对象存储

SDK 文档

产品文档

腾讯云

文档目录

SDK 文档

SDK 概览

最近更新时间：2021-07-07 10:28:44

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

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

准备工作

最近更新时间：2021-12-27 12:43:13

在使用 SDK 时可能会涉及以下概念和术语，建议您提前了解并准备相关信息：

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

说明

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

Android SDK 快速入门

最近更新时间：2024-01-04 15:52:02

相关资源

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

说明

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

准备工作

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

第一步：安装 SDK

方式一：自动集成（推荐）

说明

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

使用 mavenCentral 仓库

在项目级别（通常是根目录下）的 `build.gradle` 中添加：

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

```
}
```

标准版 SDK

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

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

精简版 SDK

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

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

关闭腾讯灯塔上报功能

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

说明

腾讯灯塔只对 COS 侧的请求性能进行监控，不会上报业务侧数据。

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

版本为5.8.0以及以上：

修改 cos-android 的依赖为

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

    //lite 版本修改为
    implementation 'com.qcloud.cos:cos-android-lite-nobeacon:x.x.x'
}
```

版本为5.5.8至5.7.9：

添加去掉 beacon 的语句

```
dependencies {
    ...
```

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

方式二：手动集成

1. 下载归档文件

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

2. 集成到项目中

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

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

第二步：配置权限

网络权限

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

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

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

存储权限

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

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

注意

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

第三步：开始使用

注意

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

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

1. 实现获取临时密钥

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

```
public static class MySessionCredentialProvider
        extends BasicLifecycleCredentialProvider {

    @Override
    protected QCloudLifecycleCredentials fetchNewCredentials()
            throws QCloudClientException {

        // 首先从您的临时密钥服务器获取包含了密钥信息的响应

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

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

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

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

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

使用永久密钥进行本地调试

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

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

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

使用服务端计算的签名对请求授权

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

```
QCloudSelfSigner myQCloudSelfSigner = new QCloudSelfSigner() {
    /**
     * 对请求进行签名
     *
     * @param request 需要签名的请求
     * @throws QCloudClientException 客户端异常
     */
    @Override
    public void sign(QCloudHttpRequest request) throws QCloudClientException {
        // 1. 把 request 的请求参数传给服务端计算签名
        String auth = "get auth from server";
        // 2. 给请求添加签名
        request.addHeader(HttpConstants.Header.AUTHORIZATION, auth);
    }
});
```

2. 初始化 COS Service

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

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

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

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

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

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

说明

关于存储桶不同地域的简称请参考 地域和访问域名。

第四步：访问 COS 服务

上传对象

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

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

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

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

下载对象

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

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

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

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

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

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

```
    public void onFail(CosXmlRequest request,
                       @Nullable CosXmlClientException clientException,
                       @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
//设置任务状态回调，可以查看任务过程
cosxmlDownloadTask.setTransferStateListener(new TransferStateListener() {
    @Override
    public void onStateChanged(TransferState state) {
        // todo notify transfer state
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

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

Android SDK 常见问题

最近更新时间：2024-01-04 15:52:02

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

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

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

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

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

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

调用接口时报错 NoClassDefFoundError，该如何处理？

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

SDK 获取手机权限问题，该如何处理？

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

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

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

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

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

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

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

通过 TransferManager 上传和下载报错权限问题，该如何处理？

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

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

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

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

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

在回调中直接操作 ui 导致应用报错，该如何处理？

sdk 回调线程不一定是主线程，请不要直接操作 ui。

上传时报错 calculate md5 error，该如何处理？

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

请求返回 ServerError 错误，该如何处理？

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

调用接口报错 403 权限错误，该如何处理？

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

Android SDK 是否支持断点续传？

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

快速体验

最近更新时间：2024-01-04 15:52:02

背景

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

相关资源

本文涉及的 Demo 都存放在 Github 仓库，用户可前往获取。

准备工作

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

搭建用户客户端

配置客户端

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

注意

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

运行示例 Demo

查询存储桶列表

启动示例 App 后，将展示用户已创建的所有存储桶。

创建存储桶

点击右上角**新建桶**，在配置页面输入桶名称并选择存储桶的所属 地域。

查询对象列表

选择点击某个存储桶，将看到该存储桶内所有文件以及文件夹。

上传文件

在文件列表页面点击右上角的**上传**，然后选择文件进行上传。

下载文件

在对象列表中，点击文件下方的**下载**按钮，即可下载文件。

存储桶操作

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于存储桶基本操作的相关 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

查询存储桶列表

功能说明

用于查询指定账号下所有存储桶列表。

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

创建存储桶

功能说明

创建一个存储桶（PUT Bucket）。

示例代码

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

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

说明

更多完整示例，请前往 GitHub 查看。

检索存储桶及其权限

功能说明

HEAD Bucket 请求可以确认该存储桶是否存在，是否有权限访问。有以下几种情况：
存储桶存在且有读取权限，返回 HTTP 状态码为200。
无存储桶读取权限，返回 HTTP 状态码为403。
存储桶不存在，返回 HTTP 状态码为404。

示例代码

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

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

说明

更多完整示例，请前往 GitHub 查看。

删除存储桶

功能说明

删除指定的存储桶（DELETE Bucket）。

注意

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

示例代码

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

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

说明

更多完整示例，请前往 GitHub 查看。

对象操作
上传对象

最近更新时间：2024-01-04 15:52:02

简介

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

简单操作

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

分块操作

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

高级接口（推荐）

上传对象

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

示例代码一: 上传本地文件

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

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

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

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

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

```
                        @Nullable CosXmlClientException clientException,
                        @Nullable CosXmlServiceException serviceException) {
        if (clientException != null) {
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
//设置任务状态回调，可以查看任务过程
cosxmlUploadTask.setTransferStateListener(new TransferStateListener() {
    @Override
    public void onStateChanged(TransferState state) {
        // todo notify transfer state
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

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

示例代码二: 上传二进制数据

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

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

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

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

示例代码三: 流式上传

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

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

// 流式上传
InputStream inputStream =
        new ByteArrayInputStream("this is a test".getBytes(Charset.forName(
                "UTF-8")));
COSXMLUploadTask cosxmlUploadTask = transferManager.upload(bucket, cosPath,
        inputStream);

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

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

示例代码四: 通过 URI 上传

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

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

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

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

示例代码五: 设置智能分块阈值

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

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码六: 上传暂停、继续与取消

对于上传任务，可以通过以下方式暂停：

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

暂停之后，可以通过以下方式续传：

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

也通过以下方式取消上传：

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码七: 批量上传

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

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码八：创建目录

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

```
PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, new byte[
cosXmlService.putObjectAsync(putObjectRequest, new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutObjectResult putObjectResult =
                (PutObjectResult) result;
    }

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

说明

更多完整示例，请前往 GitHub 查看。

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

示例代码九：设置低优先级任务

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


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

说明

更多完整示例，请前往 GitHub 查看。

简单操作

简单上传对象

功能说明

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

注意

Key（文件名）不能以 `/` 结尾，否则会被识别为文件夹。

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

示例代码

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

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

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

```
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

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

表单上传对象

功能说明

使用表单请求上传对象。

示例代码

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

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

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

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

```
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

分块操作

这里说明下分块上传的流程。

分块上传的流程

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

分块继续上传的流程

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

终止分块上传的流程

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

查询分块上传

功能说明

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

示例代码

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

```
cosXmlService.listMultiUploadsAsync(listMultiUploadsRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
        ListMultiUploadsResult listMultiUploadsResult =
                (ListMultiUploadsResult) result;
    }

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

说明

更多完整示例，请前往 GitHub 查看。

初始化分块上传

功能说明

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

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

上传分块

分块上传对象（Upload Part）。

示例代码

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

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

查询已上传的分块

功能说明

查询特定分块上传操作中的已上传的块（List Parts）。

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

完成分块上传

功能说明

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

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

终止分块上传

功能说明

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

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

下载对象

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

高级接口（推荐）

下载对象

高级接口支持暂停、恢复以及取消下载请求，同时支持断点下载功能。

示例代码一: 下载对象

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

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

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码二: 下载暂停、继续与取消

对于下载任务，可以通过以下方式暂停：

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

暂停之后，可以通过以下方式续传：

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

也通过以下方式取消下载：

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码三: 设置下载支持断点续传

```
// 初始化 TransferConfig，这里使用默认配置，如果需要定制，请参考 SDK 接口文档
// TransferManager 支持断点下载，您只需要保证 bucket、cosPath、savePathDir、savedFileName
// 参数一致，SDK 便会从上次已经下载的位置继续下载。
TransferConfig transferConfig = new TransferConfig.Builder().build();
//初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
//本地目录路径
String savePathDir = context.getExternalCacheDir().toString();
//本地保存的文件名，若不填（null），则与 COS 上的文件名一样
String savedFileName = "exampleobject";

GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, cosPath, savePathD

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码四: 批量下载

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

```
        "exampleobject2",
        "exampleobject3",
};

for (String cosPath : cosPaths) {

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

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码五: 下载目录

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

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码六: 匿名下载（不用传入密钥，对公开文件进行下载）

```
// 初始化 TransferConfig，这里使用默认配置，如果需要定制，请参考 SDK 接口文档
TransferConfig transferConfig = new TransferConfig.Builder().build();
// 初始化 TransferManager
CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig.Builder()
        .setRegion("ap-guangzhou")
        .builder();
// 匿名下载生成 CosXmlService 不需要传入密钥生成器
CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig);
TransferManager transferManager = new TransferManager(cosXmlService, transferConfig
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
// 本地目录路径
String savePathDir = context.getExternalCacheDir().toString();
// 本地保存的文件名，若不填（null），则与 COS 上的文件名一样
String savedFileName = "exampleobject";

GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, cosPath, savePathD

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

说明

更多完整示例，请前往 GitHub 查看。

简单操作

下载对象

功能说明

下载一个 Object（文件/对象）至本地（GET Object）。

示例代码

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

复制与移动对象

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于复制、移动对象的 API 概览以及 SDK 示例代码。

简单操作

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

分块操作

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

复制对象

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

示例代码

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

```
//初始化 TransferManager
TransferManager transferManager = new TransferManager(cosXmlService,
        transferConfig);

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

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

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

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

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

移动对象

移动对象主要包括两个操作：复制源对象到目标位置，删除源对象。
由于 COS 通过存储桶名称（Bucket）和对象键（ObjectKey）来标识对象。移动对象也就意味着修改这个对象的标识，COS Android SDK 目前没有提供修改对象唯一标识名的单独接口，但是可以通过组合**复制对象**加上**删除对象**的基本操作，来达到修改对象标识的目的，从而实现移动对象。
例如将 mybucket-1250000000 这个存储桶中的 picture.jpg 这个对象移动到同个存储桶的 doc 路径下。首先可以复制 picture.jpg 对象到存储桶的 doc 路径下，即对象键设定为 doc/picture.jpg，复制完成后删除 picture.jpg 这个对象，来实现“移动”的效果。
同样的，如果想将 mybucket-1250000000 这个存储桶里的 picture.jpg 这个对象移动到 myanothorbucket-1250000000 这个存储桶里，可以先将对象复制到 myanothorbucket-1250000000 存储桶，然后删除原来的对象。

