对象存储 OSS

SDK 参考

SDK 参考

SDK 文档简介

使用OSS SDK之前，您需要：

- 了解并开通阿里云OSS服务。
- 创建AccessKey。

本文档详细介绍了以下主流语言SDK的安装、开发操作、参数、示例和常见问题处理。

语言	参考文档
Java	Java SDK参考
Python	Python SDK参考
Android	Android SDK参考
iOS	iOS SDK参考
.NET	.NET SDK参考
Node.js	Node.js SDK参考
Browser.js	Browser.js SDK参考
PHP	PHP SDK参考
C	C SDK参考
Ruby	Ruby SDK参考
Go	Go SDK参考
Media-C	Media-C SDK参考

Java-SDK

前言

本文档基于OSS Java SDK 2.8.3编写。

环境准备

请使用JDK 6及以上版本。

兼容性

对于2.×.×系列SDK：

- 接口： 兼容。
- 命名空间： 兼容。

对于1.0.× 系列SDK：

- 接口：兼容。
- 命名空间：不兼容。2.0.0版本移除了1.0.x版本中Table Store相关代码，将包名称com.aliyun.openservices.*和com.aliyun.openservices.oss.*更换为com.aliyun.oss.*。

SDK下载

- 直接下载
- 通过GitHub下载
- 历史版本下载

更多信息请参见OSS API文档。

SDK安装

请参见OSS Java SDK安装。

示例程序

OSS Java SDK提供丰富的示例程序，方便您参考或直接使用。您可以通过以下两种方式获取示例程序：

通过GitHub查看和下载示例代码。

解压OSS Java SDK开发包，src/samples目录下为示例代码。

示例包括以下内容：

示例文件	示例内容
GetStartedSample.java	快速入门
CreateFolderSample.java	简单上传中的创建文件夹
AppendObjectSample.java	追加上传
UploadSample.java	断点续传上传
MultipartUploadSample.java	分片上传
GetProgressSample.java	进度条
CallbackSample.java	上传回调
CRCSample.java	上传时进行CRC校验
SimpleGetObjectSample.java	下载文件
DownloadSample.java	断点续传下载
ObjectMetaSample.java	文件元信息
ListObjectsSample.java	列举文件
UploadPartCopySample.java	拷贝文件
DeleteObjectsSample.java	删除文件
BucketOperationsSample.java	设置存储空间的授权访问、生命周期、访问日志、防盗链、CORS等
ImageSample.java	图片处理
PostObjectSample.java	PostObject，该实现不依赖于Java SDK

安装

安装方式

方式一：在Maven项目中加入依赖项（推荐方式）

在Maven工程中使用OSS Java SDK，只需在pom.xml中加入相应依赖项即可。以2.8.3版本为例，在<dependencies>内加入如下内容：

```
<dependency>
<groupId>com.aliyun.oss</groupId>
<artifactId>aliyun-sdk-oss</artifactId>
<version>2.8.3</version>
</dependency>
```

方式二：在Eclipse项目中导入JAR包

以2.8.3版本为例，步骤如下：

1. 下载Java SDK开发包。
2. 解压该开发包。
3. 将解压后文件夹中的文件aliyun-sdk-oss-<versionId>.jar以及lib文件夹下的所有文件拷贝到您的项目中。
4. 在Eclipse中选择您的工程，右击选择 **Properties** > **Java Build Path** > **Libraries** > **Add JARs**。
5. 选中您在第3步拷贝的所有JAR文件，导入到Libraries中。

示例工程

OSS Java SDK提供了基于Maven和Ant的示例工程。您可以在本地设备上编译和运行示例工程，也可以以示例工程为基础开发您的应用。工程的编译和运行方法，请参见工程目录下的README.md。

- Maven示例工程：aliyun-oss-java-sdk-demo-mvn.zip
- Ant示例工程：aliyun-oss-java-sdk-demo-ant.zip

说明：编译和运行前，请修改HelloOSS.java中的以下参数为您的实际信息：

- endpoint
- accessKeyId
- accessKeySecret
- bucketName

初始化

OSSClient是OSS的Java客户端，用于管理存储空间和文件等OSS资源。使用Java SDK发起OSS请求，您需要初始化一个OSSClient实例，并根据需要修改ClientConfiguration的默认配置项。

新建OSSClient

新建OSSClient时，需要指定Endpoint。有关Endpoint的更多信息，请参见访问域名和数据中心和自定义访问

域名。

使用OSS域名新建OSSClient

以下代码用于使用OSS域名新建OSSClient：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 关闭OSSClient。
ossClient.shutdown();
```

您的工程中可以有一个或多个OSSClient。OSSClient可以并发使用。

使用自定义域名新建OSSClient

以下代码用于使用自定义域名新建OSSClient：

```
String endpoint = "<yourEndpoint>";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建ClientConfiguration实例，按照您的需要修改默认参数。
ClientConfiguration conf = new ClientConfiguration();
// 开启支持CNAME。CNAME是指将自定义域名绑定到存储空间上。
conf.setSupportCname(true);

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret, conf);

// 关闭OSSClient。
ossClient.shutdown();
```

说明：使用自定义域名时无法使用ossClient.listBuckets方法。

专有云/专有域环境新建OSSClient

以下代码用于使用专有云或专有域环境新建OSSClient：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建ClientConfiguration实例，按照您的需要修改默认参数。
ClientConfiguration conf = new ClientConfiguration();
// 关闭CNAME选项。
conf.setSupportCname(false);

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret, conf);

// 关闭OSSClient。
ossClient.shutdown();
```

使用IP新建OSSClient

以下代码用于使用IP新建OSSClient：

```
// 某些特殊情况（比如专有域）下，您需要将IP地址做作为Endpoint，请按照实际IP地址填写。
String endpoint = "http://10.10.10.10";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建ClientConfiguration。
ClientConfiguration conf = new ClientConfiguration();
// 开启二级域名访问OSS，默认不开启。OSS Java SDK 2.1.2以前的版本需要设置此值，OSS Java SDK 2.1.2以后的版本会自动检测到IP地址，不需要再设置此值。
conf.setSLDEnabled(true);

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret, conf);

// 关闭OSSClient。
ossClient.shutdown();
```

使用STS新建OSSClient

以下代码用于使用STS新建一个OSSClient：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
```

```
String securityToken = "<yourSecurityToken>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret, securityToken);

// 关闭OSSClient。
ossClient.shutdown();
```

更多信息请参见RAM和STS介绍和授权访问。

配置OSSClient

ClientConfiguration是OSSClient的配置类，您可通过此类来配置代理、连接超时、最大连接数等参数。可设置的参数如下：

参数	描述	方法
MaxConnections	允许打开的最大HTTP连接数。默认为1024	ClientConfiguration.setMaxConnections
SocketTimeout	Socket层传输数据的超时时间（单位：毫秒）。默认为50000毫秒	ClientConfiguration.setSocketTimeout
ConnectionTimeout	建立连接的超时时间（单位：毫秒）。默认为50000毫秒	ClientConfiguration.setConnectionTimeout
ConnectionRequestTimeout	从连接池中获取连接的超时时间（单位：毫秒）。默认不超时	ClientConfiguration.setConnectionRequestTimeout
IdleConnectionTime	连接空闲超时时间，超时则关闭连接（单位：毫秒，默认为60000毫秒	ClientConfiguration.setIdleConnectionTime
MaxErrorRetry	请求失败后最大的重试次数。默认3次	ClientConfiguration.setMaxErrorRetry
SupportCname	是否支持CNAME作为Endpoint，默认支持CNAME	ClientConfiguration.setSupportCname
SLDEnabled	是否开启二级域名（Second Level Domain）的访问方式，默认不开启	ClientConfiguration.setSLDEnabled
Protocol	连接OSS所采用的协议（HTTP/HTTPS），默认为HTTP	ClientConfiguration.setProtocol
UserAgent	用户代理，指HTTP的User-Agent头。默认为”aliyun-sdk-java”	ClientConfiguration.setUserAgent
ProxyHost	代理服务器主机地址	ClientConfiguration.setProxyHost
ProxyPort	代理服务器端口	ClientConfiguration.setProxyPort

ProxyUsername	代理服务器验证的用户名	ClientConfiguration.setProxyUsername
ProxyPassword	代理服务器验证的密码	ClientConfiguration.setProxyPassword

以下代码用于使用ClientConfiguration设置OSSClient参数：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建ClientConfiguration。ClientConfiguration是OSSClient的配置类，可配置代理、连接超时、最大连接数等参数。
ClientConfiguration conf = new ClientConfiguration();

// 设置OSSClient允许打开的最大HTTP连接数，默认为1024个。
conf.setMaxConnections(200);
// 设置Socket层传输数据的超时时间，默认为50000毫秒。
conf.setSocketTimeout(10000);
// 设置建立连接的超时时间，默认为50000毫秒。
conf.setConnectionTimeout(10000);
// 设置从连接池中获取连接的超时时间（单位：毫秒），默认不超时。
conf.setConnectionRequestTimeout(1000);
// 设置连接空闲超时时间。超时则关闭连接，默认为60000毫秒。
conf.setIdleConnectionTime(10000);
// 设置失败请求重试次数，默认为3次。
conf.setMaxErrorRetry(5);
// 设置是否支持将自定义域名作为Endpoint，默认支持。
conf.setSupportCname(true);
// 设置是否开启二级域名的访问方式，默认不开启。
conf.setSLDEnabled(true);
// 设置连接OSS所使用的协议（HTTP/HTTPS），默认为HTTP。
conf.setProtocol(Protocol.HTTP);
// 设置用户代理，指HTTP的User-Agent头，默认为aliyun-sdk-java。
conf.setUserAgent("aliyun-sdk-java");
// 设置代理服务器端口。
conf.setProxyHost("<yourProxyHost>");
// 设置代理服务器验证的用户名。
conf.setProxyUsername("<yourProxyUserName>");
// 设置代理服务器验证的密码。
conf.setProxyPassword("<yourProxyPassword>");

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret, conf);

// 关闭OSSClient。
ossClient.shutdown();
```

