查看审核结果。
- 目前仅支持中文、英文、阿拉伯数字的检测与审核。

参数名称	描述	类型	是否必选
Bucket	存储桶名称。	String	是
Key	对象文件名，例如 picture.jpg。	String	否
Url	文本文件的完整链接，例如： <code>https://www.test.com/test.txt</code> 。	String	否
BizType	表示审核策略的唯一标识，您可以通过控制台上的审核策略页面，配置您希望审核的场景，如涉黄、广告、违法违规等，配置指引： 设置公共审核策略 。您可以在控制台上获取到 BizType。BizType 填写时，此条审核请求	String	否

	将按照该审核策略中配置的场景进行审核。BizType 不填写时，将自动使用默认的审核策略。		
Content	当传入的内容为纯文本信息，原文长度不能超过10000个 utf8 编码字符。若超出长度限制，接口将会报错。	String	否
Callback	用户自定义回调地址，以 http:// 或者 https:// 开头的地址。	String	否
Callback Version	回调内容的结构，有效值：Simple（回调内容包含基本信息）、Detail（回调内容包含详细信息）。默认为 Simple。	String	否
Callback Type	回调片段类型，有效值：1（回调全部文本片段）、2（回调违规文本片段）。默认为 1。	Int	否
UserInfo	用户自定义业务字段。可传入的参数请参考 提交文本审核任务 的 UserInfo 字段。	Dict	否
DataId	该字段在审核结果中会返回原始内容，长度限制为512字节。您可以使用该字段对待审核的数据进行唯一业务标识。	String	否
Freeze	可通过该字段，设置根据审核结果给出的不同分值，对文本文件进行自动冻结，仅当 input 中审核的文本为 object 时有效。可传入的参数请参考 提交文本审核任务 的 Freeze 字段。	Dict	否

返回参数说明

调用 ci_auditing_text_submit 函数，会把 api 里面的 xml 返回转换成 dict，具体返回参数说明：当进行文本内容审核时，参见 [文本内容审核响应体](#)；当进行文本文件审核时，参见 [文本文件审核响应体](#)。

查询文本文件审核任务结果

功能说明

本接口用于主动查询指定的文本文件审核任务结果。文本文件审核功能为异步任务方式，您可以通过提交文本文件审核任务来审核您的文本文件，然后通过查询文本文件审核任务接口查询审核结果。

示例代码

```
def ci_auditing_text_query():
    response = client.ci_auditing_text_query(
        Bucket=bucket_name, # 桶名称
        JobID='st6a7d90fe311711eeaxxxxxxxx', # 需要查询的文本文件审核任务ID
    )
    print(response)
```

参数说明

调用 ci_auditing_text_query 函数，具体请求参数如下：

参数名称	描述	类型	是否必选
Bucket	存储桶名称。	String	是
JobID	任务 ID。	String	是

返回参数说明

调用 `ci_auditing_text_query` 函数，会把 api 里面的 xml 返回转换成 dict，具体返回参数可参见 [查询文本审核任务结果](#) 文档。

小程序 SDK

快速入门

最近更新时间：2025-03-14 19:14:32

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

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：

- 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。

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: function() {}, // 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);  
});
```

文本审核

最近更新时间：2024-11-29 09:48:13

简介

内容审核功能是由 [数据万象](#)（Cloud Infinite，CI）提供的，数据万象将处理能力与 COS SDK 完全结合，您可以直接按照本篇文档指引进行使用。

说明：

使用内容审核服务需拥有数据万象使用权限：

- 主账号请 [单击此处](#) 进行角色授权。
- 子账号请参见 [授权子账号接入数据万象服务](#) 文档。

本文档提供关于文本审核的 API 概览和 SDK 示例代码。

注意：

COS 小程序 SDK 版本需要大于等于 v1.1.1。

API	操作描述
文本内容审核	用于进行一次文本内容审核
提交文本文件审核任务	用于提交一个文本文件审核任务
查询文本审核任务结果	用于查询指定的文本审核任务

文本内容审核

功能说明

用于进行一次文本内容审核，会同步返回审核结果。

请求示例