示例代码

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

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

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

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

    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

复制对象（修改属性）

复制文件到目标路径（PUT Object-Copy）。

示例代码一: 复制对象时保留对象属性

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

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

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

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

```
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码二: 复制对象时替换对象属性

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

// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(bucket, cosPath,
        copySourceStruct);
copyObjectRequest.setCopyMetaDataDirective(MetaDataDirective.REPLACED);
copyObjectRequest.setXCOSMeta("x-cos-metadata-oldKey", "newValue");

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

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码三: 修改对象元数据

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码四: 修改对象存储类型

```
String appId = "1250000000"; //账号 APPID
```

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

CopyObjectRequest copyObjectRequest = new CopyObjectRequest(bucket, cosPath,
        copySourceStruct);
// 修改为低频存储
copyObjectRequest.setCosStorageClass(COSStorageClass.STANDARD_IA);

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

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

说明

更多完整示例，请前往 GitHub 查看。

列出对象

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于列出对象操作相关的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

示例代码一: 获取第一页数据

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

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

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

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

```
        if (getBucketResult.listBucket.isTruncated) {
            // 表示数据被截断，需要拉取下一页数据
            prevPageResult = getBucketResult;
        }
    }

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码二：请求下一页数据

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


GetBucketRequest getBucketRequest = new GetBucketRequest(bucketName);

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

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

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

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

```
    }

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码三：获取对象列表与子目录

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

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

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

// 定界符为一个符号，如果有 Prefix，
// 则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，
// 然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始
getBucketRequest.setDelimiter("/");

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

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

```
            if (clientException != null) {
                clientException.printStackTrace();
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

查询对象历史版本列表

功能说明

查询开启版本控制的存储桶下的部分或者全部对象。

示例代码一：获取对象历史版本列表第一页数据

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

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

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码二：获取对象历史版本列表下一页数据

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

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

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

// prevPageResult 是上一页的返回结果，这里的 nextMarker 与 nextVersionIdMarker
// 表示下一页的起始位置
getBucketRequest.setKeyMarker(prevPageResult.listVersionResult
        .nextKeyMarker);
getBucketRequest.setVersionIdMarker(prevPageResult.listVersionResult
        .nextVersionIdMarker);

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

删除对象

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于对象的删除操作相关的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

删除单个对象

功能说明

删除指定的对象（DELETE Object）。

示例代码

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

删除多个对象

功能说明

批量删除多个对象（Delete Multi Objects）。

示例代码

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

DeleteMultiObjectRequest deleteMultiObjectRequest =
        new DeleteMultiObjectRequest(bucket, objectList);
// Quiet 模式只返回报错的 Object 信息。否则返回每个 Object 的删除结果。
deleteMultiObjectRequest.setQuiet(true);
cosXmlService.deleteMultiObjectAsync(deleteMultiObjectRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult result) {
        DeleteMultiObjectResult deleteMultiObjectResult =
                (DeleteMultiObjectResult) result;
    }

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

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

说明

更多完整示例，请前往 GitHub 查看。

删除目录

功能说明

COS 上的文件夹概念是以 `/` 分隔对象名，形成类似文件系统的路径，从而模拟出来的。所以删除文件夹的操作，在 COS 上相当于删除一批有着同样前缀的对象。例如：文件夹 `prefix/` ，代表的是以 `prefix/` 为前缀的所有对象，所以删除 `prefix/` 意味着删除以 `prefix/` 为前缀的所有对象。
目前 COS Android SDK 没有提供一个接口去支持这样的操作，但是可以通过基本操作的组合，达到同样的效果。

示例代码

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

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

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

do {
    try {
        getBucketResult = cosXmlService.getBucket(getBucketRequest);
        List<ListBucket.Contents> contents = getBucketResult.listBucket.contentsLis
        DeleteMultiObjectRequest deleteMultiObjectRequest = new DeleteMultiObjectRe
        for (ListBucket.Contents content : contents) {
            deleteMultiObjectRequest.setObjectList(content.key);
        }
```

```
            cosXmlService.deleteMultiObject(deleteMultiObjectRequest);
            getBucketRequest.setMarker(getBucketResult.listBucket.nextMarker);
        } catch (CosXmlClientException e) {
            e.printStackTrace();
            return;
        } catch (CosXmlServiceException e) {
            e.printStackTrace();
            return;
        }
    } while (getBucketResult.listBucket.isTruncated);
```

说明

更多完整示例，请前往 GitHub 查看。

恢复归档对象

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于恢复归档对象操作相关的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

恢复归档对象

功能说明

将归档类型的对象取回访问（POST Object restore）。

示例代码

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

查询对象元数据

最近更新时间：2024-01-04 15:52:02

简介

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

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

查询对象元数据

功能说明

查询 Object 的 Meta 信息。

示例代码

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

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

```
            } else {
                serviceException.printStackTrace();
            }
        }
    });
```

说明

更多完整示例，请前往 GitHub 查看。

生成预签名 URL

最近更新时间：2024-01-04 15:52:02

简介

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

生成对象预签名链接

示例代码一：生成预签名上传链接

```
try {
      //存储桶名称
    String bucket = "examplebucket-1250000000";
    // 对象在存储桶中的位置标识符，对象键(Key)是对象在存储桶中的唯一标识。详情请参见 [对象键](ht
    // 注意：用户无需对 cosPath 进行编码操作
    String cosPath = "exampleobject";
    //请求 HTTP 方法
    String method = "PUT";
    PresignedUrlRequest presignedUrlRequest = new PresignedUrlRequest(bucket
            , cosPath) {
        @Override
        public RequestBodySerializer getRequestBody()
                throws CosXmlClientException {
            //用于计算 put 等需要带上 body 的请求的签名 URL
            return RequestBodySerializer.string("text/plain",
                    "this is test");
        }
    };
```

```
    presignedUrlRequest.setRequestMethod(method);
    // 设置签名有效期为 60s，注意这里是签名有效期，您需要自行保证密钥有效期
    presignedUrlRequest.setSignKeyTime(60);
    // 设置不签名 Host
    presignedUrlRequest.addNoSignHeader("Host");
    String urlWithSign = cosXmlService.getPresignedURL(presignedUrlRequest);
} catch (CosXmlClientException e) {
    e.printStackTrace();
}
```

说明

更多完整示例，请前往 GitHub 查看。

示例代码二：生成预签名下载链接

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

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

    String urlWithSign = cosXmlService.getPresignedURL(presignedUrlRequest);

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

说明

更多完整示例，请前往 GitHub 查看。

预请求跨域配置

最近更新时间：2024-01-04 15:52:02

简介

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

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

预请求跨域配置

功能说明

获取预请求跨域配置（Options Object）。

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

服务端加密

最近更新时间：2024-01-04 15:52:02

简介

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API。

使用 COS 托管加密密钥的服务端加密（SSE-COS）保护数据

功能说明

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

示例代码

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

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

说明

更多完整示例，请前往 GitHub 查看。

使用 KMS 托管加密密钥的服务端加密（SSE-KMS）保护数据

功能说明

SSE-KMS 加密即使用 KMS 托管密钥的服务端加密。KMS 是腾讯云推出的一款安全管理类服务，使用经过第三方认证的硬件安全模块 HSM（Hardware Security Module）来生成和保护密钥。它能够帮助用户轻松创建和管理密钥，

满足用户多应用多业务的密钥管理需求以及满足监管和合规要求。关于如何开通 KMS 服务请参考：服务端加密概述。

示例代码

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

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

说明

更多完整示例，请前往 GitHub 查看。

使用客户提供的加密密钥的服务端加密 （SSE-C）保护数据

功能说明

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

注意

该加密所运行的服务需要使用 HTTPS 请求。

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

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

示例代码

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

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

说明

更多完整示例，请前往 GitHub 查看。

单链接限速

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于调用上传下载接口时对链接进行限速。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

使用说明

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

示例代码一：上传时对单链接限速

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

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

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

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

//设置上传进度回调
```

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

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码二：下载时对单链接限速

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

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

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

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

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

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

检索对象内容

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于检索对象内容操作相关的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

检索对象内容

功能说明

COS Select 支持检索以下格式的对象数据：

CSV 格式：对象以 CSV 格式存储，并以固定的分隔符划分。

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

注意

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

CSV、JSON 对象需要以 UTF-8 格式编码。

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

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

示例代码

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

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

```
);

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

异地容灾
版本控制

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于版本控制的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置版本控制

功能说明

设置指定存储桶的版本控制功能。开启版本控制功能后，只能暂停，不能关闭。

示例代码

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

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

```
    }

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

说明

更多完整示例，请前往 GitHub 查看。

查询版本控制

功能说明

查询指定存储桶的版本控制信息。

获取存储桶版本控制的状态，需要有该存储桶的读权限。

有三种版本控制状态：未启用版本控制、启用版本控制和暂停版本控制。

示例代码

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

跨地域复制

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于跨地域复制的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置跨地域复制

功能说明

设置指定存储桶的跨地域复制规则。

示例代码

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

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

PutBucketReplicationRequest.RuleStruct ruleStruct =
        new PutBucketReplicationRequest.RuleStruct();
//用来标注具体 Rule 的名称
ruleStruct.id = "replication_01";
```

```
//标识 Rule 是否生效。true：生效；false：不生效
ruleStruct.isEnable = true;
//目标存储桶地域信息
ruleStruct.region = "ap-beijing";
// 目标存储桶
ruleStruct.bucket = "destinationbucket-1250000000";
//前缀匹配策略
ruleStruct.prefix = "dir/";
putBucketReplicationRequest.setReplicationConfigurationWithRule(ruleStruct);

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

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

说明

更多完整示例，请前往 GitHub 查看。

查询跨地域复制

功能说明

查询指定存储桶的跨地域复制规则。

示例代码

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

```
GetBucketReplicationRequest getBucketReplicationRequest =
        new GetBucketReplicationRequest(bucket);

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

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

说明

更多完整示例，请前往 GitHub 查看。

删除跨地域复制

功能说明

删除指定存储桶的跨地域复制规则。

示例代码

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

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

```
                DeleteBucketReplicationResult deleteBucketReplicationResult =
                        (DeleteBucketReplicationResult) result;
            }

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

说明

更多完整示例，请前往 GitHub 查看。

数据管理
生命周期

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于生命周期的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置生命周期

功能说明

设置指定存储桶的生命周期配置信息。

示例代码

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

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

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

putBucketLifecycleRequest.setRuleList(rule);