更多信息请参见阿里云OSS Java SDK超时时间设置。

快速入门

本节介绍如何快速使用OSS Java SDK完成常见操作，如创建存储空间、上传文件、下载文件等。

创建存储空间

存储空间是OSS全局命名空间，相当于数据的容器，可以存储若干文件。以下代码用于新建一个存储空间：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 创建存储空间。
ossClient.createBucket(bucketName);

// 关闭OSSClient。
ossClient.shutdown();
```

存储空间的命名规范，请参见基本概念中的命名规范。创建存储空间详情，请参见管理存储空间。

上传文件

以下代码用于上传文件至OSS：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 上传文件。
String content = "Hello OSS";
ossClient.putObject(bucketName, objectName, new ByteArrayInputStream(content.getBytes()));
```

```
// 关闭OSSClient。
ossClient.shutdown();
```

上传文件详情请参见上传文件。

下载文件

以下代码用于获取文件的文本内容：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 调用ossClient.getObject返回一个OSSObject实例，该实例包含文件内容及文件元信息。
OSSObject ossObject = ossClient.getObject(bucketName, objectName);
// 调用ossObject.getObjectContent获取文件输入流，可读取此输入流获取其内容。
InputStream content = ossObject.getObjectContent();
if (content != null) {
BufferedReader reader = new BufferedReader(new InputStreamReader(content));
while (true) {
String line = reader.readLine();
if (line == null) break;
System.out.println("\n" + line);
}
// 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
content.close();
}

// 关闭OSSClient。
ossClient.shutdown();
```

下载文件详情请参见下载文件。

列举文件

以下代码用于列举指定存储空间下的文件。默认列举100个文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
```

```
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// ossClient.listObjects返回ObjectListing实例，包含此次listObject请求的返回结果。
ObjectListing objectListing = ossClient.listObjects(bucketName);
// objectListing.getObjectSummaries获取所有文件的描述信息。
for (OSSObjectSummary objectSummary : objectListing.getObjectSummaries()) {
System.out.println(" - " + objectSummary.getKey() + " " +
"(size = " + objectSummary.getSize() + ")");
}

// 关闭OSSClient。
ossClient.shutdown();
```

列举功能详情请参见管理文件中的列出存储空间中的文件。

删除文件

以下代码用于删除指定文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 删除文件。
ossClient.deleteObject(bucketName, objectName);

// 关闭OSSClient。
ossClient.shutdown();
```

删除文件详情请参见管理文件中的删除文件。删除文件完整代码请参见GitHub。

管理存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。

创建存储空间

以下代码用于创建存储空间：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 创建存储空间。
String bucketName = "<yourBucketName>";
// 新建存储空间默认为标准存储类型，私有读写权限。
ossClient.createBucket(bucketName);

// 关闭OSSClient。
ossClient.shutdown();
```

存储空间的命名规范，请参见基本概念中的命名规范。

您可以在创建存储空间时指定存储空间的权限和存储类型。如果需要创建低频或归档类型的存储空间，请使用Java SDK 2.6.0及以上版本。示例代码如下：

```
CreateBucketRequest createBucketRequest = new CreateBucketRequest(bucketName);
// 设置存储空间的权限为公共读，默认是私有读写。
createBucketRequest.setCannedACL(CannedAccessControlList.PublicRead);
// 设置存储空间的存储类型为低频访问类型，默认是标准类型。
createBucketRequest.setStorageClass(StorageClass.IA);
ossClient.createBucket(createBucketRequest);
```

列举存储空间

存储空间按照字母顺序排列。您可以列举所有的存储空间，或符合指定条件的存储空间。如果有低频类型或归档类型的存储空间，请使用Java SDK 2.6.0及以上版本。

列举所有的存储空间

以下代码用于列举所有的存储空间：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 列举存储空间。
List<Bucket> buckets = ossClient.listBuckets();
for (Bucket bucket : buckets) {
System.out.println(" - " + bucket.getName());
}

// 关闭OSSClient。
ossClient.shutdown();
```

列举指定前缀的存储空间

以下代码用于列举包含指定前缀（prefix）的存储空间：

```
ListBucketsRequest listBucketsRequest = new ListBucketsRequest();
// 列举指定前缀的存储空间。
listBucketsRequest.setPrefix("<yourBucketPrefix>");
BucketList bucketList = ossClient.listBuckets(listBucketsRequest);
for (Bucket bucket : bucketList.getBucketList()) {
System.out.println(" - " + bucket.getName());
}
```

列举指定marker之后的存储空间

参数marker代表存储空间名称。以下代码用于列举指定marker之后的存储空间：

```
ListBucketsRequest listBucketsRequest = new ListBucketsRequest();
// 列举指定marker之后的存储空间。
listBucketsRequest.setMarker("<yourBucketMarker>");
BucketList bucketList = ossClient.listBuckets(listBucketsRequest);
for (Bucket bucket : bucketList.getBucketList()) {
System.out.println(" - " + bucket.getName());
}
```

列举指定个数的存储空间

以下代码用于列举指定个数（maxKeys）的存储空间:

```
ListBucketsRequest listBucketsRequest = new ListBucketsRequest();
// 限定此次列举存储空间的个数为500。默认值为100，最大值为1000。
```

```
listBucketsRequest.setMaxKeys(500);
BucketList bucketList = ossClient.listBuckets(listBucketsRequest);
for (Bucket bucket : bucketList.getBucketList()) {
System.out.println(" - " + bucket.getName());
}
```

判断存储空间是否存在

以下代码用于判断指定的存储空间是否存在：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

boolean exists = ossClient.doesBucketExist("<yourBucketName>");

// 关闭OSSClient。
ossClient.shutdown();
```

设置存储空间的访问权限

存储空间的访问权限（ACL）有以下三类：

访问权限	访问权限值
私有	CannedAccessControlList.Private
公共读私有写	CannedAccessControlList.PublicRead
公共读写	CannedAccessControlList.PublicReadWrite

以下代码用于设置存储空间的访问权限：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 设置存储空间的访问权限为私有读写。
```

```
ossClient.setBucketAcl("<yourBucketName>", CannedAccessControlList.Private);

// 关闭OSSClient。
ossClient.shutdown();
```

获取存储空间的访问权限

以下代码用于获取存储空间的访问权限：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);
// 获取存储空间的访问权限。
AccessControlList acl = ossClient.getBucketAcl("<yourBucketName>");
System.out.println(acl.toString());

// 关闭OSSClient。
ossClient.shutdown();
```

获取存储空间的地域

以下代码用于获取存储空间的地域（称为Region或Location）：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

String location = ossClient.getBucketLocation("<yourBucketName>");
System.out.println(location);