```
// sdk的引入及初始化cos请参考
https://cloud.tencent.com/document/product/436/31953
function postTextContentAuditing() {
  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，比如ap-beijing，必须
  };
  const host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
  const key = 'text/auditing'; // 固定值，必须
```

```
const url = `https://${host}/${key}`;
const body = COS.util.json2xml({
  Request: {
    Input: {
      // 使用 COS.util.encodeBase64 方法需要sdk版本至少为1.4.19
      Content: COS.util.encodeBase64('hello'), /* 需要审核的文本内容 */
    },
    Conf: {
      BizType: '',
    }
  }
});
cos.request({
  Method: 'POST', // 固定值, 必须
  Url: url, // 请求的url, 必须
  Key: key, // 固定值, 必须
  ContentType: 'application/xml', // 固定值, 必须
  Body: body // 请求体参数, 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);
  }
});
}
postTextContentAuditing();
```

参数说明

cos.request 方法参数说明:

参数名称	描述	类型	是否必选
Method	请求方法, 固定值	String	是
Url	请求 url, 固定值	String	是
Key	固定值	String	是
ContentType	固定值	String	是
Body	请求体详情请参见 文本内容审核	Container	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 文本内容审核	Object

相关链接

[API 文档](#)

[功能指南](#)

提交文本文件审核任务

功能说明

用于提交一个文本文件内容审核任务。

请求示例

```
// sdk的引入及初始化cos请参考  
https://cloud.tencent.com/document/product/436/31953  
function postTextAuditing() {  
  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，比如ap-beijing，必须
```

```
};
const host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
const key = 'text/auditing'; // 固定值, 必须
const url = `https://${host}/${key}`;
const body = COS.util.json2xml({
  Request: {
    Input: {
      // Url与Object二选一传入
      Object: 'test.txt', // 文件存储在cos的路径, 支持传入中文, 比如: 中文.txt
      // Url: `www.test.com/${encodeURIComponent('中文.txt')}`, // 文本文件url
      // 注意中文路径需要encode
    },
    Conf: {
      BizType: '',
    }
  }
});
cos.request({
  Method: 'POST', // 固定值, 必须
  Url: url, // 请求的url, 必须
  Key: key, // 固定值, 必须
  ContentType: 'application/xml', // 固定值, 必须
  Body: body // 请求体参数, 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);
  }
});
}
postTextAuditing();
```

参数说明

cos.request 方法参数说明:

参数名称	描述	类型	是否必选
Method	请求方法, 固定值	String	是
Url	请求 url, 固定值	String	是
Key	固定值	String	是

ContentType	固定值	String	是
Body	请求体详情请参见 文本文件审核	Container	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 文本文件审核	Object

相关链接

[API 文档](#)

[功能指南](#)

查询文本审核任务

功能说明

用于查询文本审核任务执行情况和结果。

请求示例

```
// sdk的引入及初始化cos请参考
https://cloud.tencent.com/document/product/436/31953
function getTextAuditingResult () {
  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, 比如ap-beijing, 必须
  };
  const jobId = 'st8d88c664aff511ecb23352540078cxxx'; // jobId 可以通过提交文本审核任务返回 (Input传入Object)
  const host = config.Bucket + '.ci.' + config.Region + '.myqcloud.com';
  const key = `text/auditing/${jobId}`; // 固定值, 必须
  const url = `https://${host}/${key}`; + jobId;
  cos.request({
    Method: 'GET', // 固定值, 必须
    Url: url, // 请求的url, 必须
    Key: key, // 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);
    }
  });
}
getTextAuditingResult ();
```

参数说明

cos.request 方法参数说明:

参数名称	描述	类型	是否必选
Method	请求方法, 固定值	String	是
Url	请求 url, 固定值	String	是
Key	固定值: text/auditing/ 要查询的 jobId	String	是

参数名称	描述	类型	是否必选
jobId	需要查询的任务 ID	String	是

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 查询文本审核任务	Object

相关链接

[API 文档](#)

[功能指南](#)