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

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

说明

更多完整示例，请前往 GitHub 查看。

查询生命周期

功能说明

查询存储桶的生命周期管理配置。

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

删除生命周期

功能说明

删除存储桶生命周期管理的配置。

示例代码

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

cosXmlService.deleteBucketLifecycleAsync(deleteBucketLifecycleRequest,
```

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

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

说明

更多完整示例，请前往 GitHub 查看。

日志管理

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于日志管理的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置日志管理

功能说明

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

示例代码

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

cosXmlService.putBucketLoggingAsync(putBucketLoggingRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutBucketLoggingResult putBucketLoggingResult =
                (PutBucketLoggingResult) result;
```

```
    }

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

说明

更多完整示例，请前往 GitHub 查看。

查询日志管理

功能说明

GET Bucket logging 用于查询指定存储桶的日志配置信息。

示例代码

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

对象标签

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于对象标签的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置对象标签

上传时添加标签

功能说明

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

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

为已存在的对象添加标签

功能说明

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

示例代码

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

```
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
PutObjectTaggingRequest putObjectTaggingRequest = new PutObjectTaggingRequest(bucke
putObjectTaggingRequest.addTag("key", "value");
try {
    PutObjectTaggingResult putObjectTaggingResult = cosXmlService.putObjectTagging(
} catch (CosXmlClientException clientException) {
    clientException.printStackTrace();
} catch (CosXmlServiceException serviceException) {
    serviceException.printStackTrace();
}
```

说明

更多完整示例，请前往 GitHub 查看。

查询对象标签

功能说明

查询指定对象下已有的对象标签。

示例代码

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

说明

更多完整示例，请前往 GitHub 查看。

删除对象标签

功能说明

删除指定对象下已有的对象标签。

示例代码

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

说明

更多完整示例，请前往 GitHub 查看。

存储桶标签

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于存储桶标签的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置存储桶标签

功能说明

PUT Bucket tagging 用于为已存在的存储桶设置标签。

示例代码

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

cosXmlService.putBucketTaggingAsync(putBucketTaggingRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        PutBucketTaggingResult putBucketTaggingResult =
                (PutBucketTaggingResult) result;
```

```
    }

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

说明

更多完整示例，请前往 GitHub 查看。

查询存储桶标签

功能说明

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

示例代码

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

删除存储桶标签

功能说明

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

示例代码

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

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

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

```
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

静态网站

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于静态网站的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置静态网站

功能说明

PUT Bucket website 用于为存储桶配置静态网站。

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

查询静态网站配置

功能说明

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

示例代码

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

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

```
            clientException.printStackTrace();
        } else {
            serviceException.printStackTrace();
        }
    }
});
```

说明

更多完整示例，请前往 GitHub 查看。

删除静态网站配置

功能说明

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

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

清单

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于清单的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置清单任务

功能说明

PUT Bucket inventory 用于在存储桶中创建清单任务。

示例代码

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

错误码说明

该请求可能会发生的一些常见的特殊错误如下：

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

查询清单任务

功能说明

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

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

删除清单任务

功能说明

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

示例代码

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

```
deleteBucketInventoryRequest.setInventoryId("exampleInventoryId");

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

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

说明

更多完整示例，请前往 GitHub 查看。

访问管理
跨域访问

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于跨域访问的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置跨域配置

功能说明

设置指定存储桶的跨域名访问配置信息（PUT Bucket cors）。

示例代码

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

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

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

```
corsRule.maxAgeSeconds = 5000;

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

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

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

putBucketCORSRequest.addCORSRule(corsRule);

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

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

说明

更多完整示例，请前往 GitHub 查看。

查询跨域配置

功能说明

查询指定存储桶的跨域名访问配置信息（GET Bucket cors）。

示例代码

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

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

说明

更多完整示例，请前往 GitHub 查看。

删除跨域配置

功能说明

删除指定存储桶的跨域名访问配置（DELETE Bucket cors）。

示例代码

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

```
DeleteBucketCORSRequest deleteBucketCORSRequest =
        new DeleteBucketCORSRequest(bucket);
cosXmlService.deleteBucketCORSAsync(deleteBucketCORSRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        DeleteBucketCORSResult deleteBucketCORSResult =
                (DeleteBucketCORSResult) result;
    }

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

说明

更多完整示例，请前往 GitHub 查看。

自定义域名

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于自定义域名的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置自定义域名

功能说明

PUT Bucket domain 用于为存储桶配置自定义域名。

示例代码

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

cosXmlService.putBucketDomainAsync(putBucketDomainRequest,
        new CosXmlResultListener() {
```

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

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

说明

更多完整示例，请前往 GitHub 查看。

返回错误码说明

该请求可能会发生的一些常见的特殊错误如下：

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

查询自定义域名

功能说明

GET Bucket domain 用于查询存储桶的自定义域名信息。

示例代码

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

```
        new GetBucketDomainRequest(bucket);
cosXmlService.getBucketDomainAsync(getBucketDomainRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetBucketDomainResult getBucketTaggingResult =
                (GetBucketDomainResult) result;
    }

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

说明

更多完整示例，请前往 GitHub 查看。

返回参数说明

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

删除自定义域名

功能说明

DELETE Bucket domain 用于删除存储桶的自定义域名信息。

注意

COS Android SDK 版本需要大于等于 v5.9.8。

示例代码

```
// 存储桶名称，由 bucketname-appid 组成，appid 必须填入，可以在 COS 控制台查看存储桶名称。 ht
String bucket = "examplebucket-1250000000";
DeleteBucketDomainRequest deleteBucketDomainRequest =
        new DeleteBucketDomainRequest(bucket);
cosXmlService.deleteBucketDomainAsync(deleteBucketDomainRequest,
        new CosXmlResultListener() {
            @Override
            public void onSuccess(CosXmlRequest request, CosXmlResult result) {
                // 详细字段请查看api文档或者SDK源码
                DeleteBucketDomainResult deleteBucketDomainResult =
                        (DeleteBucketDomainResult) result;
            }
            // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFa
            // clientException 的类型为 CosXmlClientException?, serviceException 的类
            @Override
            public void onFail(CosXmlRequest cosXmlRequest,
                               @Nullable CosXmlClientException clientException,
                               @Nullable CosXmlServiceException serviceException) {
                if (clientException != null) {
                    clientException.printStackTrace();
                } else {
                    serviceException.printStackTrace();
                }
            }
        });
```

说明

更多完整示例，请前往 GitHub 查看。

访问控制

最近更新时间：2024-01-04 15:52:02

简介

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

存储桶 ACL

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

对象 ACL

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

存储桶 ACL

设置存储桶 ACL

功能说明

设置指定存储桶的访问权限控制列表（ACL）。

示例代码

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

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

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

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

查询存储桶 ACL

功能说明

查询指定存储桶的访问权限控制列表（ACL）。

示例代码

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

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

说明

更多完整示例，请前往 GitHub 查看。

对象 ACL

设置对象 ACL

功能说明

设置存储桶中某个对象的访问控制列表（ACL）。

示例代码

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

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

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

查询对象 ACL

功能说明

查询对象的访问控制列表。

示例代码

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

```
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即对象键。
GetObjectACLRequest getBucketACLRequest = new GetObjectACLRequest(bucket,
        cosPath);
cosXmlService.getObjectACLAsync(getBucketACLRequest,
        new CosXmlResultListener() {
    @Override
    public void onSuccess(CosXmlRequest request, CosXmlResult result) {
        GetObjectACLResult getObjectACLResult = (GetObjectACLResult) result;
    }

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

说明

更多完整示例，请前往 GitHub 查看。

防盗链

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于防盗链的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置防盗链配置

功能说明

设置指定存储桶的防盗链配置信息（PUT Bucket referer）。

示例代码

```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
PutBucketRefererRequest putBucketRefererRequest = new PutBucketRefererRequest(
        bucket, true, RefererConfiguration.RefererType.White);
putBucketRefererRequest.setAllowEmptyRefer(false);
ArrayList<RefererConfiguration.Domain> domainList = new ArrayList<>();
domainList.add(new RefererConfiguration.Domain("*.qq.com"));
domainList.add(new RefererConfiguration.Domain("*.qcloud.com"));
domainList.add(new RefererConfiguration.Domain("*.google.com"));
putBucketRefererRequest.setDomainList(domainList);
cosXmlService.putBucketRefererAsync(putBucketRefererRequest,
        new CosXmlResultListener() {
            @Override
            public void onSuccess(CosXmlRequest request, CosXmlResult result) {
                // 详细字段请查看api文档或者SDK源码
```

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

说明

更多完整示例，请前往 GitHub 查看。

查询防盗链配置

功能说明

查询指定存储桶的防盗链配置信息（GET Bucket referer）。

示例代码

```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
GetBucketRefererRequest getBucketRefererRequest = new GetBucketRefererRequest(bucke
cosXmlService.getBucketRefererAsync(getBucketRefererRequest,
        new CosXmlResultListener() {
            @Override
            public void onSuccess(CosXmlRequest request, CosXmlResult result) {
                // 详细字段请查看api文档或者SDK源码
                GetBucketRefererResult getBucketRefererResult =
                        (GetBucketRefererResult) result;
            }
            // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFa
            // clientException 的类型为 CosXmlClientException?, serviceException 的类
            @Override
            public void onFail(CosXmlRequest cosXmlRequest,
                               @Nullable CosXmlClientException clientException,
                               @Nullable CosXmlServiceException serviceException) {
```

```
                if (clientException != null) {
                    clientException.printStackTrace();
                } else {
                    serviceException.printStackTrace();
                }
            }
        });
```

说明

更多完整示例，请前往 GitHub 查看。

存储桶策略

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于存储桶策略的 API 概览以及 SDK 示例代码。

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

设置存储桶策略

功能说明

设置指定存储桶的权限策略（PUT Bucket policy）。

注意

COS Android SDK 版本需要大于等于 v5.9.8。

示例代码

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

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

说明

更多完整示例，请前往 GitHub 查看。

查询存储桶策略

功能说明

查询指定存储桶的权限策略（GET Bucket policy）。

注意

COS Android SDK 版本需要大于等于 v5.9.8。

示例代码

```
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
final GetBucketPolicyRequest getBucketPolicyRequest =
        new GetBucketPolicyRequest(bucket);
cosXmlService.getBucketPolicyAsync(getBucketPolicyRequest,
        new CosXmlResultListener() {
            @Override
            public void onSuccess(CosXmlRequest request, CosXmlResult result) {
                // 详细字段请查看api文档或者SDK源码
                GetBucketPolicyResult getBucketPolicyResult =
                        (GetBucketPolicyResult) result;
                String policy = getBucketPolicyResult.policy;
            }
            // 如果您使用 kotlin 语言来调用，请注意回调方法中的异常是可空的，否则不会回调 onFa
            // clientException 的类型为 CosXmlClientException?, serviceException 的类
            @Override
            public void onFail(CosXmlRequest cosXmlRequest,
                               @Nullable CosXmlClientException clientException,
                               @Nullable CosXmlServiceException serviceException) {
                if (clientException != null) {
                    clientException.printStackTrace();
                } else {
                    serviceException.printStackTrace();
                }
            }
        });
```