// 关闭OSSClient。
ossClient.shutdown();
```

关于地域的详细信息请参见基本概念中的地域。

获取存储空间的信息

以下代码用于获取存储空间的信息（Info）：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 存储空间的信息包括地域（Region或Location）、创建日期（CreationDate）、拥有者（Owner）、权限（Grants）等。
BucketInfo info = ossClient.getBucketInfo("<yourBucketName>");
// 获取地域。
info.getBucket().getLocation();
// 获取创建日期。
info.getBucket().getCreationDate();
// 获取拥有者信息。
info.getBucket().getOwner();
// 获取权限信息。
info.getGrants();

// 关闭OSSClient。
ossClient.shutdown();
```

删除存储空间

删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。

以下代码用于删除存储空间：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 删除存储空间。
ossClient.deleteBucket("<yourBucketName>");

// 关闭OSSClient。
ossClient.shutdown();
```

上传文件

概述

在OSS中，操作的基本数据单元是文件（Object）。OSS Java SDK提供了丰富的文件上传方式：

- 简单上传：包括流式上传和文件上传。最大不能超过5GB。
- 表单上传：最大不能超过5GB。
- 追加上传：最大不能超过5GB。
- 断点续传上传：支持并发、断点续传、自定义分片大小。大文件上传推荐使用断点续传。最大不能超过48.8TB。
- 分片上传：当文件较大时，可以使用分片上传，最大不能超过48.8TB。

说明：各种上传方式的适用场景请参见开发指南中的上传文件。

上传过程中，您还可以通过上传进度条功能查看上传进度。上传完成后，您还可以进行上传回调。

简单上传

流式上传和文件上传通称为简单上传。流式上传使用InputStream作为文件的数据源。文件上传使用本地文件作为OSS文件的数据源。简单上传的完整代码请参见GitHub。

流式上传

使用ossClient.putObject上传数据流到OSS。

上传字符串

以下代码用于上传字符串：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
```

```
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 上传字符串。
String content = "Hello OSS";
ossClient.putObject("<yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(content.getBytes()));

// 关闭OSSClient。
ossClient.shutdown();
```

上传Byte数组

以下代码用于上传Byte数组：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId,accessKeySecret);

// 上传Byte数组。
byte[] content = "Hello OSS".getBytes();
ossClient.putObject("<yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(content));

// 关闭OSSClient。
ossClient.shutdown();
```

上传网络流

以下代码用于上传网络流：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 上传网络流。
InputStream inputStream = new URL("https://www.aliyun.com/").openStream();
ossClient.putObject("<yourBucketName>", "<yourObjectName>", inputStream);
```

```
// 关闭OSSClient。
ossClient.shutdown();
```

上传文件流

以下代码用于上传文件流：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践，创建并使用RAM子账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 上传文件流。
InputStream inputStream = new FileInputStream("<yourlocalFile>");
ossClient.putObject("<yourBucketName>", "<yourObjectName>", inputStream);

// 关闭OSSClient。
ossClient.shutdown();
```

文件上传

上传本地文件

以下代码用于上传本地文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 上传文件。<yourLocalFile>由本地文件路径加文件名包括后缀组成，例如/users/local/myfile.txt。
ossClient.putObject("<yourBucketName>", "<yourObjectName>", new File("<yourLocalFile>"));

// 关闭OSSClient。
ossClient.shutdown();
```

表单上传

表单上传是使用HTML表单形式上传文件到指定存储空间中，文件最大不能超过5GB。

表单上传的适用场景介绍请参见开发指南中的表单上传。PostObject更详情信息请参见Java模拟PostObject表单上传OSS。

以下代码用于表单上传：

```
public class PostObjectSample {

// 上传文件
private String localFilePath = "<yourLocalFile>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
private String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
private String accessKeyId = "<yourAccessKeyId>";
private String accessKeySecret = "<yourAccessKeySecret>";
// 存储空间名称
private String bucketName = "<yourBucketName>";
// 文件名称
private String objectName = "<yourObjectName>";

/**
* 表单上传
* @throws Exception
*/
private void PostObject() throws Exception {

// 在URL中添加存储空间名称，添加后URL如下：http://yourBucketName.oss-cn-hangzhou.aliyuncs.com
String urlStr = endpoint.replace("http://", "http://" + bucketName+ ".");

// 表单Map。
Map<String, String> formFields = new LinkedHashMap<String, String>();

// 设置文件名称。
formFields.put("key", this.objectName);
// 设置Content-Disposition。
formFields.put("Content-Disposition", "attachment;filename="
+ localFilePath);
// 设置OSSAccessKeyId。
formFields.put("OSSAccessKeyId", accessKeyId);

String policy = "{\"expiration\": \"2120-01-01T12:00:00.000Z\",\"conditions\": [[\"content-length-range\", 0, 104857600]]}";
String encodePolicy = new String(Base64.encodeBase64(policy.getBytes()));
// 设置policy。
formFields.put("policy", encodePolicy);

// 生成并格式化签名。
```

```
String signaturecom = com.aliyun.oss.common.auth.ServiceSignature.create().computeSignature(accessKeySecret,
encodePolicy);
// 设置签名。
formFields.put("Signature", signaturecom);

String ret = formUpload(urlStr, formFields, localFilePath);

System.out.println("Post Object [" + this.objectName + "] to bucket [" + bucketName + "]");
System.out.println("post reponse:" + ret);
}

private static String formUpload(String urlStr, Map<String, String> formFields, String localFile)
throws Exception {
String res = "";
HttpURLConnection conn = null;
String boundary = "9431149156168";

try {
URL url = new URL(urlStr);
conn = (HttpURLConnection) url.openConnection();
conn.setConnectTimeout(5000);
conn.setReadTimeout(30000);
conn.setDoOutput(true);
conn.setDoInput(true);
conn.setRequestMethod("POST");
conn.setRequestProperty("User-Agent",
"Mozilla/5.0 (Windows; U; Windows NT 6.1; zh-CN; rv:1.9.2.6)");
// 设置MD5值。MD5值是由整个body计算得出的。
conn.setRequestProperty("Content-MD5", "<yourContentMD5>");
conn.setRequestProperty("Content-Type",
"multipart/form-data; boundary=" + boundary);
OutputStream out = new DataOutputStream(conn.getOutputStream());

// 遍历读取表单Map中的数据，将数据写入到输出流中。
if (formFields != null) {
StringBuffer strBuf = new StringBuffer();
Iterator<Entry<String, String>> iter = formFields.entrySet().iterator();
int i = 0;

while (iter.hasNext()) {
Entry<String, String> entry = iter.next();
String inputName = entry.getKey();
String inputValue = entry.getValue();

if (inputValue == null) {
continue;
}

if (i == 0) {
strBuf.append("--").append(boundary).append("\r\n");
strBuf.append("Content-Disposition: form-data; name=\""
+ inputName + "\"\r\n\r\n");
strBuf.append(inputValue);
} else {
strBuf.append("\r\n").append("--").append(boundary).append("\r\n");
strBuf.append("Content-Disposition: form-data; name=\""
```

```
+ inputName + "\"\r\n\r\n");
strBuf.append(inputValue);
}

i++;
}
out.write(strBuf.toString().getBytes());
}

// 读取文件信息，将要上传的文件写入到输出流中。
File file = new File(localFile);
String filename = file.getName();
String contentType = new MimetypesFileTypeMap().getContentType(file);
if (contentType == null || contentType.equals("")) {
contentType = "application/octet-stream";
}

StringBuffer strBuf = new StringBuffer();
strBuf.append("\r\n").append("--").append(boundary)
.append("\r\n");
strBuf.append("Content-Disposition: form-data; name=\"file\"; "
+ "filename=\"" + filename + "\"\r\n");
strBuf.append("Content-Type: " + contentType + "\r\n\r\n");

out.write(strBuf.toString().getBytes());

DataInputStream in = new DataInputStream(new FileInputStream(file));
int bytes = 0;
byte[] bufferOut = new byte[1024];
while ((bytes = in.read(bufferOut)) != -1) {
out.write(bufferOut, 0, bytes);
}
in.close();

byte[] endData = ("\r\n--" + boundary + "--\r\n").getBytes();
out.write(endData);
out.flush();
out.close();

// 读取返回数据。
strBuf = new StringBuffer();
BufferedReader reader = new BufferedReader(new InputStreamReader(conn.getInputStream()));
String line = null;
while ((line = reader.readLine()) != null) {
strBuf.append(line).append("\n");
}
res = strBuf.toString();
reader.close();
reader = null;
} catch (Exception e) {
System.err.println("Send post request exception: " + e);
throw e;
} finally {
if (conn != null) {
conn.disconnect();
conn = null;
```

```
}
}
return res;
}

public static void main(String[] args) throws Exception {
PostObjectSample ossPostObject = new PostObjectSample();
ossPostObject.PostObject();
}
}
```

追加上传

追加类型的文件（Append Object）暂时不支持copyObject操作。

以下代码用于追加上传文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

String content1 = "Hello OSS A \n";
String content2 = "Hello OSS B \n";
String content3 = "Hello OSS C \n";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

ObjectMetadata meta = new ObjectMetadata();
// 指定上传的内容类型。
meta.setContentType("text/plain");

// 通过AppendObjectRequest设置多个参数。
AppendObjectRequest appendObjectRequest = new AppendObjectRequest("<yourBucketName>",
"<yourObjectName>", new ByteArrayInputStream(content1.getBytes()),meta);

// 通过AppendObjectRequest设置单个参数。
// 设置存储空间名称。
//appendObjectRequest.setBucketName("<yourBucketName>");
// 设置文件名称。
//appendObjectRequest.setKey("<yourObjectName>");
// 设置待追加的内容。有两种可选类型：InputStream类型和File类型。这里为InputStream类型。
//appendObjectRequest.setInputStream(new ByteArrayInputStream(content1.getBytes()));
// 设置待追加的内容。有两种可选类型：InputStream类型和File类型。这里为File类型。
//appendObjectRequest.setFile(new File("<yourLocalFile>"));
// 指定文件的元信息，第一次追加时有效。
```

```
//appendObjectRequest.setMetadata(meta);

// 第一次追加。
// 设置文件的追加位置。
appendObjectRequest.setPosition(0L);
AppendObjectResult appendObjectResult = ossClient.appendObject(appendObjectRequest);
// 文件的64位CRC值。此值根据ECMA-182标准计算得出。
System.out.println(appendObjectResult.getObjectCRC());

// 第二次追加。
// nextPosition指明下一次请求中应当提供的Position，即文件当前的长度。
appendObjectRequest.setPosition(appendObjectResult.getNextPosition());
appendObjectRequest.setInputStream(new ByteArrayInputStream(content2.getBytes()));
appendObjectResult = ossClient.appendObject(appendObjectRequest);

// 第三次追加。
appendObjectRequest.setPosition(appendObjectResult.getNextPosition());
appendObjectRequest.setInputStream(new ByteArrayInputStream(content3.getBytes()));
appendObjectResult = ossClient.appendObject(appendObjectRequest);

// 关闭OSSClient。
ossClient.shutdown();
```

追加上传详情请参见开发指南中的追加上传。追加上传的完整代码请参见GitHub。

断点续传上传

断点续传上传将要上传的文件分成若干个分片（Part）分别上传，所有分片都上传完成后，将所有分片合并成完整的文件，完成整个文件的上传。

以下代码用于断点续传上传：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

ObjectMetadata meta = new ObjectMetadata();
// 指定上传的内容类型。
meta.setContentType("text/plain");

// 通过UploadFileRequest设置多个参数。
UploadFileRequest uploadFileRequest = new UploadFileRequest("<yourBucketName>","<yourObjectName>");
```

```
// 通过UploadFileRequest设置单个参数。
// 设置存储空间名称。
//uploadFileRequest.setBucketName("yourucketname");
// 设置文件名称。
//uploadFileRequest.setKey("<yourObjectName>");
// 指定上传的本地文件。
uploadFileRequest.setUploadFile("<yourLocalFile>");
// 指定上传并发线程数，默认为1。
uploadFileRequest.setTaskNum(5);
// 指定上传的分片大小，范围为100KB~5GB，默认为文件大小/10000。
uploadFileRequest.setPartSize(1 * 1024 * 1024);
// 开启断点续传，默认关闭。
uploadFileRequest.setEnableCheckpoint(true);
// 记录本地分片上传结果的文件。开启断点续传功能时需要设置此参数，上传过程中的进度信息会保存在该文件中，如果某一分片上传失败，再次上传时会根据文件中记录的点继续上传。上传完成后，该文件会被删除。默认与待上传的本地文件同目录，为uploadFile.ucp。
uploadFileRequest.setCheckpointFile("<yourCheckpointFile>");
// 文件的元数据。
uploadFileRequest.setObjectMetadata(meta);
// 设置上传成功回调，参数为Callback类型。
uploadFileRequest.setCallback("<yourCallbackEvent>");

// 断点续传上传。
ossClient.uploadFile(uploadFileRequest);

// 关闭OSSClient。
ossClient.shutdown();
```

断点续传详情请参见开发指南中的**断点续传**。

分片上传

分片上传（Multipart Upload）分为以下三个步骤:

初始化一个分片上传事件。
调用ossClient.initiateMultipartUpload方法返回OSS创建的全局唯一的uploadId。

上传分片。
调用ossClient.uploadPart方法上传分片数据。需要注意：

- 对于同一个uploadId，分片号（partNumber）标识了该分片在整个文件内的相对位置。如果使用同一个分片号上传了新的数据，那么OSS上这个分片已有的数据将会被覆盖。
- OSS将收到的分片数据的MD5值放在ETag头内返回给用户。
- SDK自动设置Content-MD5。OSS计算上传数据的MD5值，并与SDK计算的MD5值比较，如果不一致则返回InvalidDigest错误码。

完成分片上传。
所有分片上传完成后，调用ossClient.completeMultipartUpload方法将所有分片合并成完整的文件。

以下通过一个完整的示例对分片上传的流程进行逐步解析：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

/* 步骤1：初始化一个分片上传事件。
*/
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(bucketName, objectName);
InitiateMultipartUploadResult result = ossClient.initiateMultipartUpload(request);
// 返回uploadId，它是分片上传事件的唯一标识，您可以根据这个ID来发起相关的操作，如取消分片上传、查询分片上传等。
String uploadId = result.getUploadId();

/* 步骤2：上传分片。
*/
// partETags是PartETag的集合。PartETag由分片的ETag和分片号组成。
List<PartETag> partETags = new ArrayList<PartETag>();
// 计算文件有多少个分片。
final long partSize = 1 * 1024 * 1024L; // 1MB
final File sampleFile = new File("<localFile>");
long fileLength = sampleFile.length();
int partCount = (int) (fileLength / partSize);
if (fileLength % partSize != 0) {
partCount++;
}
// 遍历分片上传。
for (int i = 0; i < partCount; i++) {
long startPos = i * partSize;
long curPartSize = (i + 1 == partCount) ? (fileLength - startPos) : partSize;
InputStream instream = new FileInputStream(sampleFile);
// 跳过已经上传的分片。
instream.skip(startPos);
UploadPartRequest uploadPartRequest = new UploadPartRequest();
uploadPartRequest.setBucketName(bucketName);
uploadPartRequest.setKey(objectName);
uploadPartRequest.setUploadId(uploadId);
uploadPartRequest.setInputStream(instream);
// 设置分片大小。除了最后一个分片没有大小限制，其他的分片最小为100KB。
uploadPartRequest.setPartSize(curPartSize);
// 设置分片号。每一个上传的分片都有一个分片号，取值范围是1~10000，如果超出这个范围，OSS将返回InvalidArgument的错误码。
uploadPartRequest.setPartNumber( i + 1);
```

```
// 每个分片不需要按顺序上传，甚至可以在不同客户端上传，OSS会按照分片号排序组成完整的文件。
UploadPartResult uploadPartResult = ossClient.uploadPart(uploadPartRequest);
// 每次上传分片之后，OSS的返回结果会包含一个PartETag。PartETag将被保存到partETags中。
partETags.add(uploadPartResult.getPartETag());
}

/* 步骤3：完成分片上传。
*/
// 排序。partETags必须按分片号升序排列。
Collections.sort(partETags, new Comparator<PartETag>() {
public int compare(PartETag p1, PartETag p2) {
return p1.getPartNumber() - p2.getPartNumber();
}
});
// 在执行该操作时，需要提供所有有效的partETags。OSS收到提交的partETags后，会逐一验证每个分片的有效性。当所有的数据分片验证通过后，OSS将把这些分片组合成一个完整的文件。
CompleteMultipartUploadRequest completeMultipartUploadRequest =
new CompleteMultipartUploadRequest(bucketName, objectName, uploadId, partETags);
ossClient.completeMultipartUpload(completeMultipartUploadRequest);