说明

更多完整示例，请前往 GitHub 查看。

删除存储桶策略

功能说明

删除指定存储桶的权限策略（DELETE Bucket policy）。

注意

COS Android SDK 版本需要大于等于 v5.9.8。

示例代码

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

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

说明

更多完整示例，请前往 GitHub 查看。

数据校验

CRC64 校验

最近更新时间：2024-01-04 15:52:02

简介

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

操作说明

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

SDK 说明

您在上传或者下载成功后，可以在响应头部中获取 CRC64 值。

注意

COS Android SDK 版本需要大于等于 v5.7.5。

上传请求示例

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

// 2. 初始化 PutObjectRequest
// 存储桶名称，由bucketname-appid 组成，appid必须填入，可以在COS控制台查看存储桶名称。 https:
String bucket = "examplebucket-1250000000";
String cosPath = "exampleobject"; //对象在存储桶中的位置标识符，即称对象键
String srcPath = "examplefilepath"; //本地文件的绝对路径
PutObjectRequest putObjectRequest = new PutObjectRequest(bucket,
        cosPath, srcPath);

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

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

说明

更多完整示例，请前往 GitHub 查看。

下载请求示例

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

CRC64 校验

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

图片处理
图片持久化处理

最近更新时间：2024-01-04 15:52:02

简介

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

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

上传时使用图片处理

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

示例代码

```
List<PicOperationRule> rules = new LinkedList<>();
// 添加一条将图片转化为 png 格式的 rule，处理后的图片在存储桶中的位置标识符为
// examplepngobject
rules.add(new PicOperationRule("examplepngobject", "imageView2/format/png"));
PicOperations picOperations = new PicOperations(true, rules);

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

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

说明

更多完整示例，请前往 GitHub 查看。

基础图片处理

最近更新时间：2024-01-04 15:52:02

简介

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

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

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

缩放

示例代码

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

裁剪

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

示例代码

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

GetObjectRequest getObjectRequest = new GetObjectRequest(bucket, cosPath,
        savePath);
```

```
getObjectRequest.addQuery("imageMogr2/iradius/150", null);

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

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

说明

更多完整示例，请前往 GitHub 查看。

旋转

下面示例展示了如何在对已存储在 COS 的图片在下载时进行处理操作。

示例代码

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

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

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

```
    }

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

说明

更多完整示例，请前往 GitHub 查看。

图片高级压缩

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于图片高级压缩的 API 概览以及 SDK 示例代码。

API	操作描述
图片高级压缩	对指定存储桶下的图片进行压缩

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

图片高级压缩

功能说明

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

示例代码：下载时进行高级压缩

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

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

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

盲水印

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于盲水印的 API 概览以及 SDK 示例代码。

API	操作描述
盲水印	对本地图片添加或提取盲水印并上传至存储桶

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

添加盲水印

功能说明

盲水印支持在上传时添加以及下载时添加。

示例代码一：上传时添加盲水印

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

示例代码二：下载时添加盲水印

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

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

说明

更多完整示例，请前往 GitHub 查看。

设置自定义头部

最近更新时间：2024-01-04 15:52:02

简介

本文档主要介绍 SDK 如何在请求时携带自定义头部。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

功能说明

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

示例代码

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

CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig.Builder()
        .isHttps(true)
        .setRegion(region)
        .setDebuggable(false)
        // 给所有的请求添加公共的自定义 Header
        .addHeader(commonHeaderKey, commonHeaderValue)
        .builder();

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

// 给单个请求添加自定义 Header，优先级比公共 Header 更高
HeadObjectRequest headObjectRequest = new HeadObjectRequest(bucket, cosPath);
try {
```

```
    headObjectRequest.setRequestHeaders(requestHeaderKey, requestHeaderValue, false
} catch (CosXmlClientException e) {
    e.printStackTrace();
}

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

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

说明

更多完整示例，请前往 GitHub 查看。

设置访问域名（CDN/全球加速）

最近更新时间：2024-01-04 15:52:02

简介

本文档提供关于如何使用非默认域名请求 COS 服务。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考。

CDN 默认加速域名

关于如何开启默认加速域名请参考 开启默认 CDN 加速域名。

以下代码展示了如何使用默认加速域名访问 COS 服务。

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

CDN 自定义加速域名

关于如何开启 CDN 自定义加速域名请参考 开启自定义 CDN 加速域名。

以下代码展示了如何使用自定义加速域名访问 COS 服务。

示例代码

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

CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig.Builder()
        .isHttps(true)
        .setRegion(region)
        .setDebuggable(false)
        .setHostFormat(cdnCustomDomain) // 修改请求的域名
        .addHeader("Host", cdnCustomDomain) // 修改 header 中的 host 字段
        .builder();
// 不提供 credentialProvider 类，下载时可以通过给 url 添加 params 参数，来
// 支持 CDN 权限校验
CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig);
```

说明

更多完整示例，请前往 GitHub 查看。

自定义源站域名

关于如何设置自定义源站域名请参考 自定义源站域名。

以下代码展示了如何使用自定义源站域名访问 COS 服务。

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

全球加速域名

关于全球加速功能请参考 全球加速功能概述。

以下代码展示了如何使用全球加速域名访问 COS 服务。

示例代码

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

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

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

说明

更多完整示例，请前往 GitHub 查看。

异常处理

最近更新时间：2024-01-04 15:52:02

简介

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

客户端异常

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

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

服务端异常

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

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

使用自助诊断工具

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

使用步骤

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

C SDK
快速入门

最近更新时间：2024-01-04 17:06:08

下载与安装

相关资源

对象存储的 XML C SDK 源码下载地址：XML C SDK。

演示示例 Demo 下载地址：XML C SDK Demo。

SDK 更新日志请参见 ChangeLog。

SDK 常见问题请参见：C SDK 常见问题。

说明

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

环境依赖

依赖库：libcurl apr apr-util minixml。

安装 SDK

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

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

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

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

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

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

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

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

```
make install
```

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

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

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

```
cmake .
make
make install
```

开始使用

下面为您介绍使用 XML C SDK 的一般流程。

1. 初始化 SDK。

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

APPID 是申请腾讯云账号后，系统分配的账户标识之一。

access_key_id 与 access_key_secret 是账号 API 密钥。

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

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

3. 设置 API 接口必需的参数。

4. 调用 SDK API 发起请求并获得请求响应结果。

初始化

注意

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

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

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

```
        exit(1);
    }

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

    /* 程序结束前，调用 cos_http_io_deinitialize 方法释放之前分配的全局资源 */
    cos_http_io_deinitialize();
    return 0;
}
```

初始化请求选项

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

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

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

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

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

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

/* 用于设置上传请求是否自动添加 Content-MD5 头部，enable 为 COS_FALSE 时上传请求将不自动添加
```

```
cos_set_content_md5_enable(options->ctl, COS_FALSE);

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

说明

临时密钥生成和使用可参见 临时密钥生成及使用指引。

创建存储桶

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

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

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

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

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

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

查询对象列表

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

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

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

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

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

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

上传对象

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

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

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

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

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

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

下载对象

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

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

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

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

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

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

删除对象

```
cos_pool_t *p = NULL;
```

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

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

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

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

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

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

C SDK 常见问题

最近更新时间：2024-01-04 17:06:08

使用 C SDK 如何实现断点续传？

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

使用 C SDK 出现 HttpIOError 错误？

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

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

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

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

存储桶操作

最近更新时间：2024-01-04 17:06:08

简介

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

注意

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

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

基本操作

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

检索存储桶及其权限

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

基本操作

创建存储桶

功能说明

在指定账号下创建一个存储桶。

方法原型

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

参数说明

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

返回结果说明

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

示例

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

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
```

```
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

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

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

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

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

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

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

```
}

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

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

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

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

    test_create_bucket();

    cos_http_io_deinitialize();

    return 0;
}
```

删除存储桶

功能说明

删除指定账号下的空存储桶。

方法原型

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

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String

resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

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

示例

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

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

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

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

```
}

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

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

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

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

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

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

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

    //set log level, default COS_LOG_WARN
```

```
    cos_log_set_level(COS_LOG_WARN);

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

    test_delete_bucket();

    cos_http_io_deinitialize();

    return 0;
}
```

判断存储桶是否存在

功能说明

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

方法原型

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

参数说明

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

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String

error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

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

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

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

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

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

void test_check_bucket_exist()
```

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

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

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

    cos_str_set(&bucket, TEST_BUCKET_NAME);

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

    cos_pool_destroy(pool);
}

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

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

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

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

```
    test_check_bucket_exist();

    cos_http_io_deinitialize();

    return 0;
}
```

对象操作
上传对象

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于对象的上传操作相关的 API 概览以及 SDK 示例代码。

简单操作

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

分块操作

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

高级接口（推荐）

上传对象（断点续传）

功能说明

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

方法原型

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

参数说明

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

返回结果说明

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

示例

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

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_FILE[] = "test.zip";
static char TEST_MULTIPART_OBJECT4[] = "multipart4.dat";

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

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

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

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

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    headers = cos_table_make(p, 0);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_MULTIPART_OBJECT4);
    cos_str_set(&filename, TEST_MULTIPART_FILE);

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

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

    cos_pool_destroy(p);
}

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

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

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

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

    test_resumable_upload_with_multi_threads();

    cos_http_io_deinitialize();

    return 0;
}
```

简单操作

简单上传对象（创建文件夹）

功能说明

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

方法原型

```
cos_status_t *cos_put_object_from_file(const cos_request_options_t *options,
                                       const cos_string_t *bucket,
                                       const cos_string_t *object,
                                       const cos_string_t *filename,
                                       cos_table_t *headers,
                                       cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String

filename	Object 本地保存文件名称	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

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

示例1：上传对象

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

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

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

void init_test_config(cos_config_t *config, int is_cname)
```

```
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

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

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

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_table_t *headers = NULL;
    if (traffic_limit) {
        // 限速值设置范围为819200 - 838860800，即100KB/s - 100MB/s，如果超出该范围将返回40
        headers = cos_table_make(p, 1);
        cos_table_add_int(headers, "x-cos-traffic-limit", 819200);
    }
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&file, TEST_OBJECT_NAME1);
    cos_str_set(&object, TEST_OBJECT_NAME1);
    s = cos_put_object_from_file(options, &bucket, &object, &file, headers, &resp_h
    log_status(s);

    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
```

```
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

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

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

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

    test_put_object_from_file();

    cos_http_io_deinitialize();

    return 0;
}
```

示例2：创建文件夹

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

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

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

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

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

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

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

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

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

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

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

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

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

    test_create_dir();

    cos_http_io_deinitialize();

    return 0;
}
```

示例3：上传文件到虚拟目录

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

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

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

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

```
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

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

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

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

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&file, "examplefile");
    cos_str_set(&object, "folder/exampleobject");
    s = cos_put_object_from_file(options, &bucket, &object, &file, NULL, &resp_head
    if (cos_status_is_ok(s)) {
        printf("put object succeeded\\n");
    } else {
        printf("put object failed\\n");
    }
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
```

```
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

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

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

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

    test_upload_file_to_dir();

    cos_http_io_deinitialize();

    return 0;
}
```

追加上传对象

功能说明

使用分块追加的方式上传对象，追加上传的对象，每个分块大小默认最大为5GB，无最小限制，同时通过追加方式产生的对象大小不得超过5GB。如果 Position 的值和当前对象的长度不致，COS 将返回409错误。如果追加一个 normal 属性的文件，COS 将返回409 ObjectNotAppendable。