// 关闭OSSClient。
ossClient.shutdown();
```

分片上传的完整代码请参见GitHub。

取消分片上传事件

您可以调用ossClient.abortMultipartUpload方法来取消分片上传事件。当一个分片上传事件被取消后，无法再使用这个uploadId做任何操作，已经上传的分片数据会被删除。

以下代码用于取消分片上传事件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 取消分片上传，其中uploadId来自于InitiateMultipartUpload。
AbortMultipartUploadRequest abortMultipartUploadRequest =
new AbortMultipartUploadRequest("<yourBucketName>", "<yourObjectName>", "<uploadId>");
ossClient.abortMultipartUpload(abortMultipartUploadRequest);

// 关闭OSSClient。
ossClient.shutdown();
```

列举已上传的分片

调用ossClient.listParts方法列举出指定uploadId下所有已经上传成功的分片。

简单列举已上传的分片

以下代码用于简单列举已上传的分片：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 列举已上传的分片，其中uploadId来自于InitiateMultipartUpload返回的结果。
ListPartsRequest listPartsRequest = new ListPartsRequest("<yourBucketName>", "<yourObjectName>", "<uploadId>");
// 设置uploadId。
//listPartsRequest.setUploadId(uploadId);
// 设置分页时每一页中分片数量为100个。默认列举1000个分片。
listPartsRequest.setMaxParts(100);
// 指定List的起始位置。只有分片号大于该参数值的分片会被列出。
listPartsRequest.setPartNumberMarker(2);
PartListing partListing = ossClient.listParts(listPartsRequest);

for (PartSummary part : partListing.getParts()) {
// 获取分片号。
part.getPartNumber();
// 获取分片数据大小。
part.getSize();
// 获取分片的ETag。
part.getETag();
// 获取分片的最后修改时间。
part.getLastModified();
}

// 关闭OSSClient。
ossClient.shutdown();
```

列举所有已上传的分片

默认情况下，listParts只能一次列举1000个分片。当分片数量大于1000时，请使用以下代码列举所有已上传的分片。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
```

```
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 列举所有已上传的分片。
PartListing partListing;
ListPartsRequest listPartsRequest = new ListPartsRequest("<yourBucketName>", "<yourObjectName>", "<uploadId>");

do {
partListing = ossClient.listParts(listPartsRequest);

for (PartSummary part : partListing.getParts()) {
// 获取分片号。
part.getPartNumber();
// 获取分片数据大小。
part.getSize();
// 获取分片的ETag。
part.getETag();
// 获取分片的最后修改时间。
part.getLastModified();
}
// 指定List的起始位置，只有分片号大于该参数值的分片会被列出。
listPartsRequest.setPartNumberMarker(partListing.getNextPartNumberMarker());
} while (partListing.isTruncated());

// 关闭OSSClient。
ossClient.shutdown()
```

分页列举所有已上传的分片

以下代码用于指定每页分片的数量，分页列举所有分片。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 分页列举已上传的分片。
PartListing partListing;
ListPartsRequest listPartsRequest = new ListPartsRequest("<yourBucketName>", "<yourObjectName>", "<uploadId>");
// 设置分页时每一页100个分片。
listPartsRequest.setMaxParts(100);
```

```
do {
partListing = ossClient.listParts(listPartsRequest);

for (PartSummary part : partListing.getParts()) {
// 获取分片号。
part.getPartNumber();
// 获取分片数据大小。
part.getSize();
// 获取分片的ETag。
part.getETag();
// 获取分片的最后修改时间。
part.getLastModified();
}

listPartsRequest.setPartNumberMarker(partListing.getNextPartNumberMarker());

} while (partListing.isTruncated());

// 关闭OSSClient。
ossClient.shutdown();
```

列举存储空间下所有分片上传事件

调用ossClient.listMultipartUploads方法列举出所有执行中的分片上传事件，即已初始化但尚未完成或已取消的分片上传事件。可设置的参数如下：

参数	作用	如何设置
prefix	限定返回的文件名称必须以指定的prefix作为前缀。注意使用prefix查询时，返回的文件名称中仍会包含prefix。	ListMultipartUploadsRequest.setPrefix(String prefix)
delimiter	用于对文件名称进行分组的一个字符。所有名称包含指定的前缀且第一次出现delimiter字符之间的文件作为一组元素。	ListMultipartUploadsRequest.setDelimiter(String delimiter)
maxUploads	限定此次返回分片上传事件的最大数目，默认值和最大值均为1000。	ListMultipartUploadsRequest.setMaxUploads(Integer maxUploads)
keyMarker	所有文件名称的字母序大于keyMarker参数值的分片上传事件。可以与uploadIdMarker参数一同使用来指定返回结果的起始位置。	ListMultipartUploadsRequest.setKeyMarker(String keyMarker)
uploadIdMarker	与keyMarker参数一同使用来指定返回结果的起始位置。 如果未设置keyMarker参数，则此参数无效。如果设置了keyMarker参数，则查询结果中包含： - 名称的字母序大于keyMarker参数值的所有文件。 - 文件名称等于keyMarker参数	ListMultipartUploadsRequest.setUploadIdMarker(String uploadIdMarker)

	值且uploadId比uploadIdMarker参数值大的所有分片上传事件。	

简单列举分片上传事件

以下代码用于列举分片上传事件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 列举分片上传事件。默认列举1000个分片。
ListMultipartUploadsRequest listMultipartUploadsRequest = new ListMultipartUploadsRequest(bucketName);
MultipartUploadListing multipartUploadListing = ossClient.listMultipartUploads(listMultipartUploadsRequest);

for (MultipartUpload multipartUpload : multipartUploadListing.getMultipartUploads()) {
// 获取uploadId。
multipartUpload.getUploadId();
// 获取Key。
multipartUpload.getKey();
// 获取分片上传的初始化时间。
multipartUpload.getInitiated();
}

// 关闭OSSClient。
ossClient.shutdown();
```

返回结果中isTruncated为false，返回nextKeyMarker和nextUploadIdMarker作为下次读取的起点。如果没有一次性获取所有的上传事件，可以采用分页列举的方式。

列举全部分片上传事件

默认情况下，listMultipartUploads 只能一次列举1000个分片。当分片数量大于1000时，请使用以下代码列举全部分片上传事件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
```

```
// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 列举分片上传事件。
MultipartUploadListing multipartUploadListing;
ListMultipartUploadsRequest listMultipartUploadsRequest = new ListMultipartUploadsRequest(bucketName);

do {
multipartUploadListing = ossClient.listMultipartUploads(listMultipartUploadsRequest);

for (MultipartUpload multipartUpload : multipartUploadListing.getMultipartUploads()) {
// 获取uploadId。
multipartUpload.getUploadId();
// 获取文件名称。
multipartUpload.getKey();
// 获取分片上传的初始化时间。
multipartUpload.getInitiated();
}

listMultipartUploadsRequest.setKeyMarker(multipartUploadListing.getNextKeyMarker());

listMultipartUploadsRequest.setUploadIdMarker(multipartUploadListing.getNextUploadIdMarker());
} while (multipartUploadListing.isTruncated());

// 关闭OSSClient。
ossClient.shutdown();
```

分页列举全部上传事件

以下代码用于分页列举所有上传事件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 列举分片上传事件。
MultipartUploadListing multipartUploadListing;
ListMultipartUploadsRequest listMultipartUploadsRequest = new ListMultipartUploadsRequest(bucketName);
// 设置每页列举的分片上传事件数目。
listMultipartUploadsRequest.setMaxUploads(50);

do {
multipartUploadListing = ossClient.listMultipartUploads(listMultipartUploadsRequest);

for (MultipartUpload multipartUpload : multipartUploadListing.getMultipartUploads()) {
// 获取uploadId。
```

```
multipartUpload.getUploadId();
// 获取Key。
multipartUpload.getKey();
// 获取分片上传的初始化时间。
multipartUpload.getInitiated();
}

listMultipartUploadsRequest.setKeyMarker(multipartUploadListing.getNextKeyMarker());
listMultipartUploadsRequest.setUploadIdMarker(multipartUploadListing.getNextUploadIdMarker());

} while (multipartUploadListing.isTruncated());

// 关闭OSSClient。
ossClient.shutdown();
```