方法原型

```
cos_status_t *cos_append_object_from_file(const cos_request_options_t *options,
                                          const cos_string_t *bucket,
                                          const cos_string_t *object,
                                          int64_t position,
                                          const cos_string_t *append_file,
                                          cos_table_t *headers,
                                          cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称	String

	必须为此格式	
object	Object 名称	String
position	追加操作的起始点，单位为字节。首次追加则设置 Position=0，后续追加则设置 Position 为当前 Object 的 content-length	String
append_file	Object 本地保存文件名称	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

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

示例

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

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_OBJECT_NAME3[] = "test3.dat";
static char *TEST_APPEND_NAMES[] = {"test.7z.001", "test.7z.002"};

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

```
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

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

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

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

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

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

    //追加上传对象
    cos_str_set(&object, TEST_OBJECT_NAME3);
    int32_t count = sizeof(TEST_APPEND_NAMES)/sizeof(char*);
    int32_t index = 0;
    int64_t position = 0;
    s = cos_head_object(options, &bucket, &object, NULL, &resp_headers);
    if(s->code == 200) {
        char *content_length_str = (char*)apr_table_get(resp_headers, COS_CONTENT_L
```

```
        if (content_length_str != NULL) {
            position = atol(content_length_str);
        }
    }
    for (; index < count; index++)
    {
        cos_str_set(&file, TEST_APPEND_NAMES[index]);
        s = cos_append_object_from_file(options, &bucket, &object,
                                        position, &file, NULL, &resp_headers);
        log_status(s);

        s = cos_head_object(options, &bucket, &object, NULL, &resp_headers);
        if(s->code == 200) {
            char *content_length_str = (char*)apr_table_get(resp_headers, COS_CONTE
            if (content_length_str != NULL) {
                position = atol(content_length_str);
            }
        }
    }

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

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

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

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

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

    test_append_object();

    cos_http_io_deinitialize();

    return 0;
}
```

分块操作

查询分块上传

功能说明

查询正在进行中的分块上传信息。单次最多列出1000个正在进行中的分块上传。

方法原型

```
cos_status_t *cos_list_multipart_upload(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        cos_list_multipart_upload_params_t *params,
                                        cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
params	List Multipart Uploads 操作参数	Struct
encoding_type	规定返回值的编码方式	String
prefix	前缀匹配，用来规定返回的文件前缀地址	String
delimiter	界符为一个符号： 如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix 如果没有 Prefix，则从路径起点开始	String
max_ret	单次返回最大的条目数量，默认1000	String
key_marker	与 upload-id-marker 一起使用： 当 upload-id-marker 未被指定时，ObjectName 字母顺序大于 key-marker 的条目将被列出 当 upload-id-marker 被指定时，ObjectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出	String
upload_id_marker	与 key-marker 一起使用： 当 key-marker 未被指定时，upload-id-marker 将被忽略	String

	当 key-marker 被指定时，ObjectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出	
truncated	返回条目是否被截断，'true' 或者 'false'	Boolean
next_key_marker	假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点	String
next_upload_id_marker	假如返回条目被截断，则返回 UploadId 就是下一个条目的起点	String
upload_list	分块上传的信息	Struct
key	Object 的名称	String
upload_id	标示本次分块上传的 ID	String
initiated	标示本次分块上传任务的启动时间	String
resp_headers	返回 HTTP 响应消息的头域	Struct

```
typedef struct {
    cos_list_t node;
    cos_string_t key;
    cos_string_t upload_id;
    cos_string_t initiated;
} cos_list_multipart_upload_content_t;
```

返回结果说明

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

示例

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

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
```

```
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_OBJECT[] = "multipart.dat";

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

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

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

void list_multipart()
{
    cos_pool_t *p = NULL;
    cos_string_t bucket;
    cos_string_t object;
    int is_cname = 0;
    cos_table_t *resp_headers = NULL;
    cos_request_options_t *options = NULL;
    cos_status_t *s = NULL;
    cos_list_multipart_upload_params_t *list_multipart_params = NULL;
    cos_list_upload_part_params_t *list_upload_param = NULL;

    cos_pool_create(&p, NULL);
```

```
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    list_multipart_params = cos_create_list_multipart_upload_params(p);
    list_multipart_params->max_ret = 999;
    s = cos_list_multipart_upload(options, &bucket, list_multipart_params, &resp_he
    log_status(s);

    list_upload_param = cos_create_list_upload_part_params(p);
    list_upload_param->max_ret = 1000;
    cos_string_t upload_id;
    cos_str_set(&upload_id,"149373379126aee264fecbf5fe8ddb8b9cd23b76c73ab1af0bcfd50
    cos_str_set(&object, TEST_MULTIPART_OBJECT);
    s = cos_list_upload_part(options, &bucket, &object, &upload_id,
                             list_upload_param, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("List upload part succeeded, upload_id::%.*s\\n",
               upload_id.len, upload_id.data);
        cos_list_part_content_t *part_content = NULL;
        cos_list_for_each_entry(cos_list_part_content_t, part_content, &list_upload
            printf("part_number = %s, size = %s, last_modified = %s, etag = %s\\n",
                   part_content->part_number.data,
                   part_content->size.data,
                   part_content->last_modified.data,
                   part_content->etag.data);
        }
    } else {
        printf("List upload part failed\\n");
    }

    cos_pool_destroy(p);
}

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

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

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

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

    list_multipart();

    cos_http_io_deinitialize();

    return 0;
}
```

初始化分块上传

功能说明

Initiate Multipart Upload 请求实现初始化分片上传，成功执行此请求以后会返回 Upload ID 用于后续的 Upload Part 请求。

方法原型

```
cos_status_t *cos_init_multipart_upload(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        const cos_string_t *object,
                                        cos_string_t *upload_id,
                                        cos_table_t *headers,
                                        cos_table_t **resp_headers);
```

参数说明

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

返回结果说明

返回结果	描述	类型

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

示例

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

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_OBJECT[] = "multipart.dat";

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

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

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
```

```
    options->ctl = cos_http_controller_create(options->pool, 0);
}

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

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

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

    //初始化分块上传
    cos_str_set(&object, TEST_MULTIPART_OBJECT);
    s = cos_init_multipart_upload(options, &bucket, &object,
                                  &upload_id, NULL, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("init multipart upload succeeded\\n");
    } else {
        printf("init multipart upload failed\\n");
    }

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

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

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

    //set log level, default COS_LOG_WARN
```

```
    cos_log_set_level(COS_LOG_WARN);

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

    test_init_multipart_upload();

    cos_http_io_deinitialize();

    return 0;
}
```

上传分块

功能说明

分块上传文件。Upload Part 请求实现在初始化以后的分块上传，支持的块的数量为1 - 10000，块的大小为1MB - 5GB。在每次请求 Upload Part 时，需要携带 partNumber 和 uploadID，partNumber 为块的编号，支持乱序上传。

方法原型

```
cos_status_t *cos_upload_part_from_file(const cos_request_options_t *options,
                                        const cos_string_t *bucket,
                                        const cos_string_t *object,
                                        const cos_string_t *upload_id,
                                        int part_num,
                                        cos_upload_file_t *upload_file,
                                        cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
upload_id	上传任务编号	String
part_num	分块编号	Int
upload_file	待上传本地文件信息	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

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

示例

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

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_OBJECT[] = "multipart.dat";
static char TEST_MULTIPART_FILE[] = "test.zip";

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

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

```
}

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

void test_upload_part()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t upload_id = cos_string("xxxxxxxxxxxxxxxxxx");  // 替换您自己的uploa
    cos_table_t *resp_headers = NULL;
    int part_num = 1;
    int64_t pos = 0;
    int64_t file_length = 0;

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

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

    //上传分块
    int res = COSE_OK;
    cos_upload_file_t *upload_file = NULL;
    cos_file_buf_t *fb = cos_create_file_buf(p);
    res = cos_open_file_for_all_read(p, TEST_MULTIPART_FILE, fb);
    if (res != COSE_OK) {
        cos_error_log("Open read file fail, filename:%s\\n", TEST_MULTIPART_FILE);
        return;
    }
    file_length = fb->file_last;
    apr_file_close(fb->file);
    while(pos < file_length) {
        upload_file = cos_create_upload_file(p);
        cos_str_set(&upload_file->filename, TEST_MULTIPART_FILE);
        upload_file->file_pos = pos;
```

```
        pos += 2 * 1024 * 1024;
        upload_file->file_last = pos < file_length ? pos : file_length; //2MB
        s = cos_upload_part_from_file(options, &bucket, &object, &upload_id,
                                      part_num++, upload_file, &resp_headers);

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

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

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

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

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

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

    test_upload_part();

    cos_http_io_deinitialize();

    return 0;
}
```

查询已上传块

功能说明

查询特定分块上传操作中的已上传的块。

方法原型

```
cos_status_t *cos_list_upload_part(const cos_request_options_t *options,
                                   const cos_string_t *bucket,
                                   const cos_string_t *object,
                                   const cos_string_t *upload_id,
                                   cos_list_upload_part_params_t *params,
                                   cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
upload_id	上传任务编号	String
params	List Parts 操作参数	Struct
part_number_marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
encoding_type	规定返回值的编码方式	String
max_ret	单次返回最大的条目数量，默认1000	String
truncated	返回条目是否被截断，'true' 或者 'false'	Boolean
next_part_number_marker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
part_list	完成分块的信息	Struct
part_number	分块编号	String
size	分块大小，单位 Byte	String
etag	分块的 SHA-1 算法校验值	String
last_modified	分块最后修改时间	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型

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

示例

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

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_OBJECT[] = "multipart.dat";

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

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

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
```

```
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_list_upload_part()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_list_upload_part_params_t *params = NULL;
    cos_list_t complete_part_list;
    cos_string_t upload_id = cos_string("xxxxxxxxxxxxxxxxxx");  // 替换您自己的uploa
    cos_table_t *resp_headers = NULL;

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

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

    //查询已上传块
    params = cos_create_list_upload_part_params(p);
    params->max_ret = 1000;
    cos_list_init(&complete_part_list);
    s = cos_list_upload_part(options, &bucket, &object, &upload_id,
                             params, &resp_headers);

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

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

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

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

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

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

    test_list_upload_part();

    cos_http_io_deinitialize();

    return 0;
}
```

完成分块上传

功能说明

完成整个文件的分块上传。当您已经使用 Upload Parts 上传所有块以后，您可以用该 API 完成上传。在使用该 API 时，您必须在 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。

方法原型

```
cos_status_t *cos_complete_multipart_upload(const cos_request_options_t *options,
                                            const cos_string_t *bucket,
                                            const cos_string_t *object,
                                            const cos_string_t *upload_id,
                                            cos_list_t *part_list,
                                            cos_table_t *headers,
                                            cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String

upload_id	上传任务编号	String
part_list	完成分块上传的参数	Struct
part_number	分块编号	String
etag	分块的 ETag 值，为 sha1 校验值，需要在校验值前后加上双引号，如 "3a0f1fd698c235af9cf098cb74aa25bc"	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

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

示例

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

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_OBJECT[] = "multipart.dat";

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

```
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

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

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

void test_complete_multipart_upload()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_list_upload_part_params_t *params = NULL;
    cos_list_t complete_part_list;
    cos_string_t upload_id = cos_string("xxxxxxxxxxxxxxxxxx");  // 替换您自己的uploa
    cos_table_t *resp_headers = NULL;
    cos_list_part_content_t *part_content = NULL;
    cos_complete_part_content_t *complete_part_content = NULL;

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

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

    //查询已上传分块
    params = cos_create_list_upload_part_params(p);
    params->max_ret = 1000;
    cos_list_init(&complete_part_list);
```

```
    s = cos_list_upload_part(options, &bucket, &object, &upload_id,
                             params, &resp_headers);

    if (cos_status_is_ok(s)) {
        printf("List multipart succeeded\\n");
    } else {
        printf("List multipart failed\\n");
        cos_pool_destroy(p);
        return;
    }

    cos_list_for_each_entry(cos_list_part_content_t, part_content, &params->part_li
        complete_part_content = cos_create_complete_part_content(p);
        cos_str_set(&complete_part_content->part_number, part_content->part_number.
        cos_str_set(&complete_part_content->etag, part_content->etag.data);
        cos_list_add_tail(&complete_part_content->node, &complete_part_list);
    }

    //完成分块上传
    s = cos_complete_multipart_upload(options, &bucket, &object, &upload_id,
                                      &complete_part_list, NULL, &resp_headers);

    if (cos_status_is_ok(s)) {
        printf("Complete multipart upload from file succeeded, upload_id:%.*s\\n",
            upload_id.len, upload_id.data);
    } else {
        printf("Complete multipart upload from file failed\\n");
    }

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

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

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

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

    //set log output, default stderr
```

```
    cos_log_set_output(NULL);

    test_complete_multipart_upload();

    cos_http_io_deinitialize();

    return 0;
}
```

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块。当您调用 Abort Multipart Upload 时，如果有正在使用这个 Upload Parts 上传块的请求，则 Upload Parts 会返回失败。

方法原型

```
cos_status_t *cos_abort_multipart_upload(const cos_request_options_t *options,
                                         const cos_string_t *bucket,
                                         const cos_string_t *object,
                                         cos_string_t *upload_id,
                                         cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
upload_id	上传任务编号	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String

error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

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

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_MULTIPART_OBJECT[] = "multipart.dat";

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

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

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

void abort_multipart_upload()
```

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

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    headers = cos_table_make(p, 1);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_MULTIPART_OBJECT);

    s = cos_init_multipart_upload(options, &bucket, &object,
                                  &upload_id, headers, &resp_headers);

    if (cos_status_is_ok(s)) {
        printf("Init multipart upload succeeded, upload_id:%.*s\\n",
               upload_id.len, upload_id.data);
    } else {
        printf("Init multipart upload failed\\n");
        cos_pool_destroy(p);
        return;
    }

    s = cos_abort_multipart_upload(options, &bucket, &object, &upload_id,
                                   &resp_headers);

    if (cos_status_is_ok(s)) {
        printf("Abort multipart upload succeeded, upload_id::%.*s\\n",
               upload_id.len, upload_id.data);
    } else {
        printf("Abort multipart upload failed\\n");
    }

    cos_pool_destroy(p);
}

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

```
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

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

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

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

    abort_multipart_upload();

    cos_http_io_deinitialize();

    return 0;
}
```

复制与移动对象

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于对象的复制与移动 API 概览以及 SDK 示例代码。

简单操作

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

分块操作

API	操作名	操作描述
Upload Part - Copy	复制分块	将其他对象复制为一个分块

简单复制

设置对象复制（修改属性）

功能说明

复制文件到目标路径。该接口还可以修改对象属性，例如修改对象的存储类型。

方法原型

```
cos_status_t *cos_copy_object(const cos_request_options_t *options,
                              const cos_string_t *src_bucket,
                              const cos_string_t *src_object,
                              const cos_string_t *src_endpoint,
                              const cos_string_t *dest_bucket,
                              const cos_string_t *dest_object,
                              cos_table_t *headers,
                              cos_copy_object_params_t *copy_object_param,
                              cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
src_bucket	源存储桶	String
src_object	源对象名称	String
src_endpoint	源对象访问域名信息	String
dest_bucket	目的存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
dest_object	目的 Object 名称	String
headers	COS 请求附加头域	Struct
copy_object_param	Put Object Copy 操作参数	Struct
etag	返回文件的 MD5 算法校验值	String
last_modify	返回文件最后修改时间，GMT 格式	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

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

示例1：普通对象复制

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

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

```
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";
static char TEST_OBJECT_NAME2[] = "test2.dat";

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

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

void test_copy()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t src_bucket;
    cos_string_t src_object;
    cos_string_t src_endpoint;
    cos_table_t *resp_headers = NULL;

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

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

    //设置对象复制
```

```
    cos_str_set(&object, TEST_OBJECT_NAME2);
    cos_str_set(&src_bucket, TEST_BUCKET_NAME);
    cos_str_set(&src_endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&src_object, TEST_OBJECT_NAME1);

    cos_copy_object_params_t *params = NULL;
    params = cos_create_copy_object_params(p);
    s = cos_copy_object(options, &src_bucket, &src_object, &src_endpoint, &bucket,
    if (cos_status_is_ok(s)) {
        printf("put object copy succeeded\\n");
    } else {
        printf("put object copy failed\\n");
    }

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

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

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

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

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

    test_copy();

    cos_http_io_deinitialize();

    return 0;
}
```

示例2：修改存储类型

注意

标准存储可以修改为低频存储、智能分层存储、归档存储和深度归档存储等，如果希望将归档存储或深度归档存储的对象修改为其他存储类型，首先需要使用 cos_post_object_restore() 将归档存储或深度归档存储的对象回热，才能

使用该接口请求修改存储类型；关于存储类型的详细说明请参见 存储类型概述。

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

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

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

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

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

void test_modify_storage_class()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
```

```
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t src_bucket;
    cos_string_t src_object;
    cos_string_t src_endpoint;
    cos_table_t *resp_headers = NULL;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_OBJECT_NAME1);
    cos_str_set(&src_bucket, TEST_BUCKET_NAME);
    cos_str_set(&src_endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&src_object, TEST_OBJECT_NAME1);

    // 设置x-cos-metadata-directive和x-cos-storage-class头域(替换为自己要更改的存储类型)
    cos_table_t *headers = cos_table_make(p, 2);
    apr_table_add(headers, "x-cos-metadata-directive", "Replaced");
    // 存储类型包括NTELLIGENT_TIERING, MAZ_INTELLIGENT_TIERING, STANDARD_IA, ARCHIVE, I
    apr_table_add(headers, "x-cos-storage-class", "ARCHIVE");

    cos_copy_object_params_t *params = NULL;
    params = cos_create_copy_object_params(p);
    s = cos_copy_object(options, &src_bucket, &src_object, &src_endpoint, &bucket,
    log_status(s);

    cos_pool_destroy(p);
}

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

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

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

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

```
    test_modify_storage_class();

    cos_http_io_deinitialize();

    return 0;
}
```

分块复制

复制分块

功能说明

将其他对象复制为一个分块。

方法原型

```
cos_status_t *cos_upload_part_copy(const cos_request_options_t *options,
                                   cos_upload_part_copy_params_t *params,
                                   cos_table_t *headers,
                                   cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
params	复制分块参数信息	Struct
copy_source	源文件路径	String
dest_bucket	目的 存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
dest_object	目的 Object 名称	String
upload_id	上传任务编号	String
part_num	分块编号	Int
range_start	源文件起始偏移	Int
range_end	源文件终止偏移	Int

rsp_content	复制分块结果信息	Struct
etag	返回文件的 MD5 算法校验值	String
last_modify	返回文件最后修改时间，GMT 格式	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

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

示例

```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"
#include <sys/stat.h>

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

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

void init_test_config(cos_config_t *config, int is_cname)
```

```
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

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

void make_rand_string(cos_pool_t *p, int len, cos_string_t *data)
{
    char *str = NULL;
    int i = 0;
    str = (char *)cos_palloc(p, len + 1);
    for ( ; i < len; i++) {
        str[i] = 'a' + rand() % 32;
    }
    str[len] = '\\0';
    cos_str_set(data, str);
}

unsigned long get_file_size(const char *file_path)
{
    unsigned long filesize = -1;
    struct stat statbuff;

    if(stat(file_path, &statbuff) < 0){
        return filesize;
    } else {
        filesize = statbuff.st_size;
    }

    return filesize;
}

void test_part_copy()
{
    cos_pool_t *p = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
```

```
    cos_string_t file;
    int is_cname = 0;
    cos_string_t upload_id;
    cos_list_upload_part_params_t *list_upload_part_params = NULL;
    cos_upload_part_copy_params_t *upload_part_copy_params1 = NULL;
    cos_upload_part_copy_params_t *upload_part_copy_params2 = NULL;
    cos_table_t *headers = NULL;
    cos_table_t *query_params = NULL;
    cos_table_t *resp_headers = NULL;
    cos_table_t *list_part_resp_headers = NULL;
    cos_list_t complete_part_list;
    cos_list_part_content_t *part_content = NULL;
    cos_complete_part_content_t *complete_content = NULL;
    cos_table_t *complete_resp_headers = NULL;
    cos_status_t *s = NULL;
    int part1 = 1;
    int part2 = 2;
    char *local_filename = "test_upload_part_copy.file";
    char *download_filename = "test_upload_part_copy.file.download";
    char *source_object_name = "cos_test_upload_part_copy_source_object";
    char *dest_object_name = "cos_test_upload_part_copy_dest_object";
    FILE *fd = NULL;
    cos_string_t download_file;
    cos_string_t dest_bucket;
    cos_string_t dest_object;
    int64_t range_start1 = 0;
    int64_t range_end1 = 6000000;
    int64_t range_start2 = 6000001;
    int64_t range_end2;
    cos_string_t data;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);

    // create multipart upload local file
    make_rand_string(p, 10 * 1024 * 1024, &data);
    fd = fopen(local_filename, "w");
    fwrite(data.data, sizeof(data.data[0]), data.len, fd);
    fclose(fd);

    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, source_object_name);
    cos_str_set(&file, local_filename);
    s = cos_put_object_from_file(options, &bucket, &object, &file, NULL, &resp_head
    log_status(s);
```

```
    //init mulitipart
    cos_str_set(&object, dest_object_name);
    s = cos_init_multipart_upload(options, &bucket, &object,
                                  &upload_id, NULL, &resp_headers);
    log_status(s);

    //upload part copy 1
    upload_part_copy_params1 = cos_create_upload_part_copy_params(p);
    cos_str_set(&upload_part_copy_params1->copy_source, "bucket-appid.cn-south.myqc
    cos_str_set(&upload_part_copy_params1->dest_bucket, TEST_BUCKET_NAME);
    cos_str_set(&upload_part_copy_params1->dest_object, dest_object_name);
    cos_str_set(&upload_part_copy_params1->upload_id, upload_id.data);
    upload_part_copy_params1->part_num = part1;
    upload_part_copy_params1->range_start = range_start1;
    upload_part_copy_params1->range_end = range_end1;
    headers = cos_table_make(p, 0);
    s = cos_upload_part_copy(options, upload_part_copy_params1, headers, &resp_head
    log_status(s);
    printf("last modified:%s, etag:%s\\n", upload_part_copy_params1->rsp_content->l

    //upload part copy 2
    resp_headers = NULL;
    range_end2 = get_file_size(local_filename) - 1;
    upload_part_copy_params2 = cos_create_upload_part_copy_params(p);
    cos_str_set(&upload_part_copy_params2->copy_source, "bucket-appid.cn-south.myqc
    cos_str_set(&upload_part_copy_params2->dest_bucket, TEST_BUCKET_NAME);
    cos_str_set(&upload_part_copy_params2->dest_object, dest_object_name);
    cos_str_set(&upload_part_copy_params2->upload_id, upload_id.data);
    upload_part_copy_params2->part_num = part2;
    upload_part_copy_params2->range_start = range_start2;
    upload_part_copy_params2->range_end = range_end2;
    headers = cos_table_make(p, 0);
    s = cos_upload_part_copy(options, upload_part_copy_params2, headers, &resp_head
    log_status(s);
    printf("last modified:%s, etag:%s\\n", upload_part_copy_params1->rsp_content->l

    //list part
    list_upload_part_params = cos_create_list_upload_part_params(p);
    list_upload_part_params->max_ret = 10;
    cos_list_init(&complete_part_list);

    cos_str_set(&dest_bucket, TEST_BUCKET_NAME);
    cos_str_set(&dest_object, dest_object_name);
    s = cos_list_upload_part(options, &dest_bucket, &dest_object, &upload_id,
                             list_upload_part_params, &list_part_resp_headers);
    log_status(s);
    cos_list_for_each_entry(cos_list_part_content_t, part_content, &list_upload_par
```

```
        complete_content = cos_create_complete_part_content(p);
        cos_str_set(&complete_content->part_number, part_content->part_number.data)
        cos_str_set(&complete_content->etag, part_content->etag.data);
        cos_list_add_tail(&complete_content->node, &complete_part_list);
    }

    //complete multipart
    headers = cos_table_make(p, 0);
    s = cos_complete_multipart_upload(options, &dest_bucket, &dest_object,
            &upload_id, &complete_part_list, headers, &complete_resp_headers);
    log_status(s);

    //check upload copy part content equal to local file
    headers = cos_table_make(p, 0);
    cos_str_set(&download_file, download_filename);
    s = cos_get_object_to_file(options, &dest_bucket, &dest_object, headers,
                               query_params, &download_file, &resp_headers);
    log_status(s);
    printf("local file len = %"APR_INT64_T_FMT", download file len = %"APR_INT64_T_
    remove(download_filename);
    remove(local_filename);
    cos_pool_destroy(p);

    printf("test part copy ok\\n");
}

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

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

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

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

    test_part_copy();

    cos_http_io_deinitialize();

    return 0;
```

```
}
```