上传进度条

进度条用于指示上传或下载的进度。下面的代码以ossClient.putObject方法为例，介绍如何使用进度条。

```
static class PutObjectProgressListener implements ProgressListener {
private long bytesWritten = 0;
private long totalBytes = -1;
private boolean succeed = false;

@Override
public void progressChanged(ProgressEvent progressEvent) {
long bytes = progressEvent.getBytes();
ProgressEventType eventType = progressEvent.getEventType();
switch (eventType) {
case TRANSFER_STARTED_EVENT:
System.out.println("Start to upload.....");
break;
case REQUEST_CONTENT_LENGTH_EVENT:
this.totalBytes = bytes;
System.out.println(this.totalBytes + " bytes in total will be uploaded to OSS");
break;z'z'z'za
case REQUEST_BYTE_TRANSFER_EVENT:
this.bytesWritten += bytes;
if (this.totalBytes != -1) {
int percent = (int)(this.bytesWritten * 100.0 / this.totalBytes);
System.out.println(bytes + " bytes have been written at this time, upload progress: " + percent + "%(" +
this.bytesWritten + "/" + this.totalBytes + ")");
} else {
System.out.println(bytes + " bytes have been written at this time, upload ratio: unknown" + "(" + this.bytesWritten
+ "/...)");
}
break;
case TRANSFER_COMPLETED_EVENT:
```

```
this.succeed = true;
System.out.println("Succeed to upload, " + this.bytesWritten + " bytes have been transferred in total");
break;
case TRANSFER_FAILED_EVENT:
System.out.println("Failed to upload, " + this.bytesWritten + " bytes have been transferred");
break;
default:
break;
}
}

public boolean isSucceed() {
return succeed;
}
}

public static void main(String[] args) {
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

try {
// 带进度条的上传。
ossClient.putObject(new PutObjectRequest(bucketName, objectName, new FileInputStream("<yourLocalFile>")).
<PutObjectRequest>withProgressListener(new PutObjectProgressListener()));

} catch (Exception e) {
e.printStackTrace();
}
// 关闭OSSClient。
ossClient.shutdown();
}
```

ossClient.putObject、ossClient.getObject和ossClient.uploadPart方法支持进度条功能。
ossClient.uploadFile和ossClient.downloadFile方法不支持进度条功能。

上传进度条的完整代码请参见GitHub。

上传回调

以下代码用于上传回调（callback）：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
// 您的回调服务器地址，如http://oss-demo.aliyuncs.com或http://127.0.0.1:9090。
String callbackUrl = "<yourCallbackServerUrl>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

String content = "Hello OSS";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, objectName,new
ByteArrayInputStream(content.getBytes()));

// 上传回调参数。
Callback callback = new Callback();
callback.setCallbackUrl(callbackUrl);
// Endpoint以杭州为例，其它Region请按实际情况填写
callback.setCallbackHost("oss-cn-hangzhou.aliyuncs.com");
callback.setCallbackBody("{\\\"mimeType\\\":${mimeType},\\\"size\\\":${size}}");
callback.setCallbackBodyType(CallbackBodyType.JSON);
callback.addCallbackVar("x:var1", "value1");
callback.addCallbackVar("x:var2", "value2");
putObjectRequest.setCallback(callback);

PutObjectResult putObjectResult = ossClient.putObject(putObjectRequest);

// 读取上传回调返回的消息内容。
byte[] buffer = new byte[1024];
putObjectResult.getCallbackResponseBody().read(buffer);
// 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
putObjectResult.getCallbackResponseBody().close();

// 关闭OSSClient。
ossClient.shutdown();
```

上传回调详情请参见开发指南中的**上传回调**。上传回调的完整代码请参见GitHub。

下载文件

概述

OSS Java SDK提供了丰富的文件下载方式：

- 流式下载
- 下载到本地文件
- 范围下载
- 断点续传下载
- 限定条件下载

下载过程中，您还可以通过**下载进度条**功能查看下载进度。

流式下载

如果要下载的文件太大，或者一次性下载耗时太长，您可以通过流式下载，一次处理部分内容，直到完成文件的下载。

OssObject对象使用完毕后必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。关闭方法如下：

```
OSSObject ossObject = ossClient.getObject(bucketName, objectName);
ossObject.close();
```

以下代码用于流式下载：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// ossObject包含文件所在的存储空间名称、文件名称、文件元信息以及一个输入流。
OSSObject ossObject = ossClient.getObject(bucketName, objectName);

// 读取文件内容。
System.out.println("Object content:");
BufferedReader reader = new BufferedReader(new InputStreamReader(ossObject.getObjectContent()));
while (true) {
String line = reader.readLine();
if (line == null) break;
```

```
System.out.println("\n" + line);
}
// 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
reader.close();

// 关闭OSSClient。
ossClient.shutdown();
```

流式下载的完整代码请参见GitHub。

下载到本地文件

以下代码用于把指定的OSS文件下载到本地文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 下载OSS文件到本地文件。如果指定的本地文件存在会覆盖，不存在则新建。
ossClient.getObject(new GetObjectRequest(bucketName, objectName), new File("<yourLocalFile>"));

// 关闭OSSClient。
ossClient.shutdown();
```

范围下载

如果仅需要文件中的部分数据，您可以使用范围下载，下载指定范围内的数据，代码如下：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
```

```
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objectName);
// 获取0~1000字节范围内的数据，包括0和1000，共1001个字节的数据。如果指定的范围无效（比如开始或结束位置的指定值为负数，或指定值大于文件大小），则下载整个文件。
getObjectRequest.setRange(0, 1000);

// 范围下载。
OSSObject ossObject = ossClient.getObject(getObjectRequest);

// 读取数据。
byte[] buf = new byte[1024];
InputStream in = ossObject.getObjectContent();
for (int n = 0; n != -1; ) {
n = in.read(buf, 0, buf.length);
}

// 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
in.close();

// 关闭OSSClient。
ossClient.shutdown();
```

流式读取一次可能无法读取全部数据。如果您需要流式读取64KB的数据，请使用如下的方式多次读取，直到读取到64KB或者文件结束。详情请参见InputStream.read。

```
byte[] buf = new byte[1024];
InputStream in = ossObject.getObjectContent();
for (int n = 0; n != -1; ) {
n = in.read(buf, 0, buf.length);
}
in.close();
```

断点续传下载

断点续传下载

当下载大文件时，如果网络不稳定或者程序异常退出，会导致下载失败，甚至重试多次仍无法完成下载。为此OSS提供了断点续传下载功能。

断点续传下载将需要下载的文件分成若干个分片分别下载，所有分片都下载完成后，将所有分片合并成完整的文件。

您可以通过ossClient.downloadFile方法实现断点续传下载。此方法的DownloadFileRequest请求包含以下参数：

参数	描述	是否必需	默认值	如何设置
bucket	存储空间名称。	是	无	通过构造方法设置。
key	OSS文件名称。	是	无	通过构造方法设置。
downloadFile	本地文件。OSS文件将下载到该文件。	否	OSS文件名称	通过构造方法或setDownloadFile设置。
partSize	分片大小，取值范围为1B~5GB。	否	文件大小/10000	通过setPartSize设置。
taskNum	分片下载的并发数。	否	1	通过setTaskNum设置。
enableCheckpoint	是否开启断点续传功能。	否	关闭	通过setEnableCheckpoint设置。
checkpointFile	记录本地分片下载结果的文件。开启断点续传功能时需要设置此参数。下载过程中的进度信息会保存在该文件中，如果某一分片下载失败，再次下载时会根据文件中记录的点继续下载。下载完成后，该文件会被删除。	否	downloadFile.ucp（与DownloadFile同目录）。	通过setCheckpointFile设置。

以下代码用于断点续传下载：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
```

```
// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 下载请求，10个任务并发下载，启动断点续传。
DownloadFileRequest downloadFileRequest = new DownloadFileRequest(bucketName, objectName);
downloadFileRequest.setDownloadFile("<yourDownloadFile>");
downloadFileRequest.setPartSize(1 * 1024 * 1024);
downloadFileRequest.setTaskNum(10);
downloadFileRequest.setEnableCheckpoint(true);
downloadFileRequest.setCheckpointFile("<yourCheckpointFile>");

// 下载文件。
DownloadFileResult downloadRes = ossClient.downloadFile(downloadFileRequest);
// 下载成功时，会返回文件元信息。
downloadRes.getObjectMeta();