移动对象

移动对象

功能说明

移动对象主要包括两个操作：复制源对象到目标位置，删除源对象。
由于 COS 通过存储桶名称（Bucket）和对象键（ObjectKey）来标识对象。移动对象也就意味着修改这个对象的标识，COS C SDK 目前没有提供修改对象唯一标识名的单独接口，但是可以通过组合**复制对象**加上**删除对象**的基本操作，来达到修改对象标识的目的，从而实现移动对象。
复制对象方法详细说明请参见 设置对象复制
删除对象方法详细说明请参见：删除对象

示例

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

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";
static char TEST_OBJECT_NAME2[] = "test2.dat";

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

void init_test_config(cos_config_t *config, int is_cname)
{
```

```
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

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

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

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

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

    //设置对象复制
    cos_str_set(&object, TEST_OBJECT_NAME1);
    cos_str_set(&src_endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&src_object, TEST_OBJECT_NAME2);

    cos_copy_object_params_t *params = NULL;
    params = cos_create_copy_object_params(p);
    s = cos_copy_object(options, &bucket, &src_object, &src_endpoint, &bucket, &obj
    log_status(s);
    if (cos_status_is_ok(s)) {
        s = cos_delete_object(options, &bucket, &src_object, &resp_headers);
        log_status(s);
        printf("move object succeeded\\n");
```

```
    } else {
        printf("move object failed\\n");
    }

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

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

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

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

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

    test_move();

    cos_http_io_deinitialize();

    return 0;
}
```

下载对象

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于对象的下载操作相关的 API 概览以及 SDK 示例代码。

简单操作

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

高级接口（推荐）

下载对象（断点续传）

功能说明

分块下载接口根据用户对象的长度，自动使用 Range 下载数据，实现并发下载，分块大小默认1048576（1MB），可通过 part_size 参数调整。

方法原型

```
cos_status_t *cos_resumable_download_file(cos_request_options_t *options,
                                          cos_string_t *bucket,
                                          cos_string_t *object,
                                          cos_string_t *filepath,
                                          cos_table_t *headers,
                                          cos_table_t *params,
                                          cos_resumable_clt_params_t *clt_params,
                                          cos_progress_callback progress_callback);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String

object	Object 名称	String
filepath	Object 本地文件名称	String
headers	COS 请求附加头域	Struct
params	COS 请求操作参数	Struct
clt_params	下载对象控制参数	Struct
part_size	块大小，单位为 bytes，如果用户指定的 part_size 小于4194304（4MB），则按4194304（4MB）处理	Int
thread_num	线程池大小，默认为1	Int
enable_checkpoint	是否使能断点续传	Int
checkpoint_path	当使能断点续传时，表示保存上传进度的文件路径，默认路径为 `<filepath>.cp` ，其中 filepath 为 Object 本地文件名称	String
progress_callback	下载进度回调函数	Function

结果说明

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

示例

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

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

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
static char TEST_DOWNLOAD_NAME4[] = "multipart_download.dat";
static char TEST_MULTIPART_OBJECT4[] = "multipart4.dat";

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

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

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

void test_resumable()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t filepath;
    cos_resumable_clt_params_t *clt_params;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
    cos_str_set(&object, TEST_MULTIPART_OBJECT4);
    cos_str_set(&filepath, TEST_DOWNLOAD_NAME4);
```

```
    clt_params = cos_create_resumable_clt_params_content(p, 5*1024*1024, 3, COS_FAL
    s = cos_resumable_download_file(options, &bucket, &object, &filepath, NULL, NUL
    log_status(s);

    cos_pool_destroy(p);
}

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

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

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_resumable();

    cos_http_io_deinitialize();

    return 0;
}
```

批量下载（从 COS 下载目录）

功能说明

将 COS 目录及其文件下载到本地磁盘。

示例

```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
```

```
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_download_directory()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t file_name;
    cos_string_t suffix = cos_string("/");
    cos_table_t *resp_headers;
    cos_table_t *headers = NULL;
    cos_table_t *params = NULL;
    int is_truncated = 1;
    cos_string_t marker;
    apr_status_t status;

    //初始化请求选项
    cos_pool_create(&p, NULL);
```

```
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //list object (get bucket)
    cos_list_object_params_t *list_params = NULL;
    list_params = cos_create_list_object_params(p);
    cos_str_set(&list_params->prefix, "folder/");    //替换为您自己的目录名称
    cos_str_set(&marker, "");
    while (is_truncated) {
        list_params->marker = marker;
        s = cos_list_object(options, &bucket, list_params, &resp_headers);
        log_status(s);
        if (!cos_status_is_ok(s)) {
            printf("list object failed, req_id:%s\\n", s->req_id);
            break;
        }
        cos_list_object_content_t *content = NULL;
        cos_list_for_each_entry(cos_list_object_content_t, content, &list_params->o
            cos_str_set(&file_name, content->key.data);
            if (cos_ends_with(&content->key, &suffix)) {
                //如果是目录需要先创建，0x0755权限可以自己按需修改，参考apr_file_info.h中定
                status = apr_dir_make(content->key.data, 0x0755, options->pool);
                if (status != APR_SUCCESS && !APR_STATUS_IS_EEXIST(status)) {
                    printf("mkdir: %s failed, status: %d\\n", content->key.data, st
                }
            } else {
                //下载对象到本地目录，这里默认下载在程序运行的当前目录
                s = cos_get_object_to_file(options, &bucket, &content->key, headers
                if (!cos_status_is_ok(s)) {
                    printf("get object[%s] failed, req_id:%s\\n", content->key.data
                }
            }
        }
        is_truncated = list_params->truncated;
        marker = list_params->next_marker;
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");
```

```
    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
        exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_download_directory();

    cos_http_io_deinitialize();

    return 0;
}
```

简单操作

下载对象

功能说明

下载一个对象至本地。该操作需要对目标对象具有读权限或该目标对象已对所有人都开放了读权限（公有读）。

方法原型

```
cos_status_t *cos_get_object_to_file(const cos_request_options_t *options,
                                     const cos_string_t *bucket,
                                     const cos_string_t *object,
                                     cos_table_t *headers,
                                     cos_table_t *params,
                                     cos_string_t *filename,
                                     cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String

object	Object 名称	String
headers	COS 请求附加头域	Struct
params	COS 请求操作参数	Struct
filename	Object 本地保存文件名称	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例：普通下载

```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";
static char TEST_DOWNLOAD_NAME[] = "download_test3.dat";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
```

```
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_download_object()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_string_t file;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //获取对象
    cos_str_set(&file, TEST_DOWNLOAD_NAME);
    cos_str_set(&object, TEST_OBJECT_NAME1);
    s = cos_get_object_to_file(options, &bucket, &object, NULL, NULL, &file, &resp_
    if (cos_status_is_ok(s)) {
        printf("get object succeeded\\n");
    } else {
        printf("get object failed\\n");
    }
```

```
    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_download_object();

    cos_http_io_deinitialize();

    return 0;
}
```

列出对象

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于对象列举相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
GET Bucket（List Objects）	查询对象列表	查询存储桶下的部分或者全部对象

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

方法原型

```
cos_status_t *cos_list_object(const cos_request_options_t *options,
                              const cos_string_t *bucket,
                              cos_list_object_params_t *params,
                              cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，Bucket 的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
params	列表操作参数信息	Struct
encoding_type	规定返回值的编码方式	String
prefix	前缀匹配，用来规定返回的文件前缀地址	String
marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
delimiter	查询分隔符，用于对对象键进行分组	String

max_ret	单次返回最大的条目数量，默认1000	Struct
truncated	返回条目是否被截断，'true' 或者 'false'	Boolean
next_marker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
object_list	Get Bucket 操作返回的对象信息列表	Struct
key	Get Bucket 操作返回的 Object 名称	Struct
last_modified	Get Bucket 操作返回的 Object 最后修改时间	Struct
etag	Get Bucket 操作返回的对象的 SHA-1 算法校验值	Struct
size	Get Bucket 操作返回的对象大小，单位 Byte	Struct
owner_id	Get Bucket 操作返回的对象拥有者 UID 信息	Struct
storage_class	Get Bucket 操作返回的对象存储级别	Struct
common_prefix_list	将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例1：列出第一页对象