// 关闭OSSClient。
ossClient.shutdown();
```

限定条件下载

下载文件时，可以指定一个或多个限定条件。满足限定条件则下载，不满足则返回错误，不下载。可以使用的限定条件如下：

参数	描述
If-Modified-Since	如果指定的时间早于实际修改时间，则正常传输文件，否则返回错误（304 Not modified）。
If-Unmodified-Since	如果指定的时间等于或者晚于文件实际修改时间，则正常传输文件，否则返回错误（412 Precondition failed）。
If-Match	如果指定的ETag和OSS文件的ETag匹配，则正常传输文件，否则返回错误（412 Precondition failed）。
If-None-Match	如果指定的ETag和OSS文件的ETag不匹配，则正常传输文件，否则返回错误（304 Not modified）。

If-Modified-Since和If-Unmodified-Since可以同时存在。If-Match和If-None-Match可以同时存在。

ETag可以通过ossClient.getObjectMeta方法获取。

以下代码用于限定条件下载：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
```

```
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

GetObjectRequest request = new GetObjectRequest(bucketName, objectName);
// 设置限定条件。
request.setModifiedSinceConstraint(new Date());

// 下载OSS文件到本地文件。
ossClient.getObject(request, new File("<yourLocalFile>"));

// 关闭OSSClient。
ossClient.shutdown();
```

ossClient.getObject和ossClient.downloadFile方法支持限定条件下载。

进度条

进度条用于指示上传或下载的进度。下面的代码以ossClient.getObject方法为例，介绍如何使用进度条。

```
static class GetObjectProgressListener implements ProgressListener {
private long bytesRead = 0;
private long totalBytes = -1;
private boolean succeed = false;

@Override
public void progressChanged(ProgressEvent progressEvent) {
long bytes = progressEvent.getBytes();
ProgressEventType eventType = progressEvent.getEventType();
switch (eventType) {
case TRANSFER_STARTED_EVENT:
System.out.println("Start to download.....");
break;
case RESPONSE_CONTENT_LENGTH_EVENT:
this.totalBytes = bytes;
System.out.println(this.totalBytes + " bytes in total will be downloaded to a local file");
break;
case RESPONSE_BYTE_TRANSFER_EVENT:
this.bytesRead += bytes;
if (this.totalBytes != -1) {
int percent = (int)(this.bytesRead * 100.0 / this.totalBytes);
System.out.println(bytes + " bytes have been read at this time, download progress: " +
percent + "%(" + this.bytesRead + "/" + this.totalBytes + ")");
```

```
} else {
System.out.println(bytes + " bytes have been read at this time, download ratio: unknown" +
"(" + this.bytesRead + "/...)");
}
break;
case TRANSFER_COMPLETED_EVENT:
this.succeed = true;
System.out.println("Succeed to download, " + this.bytesRead + " bytes have been transferred in total");
break;
case TRANSFER_FAILED_EVENT:
System.out.println("Failed to download, " + this.bytesRead + " bytes have been transferred");
break;
default:
break;
}
}

public boolean isSucceed() {
return succeed;
}
}

public static void main(String[] args) {
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

try {
// 带进度条的下载。
ossClient.getObject(new GetObjectRequest(bucketName, objectName).
<GetObjectRequest>withProgressListener(new GetObjectProgressListener()),
new File("<yourLocalFile>"));
} catch (Exception e) {
e.printStackTrace();
}
// 关闭OSSClient。
ossClient.shutdown();
}
```

ossClient.putObject、ossClient.getObject和ossClient.uploadPart方法支持进度条功能。
ossClient.uploadFile和ossClient.downloadFile方法不支持进度条功能。

下载进度条的完整代码请参见GitHub。

管理文件

概述

您可以通过一系列的接口管理存储空间（Bucket）下的文件（Object），包括以下操作：

- 判断文件是否存在
- 管理文件访问权限
- 管理文件元信息
- 列举文件
- 删除文件
- 拷贝文件
- 解冻归档文件
- 管理符号链接

判断文件是否存在

以下代码用于判断指定的文件是否存在：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 判断文件是否存在。
boolean found = ossClient.doesObjectExist("<yourBucketName>", "<yourObjectName>");
System.out.println(found);

// 关闭OSSClient。
ossClient.shutdown();
```

管理文件访问权限

文件的访问权限（ACL）有以下四种：

访问权限	描述
继承Bucket	文件遵循存储空间的访问权限。
私有	文件是私有资源。文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。
公共读私有写	文件是公共读资源。文件拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。
公共读写	文件是公共读写资源。所有用户都有该文件的读写权限。

文件的访问权限优先级高于存储空间的访问权限。例如存储空间的访问权限是私有读写，而文件的访问权限是公共读写，则所有用户都有该文件的读写权限。如果某个文件没有设置过访问权限，则遵循存储空间的访问权限。

设置文件访问权限

您可以通过ossClient.setObjectAcl设置文件的访问权限。

访问权限	访问权限值
继承Bucket	CannedAccessControlList.Default
私有	CannedAccessControlList.Private
公共读私有写	CannedAccessControlList.PublicRead
公共读写	CannedAccessControlList.PublicReadWrite

以下代码用于设置指定文件的访问权限：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 设置文件的访问权限。
ossClient.setObjectAcl("<yourBucketName>", "<yourObjectName>", CannedAccessControlList.PublicRead);
```

```
// 关闭OSSClient。
ossClient.shutdown();
```

获取文件访问权限

以下代码用于获取指定文件的访问权限：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 获取文件的访问权限。
ObjectAcl objectAcl = ossClient.getObjectAcl("<yourBucketName>", "<yourObjectName>");
System.out.println(objectAcl.getPermission().toString());

// 关闭OSSClient。
ossClient.shutdown();
```

管理文件元信息

文件元信息（Object Meta）包括HTTP header和自定义元信息，详情请参见开发指南中的文件元信息。

设置元信息

设置HTTP header

以下代码用于设置HTTP header：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

String content = "Hello OSS";

// 创建上传文件的元信息，可以通过文件元信息设置HTTP header。
```

```
ObjectMetadata meta = new ObjectMetadata();

String md5 = BinaryUtil.toBase64String(BinaryUtil.calculateMd5(content.getBytes()));
// 开启文件内容MD5校验。开启后OSS会把您提供的MD5与文件的MD5比较，不一致则抛出异常。
meta.setContentMD5(md5);
// 指定上传的内容类型。内容类型决定浏览器将以什么形式、什么编码读取文件。如果没有指定则根据文件的扩展名生成，如果没有扩展名则为默认值application/octet-stream。
meta.setContentType("text/plain");
// 设置内容被下载时的名称。
meta.setContentDisposition("Download File Name");
// 设置上传文件的长度。如超过此长度，则会被截断，为设置的长度。如不足，则为上传文件的实际长度。
meta.setContentLength(content.length());
// 设置内容被下载时网页的缓存行为。
meta.setCacheControl("Download Action");
// 设置缓存过期时间，格式是格林威治时间（GMT）。
meta.setExpirationTime(DateUtil.parseIso8601Date("2022-10-12T00:00:00.000Z"));
// 设置内容被下载时的编码格式。
meta.setContentEncoding("utf-8");
// 设置header。
meta.setHeader("<yourHeader>", "<yourHeaderValue>");
// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 上传文件。
ossClient.putObject("<yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(content.getBytes()), meta);

// 关闭OSSClient。
ossClient.shutdown();
```

HTTP header详情请参见RFC2616。

设置自定义元信息

您可以自定义文件的元信息来对文件进行描述。

以下代码用于设置文件的自定义元信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

String content = "Hello OSS";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 创建文件元信息。
ObjectMetadata meta = new ObjectMetadata();

// 设置自定义元信息property值为property-value。
```

```
meta.addUserMetadata("property", "property-value");

// 上传文件。
ossClient.putObject("<yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(content.getBytes()),
meta);

// 获取文件元信息。
ossClient.getObjectMetadata("<yourBucketName>", "<yourObjectName>");

// 关闭OSSClient。
ossClient.shutdown();
```

下载文件时，文件元信息也会同时下载。 一个文件可以有多个元信息，总大小不能超过8KB。

修改元信息

以下代码用于修改文件的元信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String sourceBucketName = "<yourSourceBucketName>";
String sourceObjectName = "<yourSourceObjectName>";
String destinationBucketName = "<yourDestinationBucketName>";
String destinationObjectName = "<yourDestinationObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId,accessKeySecret);

// 设置源文件与目标文件相同，调用ossClient.copyObject方法修改文件元信息。
CopyObjectRequest request = new CopyObjectRequest(sourceBucketName, sourceObjectName,
destinationBucketName, destinationObjectName);

ObjectMetadata meta = new ObjectMetadata();
// 指定上传的内容类型。内容类型决定浏览器将以什么形式、什么编码读取文件。如果没有指定则根据文件的扩展名生成，如
果没有扩展名则为默认值application/octet-stream。
meta.setContentType("text/plain");
// 设置内容被下载时的名称。
meta.setContentDisposition("Download File Name");
// 设置内容被下载时网页的缓存行为。
meta.setCacheControl("Download Action");
// 设置缓存过期时间，格式是格林威治时间（GMT）。
meta.setExpirationTime(DateUtil.parseIso8601Date("2022-10-12T00:00:00.000Z"));
// 设置内容被下载时的编码格式。
meta.setContentEncoding("utf-8");
// 设置header。
meta.setHeader("<yourHeader>", "<yourHeaderValue>");
// 设置自定义元信息property值为property-value。
meta.addUserMetadata("property", "property-value");
request.setNewObjectMetadata(meta);
```

```
//修改元信息。
ossClient.copyObject(request);

// 关闭OSSClient。
ossClient.shutdown();
```