```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
```

```
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_list_objects()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //获取对象列表
    cos_list_object_params_t *list_params = NULL;
    cos_list_object_content_t *content = NULL;
    list_params = cos_create_list_object_params(p);
    s = cos_list_object(options, &bucket, list_params, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("list object succeeded\\n");
        cos_list_for_each_entry(cos_list_object_content_t, content, &list_params->o
           printf("object: %.*s\\n", content->key.len, content->key.data);
```

```
        }
    } else {
        printf("list object failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_list_objects();

    cos_http_io_deinitialize();

    return 0;
}
```

示例2：列出目录下的对象

对象存储中本身是没有文件夹和目录的概念的，为了满足用户使用习惯，用户可通过分隔符/来模拟“文件夹”。

```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
```

```
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_list_directory()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers;
    int is_truncated = 1;
    cos_string_t marker;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //list object (get bucket)
    cos_list_object_params_t *list_params = NULL;
    list_params = cos_create_list_object_params(p);
    // prefix表示列出的object的key以prefix开始
    cos_str_set(&list_params->prefix, "folder/");
    // deliter表示分隔符, 设置为/表示列出当前目录下的object, 设置为空表示列出所有的object
    cos_str_set(&list_params->delimiter, "/");
    // 设置最大遍历出多少个对象, 一次listobject最大支持1000
    list_params->max_ret = 1000;
    cos_str_set(&marker, "");
    while (is_truncated) {
        list_params->marker = marker;
```

```
        s = cos_list_object(options, &bucket, list_params, &resp_headers);
        if (!cos_status_is_ok(s)) {
            printf("list object failed, req_id:%s\\n", s->req_id);
            break;
        }
        // list_params->object_list 返回列出的object对象。
        cos_list_object_content_t *content = NULL;
        cos_list_for_each_entry(cos_list_object_content_t, content, &list_params->o
            printf("object: %s\\n", content->key.data);
        }
        // list_params->common_prefix_list 表示被delimiter截断的路径, 如delimter设置为/
        cos_list_object_common_prefix_t *common_prefix = NULL;
        cos_list_for_each_entry(cos_list_object_common_prefix_t, common_prefix, &li
            printf("common prefix: %s\\n", common_prefix->prefix.data);
        }

        is_truncated = list_params->truncated;
        marker = list_params->next_marker;
    }
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_list_directory();

    cos_http_io_deinitialize();

    return 0;
}
```

示例3：列出桶中的所有对象

一次列举的最大对象数为1000，当桶内对象个数超过1000后，需要去循环列举桶中所有的对象。

```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_list_all_objects()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers;
    int is_truncated = 1;
    cos_string_t marker;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);
```

```
    //list object (get bucket)
    cos_list_object_params_t *list_params = NULL;
    list_params = cos_create_list_object_params(p);
    // 设置最大遍历出多少个对象, 一次listobject最大支持1000
    list_params->max_ret = 1000;
    cos_str_set(&marker, "");
    while (is_truncated) {
        list_params->marker = marker;
        cos_list_init(&list_params->object_list);
        s = cos_list_object(options, &bucket, list_params, &resp_headers);
        if (!cos_status_is_ok(s)) {
            printf("list object failed, req_id:%s\\n", s->req_id);
            break;
        }
        // list_params->object_list 返回列出的object对象。
        cos_list_object_content_t *content = NULL;
        cos_list_for_each_entry(cos_list_object_content_t, content, &list_params->o
            printf("object: %s\\n", content->key.data);
        }

        is_truncated = list_params->truncated;
        marker = list_params->next_marker;
    }
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_list_all_objects();

    cos_http_io_deinitialize();
```

```
    return 0;
}
```

删除对象

最近更新时间：2024-01-04 17:06:08

简介

本文档提供关于对象删除操作相关的 API 概览以及 SDK 示例代码。

API	操作名	操作描述
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

删除单个对象

功能说明

在存储桶中删除指定对象。

方法原型

```
cos_status_t *cos_delete_object(const cos_request_options_t *options,
                                const cos_string_t *bucket,
                                const cos_string_t *object,
                                cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型

code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例1：删除对象

```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME1[] = "1.txt";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_object()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
```

```
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_table_t *resp_headers = NULL;

    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //删除单个对象
    cos_str_set(&object, TEST_OBJECT_NAME1);
    s = cos_delete_object(options, &bucket, &object, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("delete object succeeded\\n");
    } else {
        printf("delete object failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_delete_object();

    cos_http_io_deinitialize();
```

```
    return 0;
}
```

示例2：删除文件夹

该请求不会删除文件夹内的对象，只会删除指定的 key。

```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_dir()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_string_t object;
    cos_table_t *resp_headers = NULL;
```

```
    //创建内存池
    cos_pool_create(&p, NULL);

    //初始化请求选项
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    //删除目录对象
    cos_str_set(&object, "folder/");
    s = cos_delete_object(options, &bucket, &object, &resp_headers);
    if (cos_status_is_ok(s)) {
        printf("delete object succeeded\\n");
    } else {
        printf("delete object failed\\n");
    }

    //销毁内存池
    cos_pool_destroy(p);
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");

    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
       exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_delete_dir();

    cos_http_io_deinitialize();

    return 0;
}
```

删除多个对象

功能说明

在存储桶中批量删除对象，最大支持单次删除1000个对象。对于返回结果，COS 提供 Verbose 和 Quiet 两种结果模式。Verbose 模式将返回每个 Object 的删除结果。Quiet 模式只返回报错的 Object 信息。

方法原型

```
cos_status_t *cos_delete_objects(const cos_request_options_t *options,
                                 const cos_string_t *bucket,
                                 cos_list_t *object_list,
                                 int is_quiet,
                                 cos_table_t **resp_headers,
                                 cos_list_t *deleted_object_list);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object_list	Object 待删除列表	Struct
key	待删除 Object 名称	String
is_quiet	决定是否启动 Quiet 模式： True(1)：启动 Quiet 模式，False(0)：启动 Verbose 模式。默认为 False(0)	Boolean
resp_headers	返回 HTTP 响应消息的头域	Struct
deleted_object_list	Object 删除信息列表	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String

req_id	请求消息 ID	String

示例1：删除多个对象

```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://cloud.tencent.com/document/product/
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.c
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";
// 对象键，对象（Object）在存储桶（Bucket）中的唯一标识。有关对象与对象键的进一步说明，请参见 ht
static char TEST_OBJECT_NAME2[] = "test2.dat";
static char TEST_OBJECT_NAME3[] = "test3.dat";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}

void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_objects()
```

```
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_string_t bucket;
    cos_status_t *s = NULL;
    cos_table_t *resp_headers = NULL;
    cos_request_options_t *options = NULL;
    char *object_name1 = TEST_OBJECT_NAME2;
    char *object_name2 = TEST_OBJECT_NAME3;
    cos_object_key_t *content1 = NULL;
    cos_object_key_t *content2 = NULL;
    cos_list_t object_list;
    cos_list_t deleted_object_list;
    int is_quiet = COS_TRUE;

    cos_pool_create(&p, NULL);
    options = cos_request_options_create(p);
    init_test_request_options(options, is_cname);
    cos_str_set(&bucket, TEST_BUCKET_NAME);

    cos_list_init(&object_list);
    cos_list_init(&deleted_object_list);
    content1 = cos_create_cos_object_key(p);
    cos_str_set(&content1->key, object_name1);
    cos_list_add_tail(&content1->node, &object_list);
    content2 = cos_create_cos_object_key(p);
    cos_str_set(&content2->key, object_name2);
    cos_list_add_tail(&content2->node, &object_list);

    s = cos_delete_objects(options, &bucket, &object_list, is_quiet,
        &resp_headers, &deleted_object_list);
    log_status(s);

    cos_pool_destroy(p);

    if (cos_status_is_ok(s)) {
        printf("delete objects succeeded\\n");
    } else {
        printf("delete objects failed\\n");
    }
}

int main(int argc, char *argv[])
{
    // 通过环境变量获取 SECRETID 和 SECRETKEY
    TEST_ACCESS_KEY_ID     = getenv("COS_SECRETID");
    TEST_ACCESS_KEY_SECRET = getenv("COS_SECRETKEY");
```

```
    if (cos_http_io_initialize(NULL, 0) != COSE_OK) {
        exit(1);
    }

    //set log level, default COS_LOG_WARN
    cos_log_set_level(COS_LOG_WARN);

    //set log output, default stderr
    cos_log_set_output(NULL);

    test_delete_objects();

    cos_http_io_deinitialize();

    return 0;
}
```

示例2：删除文件夹及其文件

对象存储中本身是没有文件夹和目录的概念的，为了满足用户使用习惯，用户可通过分隔符/来模拟“文件夹”。
删除文件夹及其文件这一场景，实际在 COS 上相当于删除一批有着同样前缀的对象。目前 COS C SDK 没有提供一个接口去实现这样的操作，但是可以通过组合**查询对象列表**加上**批量删除对象**的基本操作，达到删除文件夹及其文件的效果。

```
#include "cos_http_io.h"
#include "cos_api.h"
#include "cos_log.h"

// endpoint 是 COS 访问域名信息，详情请参见 https://intl.cloud.tencent.com/document/proc
static char TEST_COS_ENDPOINT[] = "cos.ap-guangzhou.myqcloud.com";
// 开发者拥有的项目身份ID/密钥，可在 https://console.intl.cloud.tencent.com/cam/capi 页面
static char *TEST_ACCESS_KEY_ID;                //your secret_id
static char *TEST_ACCESS_KEY_SECRET;            //your secret_key
// 开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 https://console.intl.cl
static char TEST_APPID[] = "<APPID>";    //your appid
//the cos bucket name, syntax: [bucket]-[appid], for example: mybucket-1253666666,
static char TEST_BUCKET_NAME[] = "<bucketname-appid>";

void log_status(cos_status_t *s)
{
    cos_warn_log("status->code: %d", s->code);
    if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}
```

```
void init_test_config(cos_config_t *config, int is_cname)
{
    cos_str_set(&config->endpoint, TEST_COS_ENDPOINT);
    cos_str_set(&config->access_key_id, TEST_ACCESS_KEY_ID);
    cos_str_set(&config->access_key_secret, TEST_ACCESS_KEY_SECRET);
    cos_str_set(&config->appid, TEST_APPID);
    config->is_cname = is_cname;
}

void init_test_request_options(cos_request_options_t *options, int is_cname)
{
    options->config = cos_config_create(options->pool);
    init_test_config(options->config, is_cname);
    options->ctl = cos_http_controller_create(options->pool, 0);
}

void test_delete_directory()
{
    cos_pool_t *p = NULL;
    int is_cname = 0;
    cos_status_t *s = NULL;
    cos_request_options_t *options = NULL;
    cos_string_t bucket;
    cos_table_t *resp_headers;
    int is_truncated = 1
```