获取文件元信息

您可以通过以下两种方法获取文件元信息：

方法	描述	优势
ossClient.getSimplifiedObjectMeta	获取文件的ETag、Size（文件大小）、 LastModified（最后修改时间）。	更轻量、更快
ossClient.getObjectMetadata	获取文件的全部元信息。	无

以下代码用于获取文件元信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 获取文件的部分元信息。
SimplifiedObjectMeta objectMeta = ossClient.getSimplifiedObjectMeta("<yourBucketName>", "<yourObjectName>");
System.out.println(objectMeta.getSize());
System.out.println(objectMeta.getETag());
System.out.println(objectMeta.getLastModified());

// 获取文件的全部元信息。
ObjectMetadata metadata = ossClient.getObjectMetadata("<yourBucketName>", "<yourObjectName>");
System.out.println(metadata.getContentType());
System.out.println(metadata.getLastModified());
System.out.println(metadata.getExpirationTime());

// 关闭OSSClient。
ossClient.shutdown();
```

列举文件

OSS文件按照字母顺序排列。您可以通过ossClient.listObjects列出存储空间下的文件。listObjects有以下三类参数格式：

- ObjectListing listObjects(String bucketName)：列举存储空间下的文件。最多列举100个文件。
- ObjectListing listObjects(String bucketName, String prefix)： 列举存储空间下指定前缀的文件。最多列举100个文件。
- ObjectListing listObjects(ListObjectsRequest listObjectsRequest)：提供多种过滤功能，实现灵活的查询功能。

ObjectListing的参数如下：

参数	描述	方法
objectSummaries	限定返回的文件元信息。	List<OSSObjectSummary> getObjectSummaries()
prefix	本次查询结果的前缀。	String getPrefix()
delimiter	对文件名称进行分组的一个字符。	String getDelimiter()
marker	标明本次列举文件的起点。	String getMarker()
maxKeys	列举文件的最大个数。	int getMaxKeys()
nextMarker	下一次列举文件的起点。	String getNextMarker()
isTruncated	指明是否所有的结果都已经返回。	boolean isTruncated()
commonPrefixes	以delimiter结尾，并有共同前缀的文件集合。	List<String> getCommonPrefixes()
encodingType	指明返回结果中编码使用的类型。	String getEncodingType()

listObjects的完整代码请参见GitHub。

简单列举文件

以下代码用于列举指定存储空间下的文件，默认列举100个文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String KeyPrifex = "<yourKeyPrifex>";
```

```
// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 列举文件。 如果不设置KeyPrifex，则列举存储空间下所有的文件。如果设置了KeyPrifex，则列举包含指定前缀的文件。
ObjectListing objectListing = ossClient.listObjects(bucketName, KeyPrifex);
List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
for (OSSObjectSummary s : sums) {
System.out.println("\t" + s.getKey());
}

// 关闭OSSClient。
ossClient.shutdown();
```

通过ListObjectsRequest列举文件

通过设置ListObjectsReques的参数实现各种灵活的查询功能。ListObjectsReques的参数如下：

参数	描述	方法
prefix	限定返回的文件必须以prefix作为前缀。	setPrefix(String prefix)
delimiter	对文件名称进行分组的一个字符。所有名称包含指定的前缀且第一次出现delimiter字符之间的文件作为一组元素（commonPrefixes）。	setDelimiter(String delimiter)
marker	列举指定marker之后的文件。	setMarker(String marker)
maxKeys	限定此次列举文件的最大个数。默认值为100，最大值为1000。	setMaxKeys(Integer maxKeys)
encodingType	请求响应体中文件名称采用的编码方式，目前仅支持url。	setEncodingType(String encodingType)

列举指定个数的文件

以下代码用于列举指定个数的文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 设置最大个数。
final int maxKeys = 200;
```

```
// 列举文件。
ObjectListing objectListing = ossClient.listObjects(new ListObjectsRequest(bucketName).withMaxKeys(maxKeys));
List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
for (OSSObjectSummary s : sums) {
System.out.println("\t" + s.getKey());
}

// 关闭OSSClient。
ossClient.shutdown();
```

列举指定前缀的文件

以下代码用于列举包含指定前缀（prefix）的文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 指定前缀。
final String keyPrefix = "<yourkeyPrefix>";

// 列举包含指定前缀的文件。默认列举100个文件。
ObjectListing objectListing = ossClient.listObjects(new ListObjectsRequest(bucketName).withPrefix(keyPrefix));
List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
for (OSSObjectSummary s : sums) {
System.out.println("\t" + s.getKey());
}

// 关闭OSSClient。
ossClient.shutdown();
```

列举指定marker之后的文件

参数marker代表文件名称。以下代码用于列举指定marker之后的文件:

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
```

```
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);
final String marker = "<yourMarker>";

// 列举指定marker之后的文件。默认列举100个文件。
ObjectListing objectListing = ossClient.listObjects(new ListObjectsRequest(bucketName).withMarker(marker));
List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
for (OSSObjectSummary s : sums) {
System.out.println("\t" + s.getKey());
}

// 关闭OSSClient。
ossClient.shutdown();
```

分页列举所有文件

以下代码用于分页列举指定存储空间下的所有文件。每页列举的文件个数通过maxKeys指定。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

final int maxKeys = 200;
String nextMarker = null;
ObjectListing objectListing;

do {
objectListing = ossClient.listObjects(new
ListObjectsRequest(bucketName).withMarker(nextMarker).withMaxKeys(maxKeys));

List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
for (OSSObjectSummary s : sums) {
System.out.println("\t" + s.getKey());
}

nextMarker = objectListing.getNextMarker();

} while (objectListing.isTruncated());

// 关闭OSSClient。
ossClient.shutdown();
```

分页列举指定marker之后的文件

以下代码用于分页列举指定maker之后的文件。每页列举的文件个数通过maxKeys指定。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

final int maxKeys = 200;
String nextMarker = "<yourNextMarker>";
ObjectListing objectListing;

do {
objectListing = ossClient.listObjects(new
ListObjectsRequest(bucketName).withMarker(nextMarker).withMaxKeys(maxKeys));

List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
for (OSSObjectSummary s : sums) {
System.out.println("\t" + s.getKey());
}

nextMarker = objectListing.getNextMarker();

} while (objectListing.isTruncated());

// 关闭OSSClient。
ossClient.shutdown();
```

分页列举指定前缀的文件

以下代码用于分页列举包含指定前缀的文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 指定每页200个文件。
final int maxKeys = 200;
final String keyPrefix = "<yourkeyPrefix>";
String nextMarker = "<yourNextMarker>";
ObjectListing objectListing;

do {
```

```
objectListing = ossClient.listObjects(new ListObjectsRequest(bucketName).
withPrefix(keyPrefix).withMarker(nextMarker).withMaxKeys(maxKeys));

List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
for (OSSObjectSummary s : sums) {
System.out.println("\t" + s.getKey());
}

nextMarker = objectListing.getNextMarker();

} while (objectListing.isTruncated());

// 关闭OSSClient。
ossClient.shutdown();
```

指定文件名称编码

如果文件名称含有以下特殊字符，需要进行编码传输。OSS目前仅支持url编码。

- 单引号（’ ）
- 双引号（” ）
- and符号（& ）
- 尖括号（< >）
- 顿号（、 ）
- 中文

以下代码用于指定文件名称编码：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

final int maxKeys = 200;
final String keyPrefix = "<yourkeyPrefix>";
String nextMarker = "<yourNextMarker>";
ObjectListing objectListing;

do {
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);
listObjectsRequest.setPrefix(keyPrefix);
listObjectsRequest.setMaxKeys(maxKeys);
listObjectsRequest.setMarker(nextMarker);

// 指定文件名称编码。
listObjectsRequest.setEncodingType("url");
```

```
objectListing = ossClient.listObjects(listObjectsRequest);

// 文件解码。
for (OSSObjectSummary objectSummary: objectListing.getObjectSummaries()) {
System.out.println("Key:" + URLDecoder.decode(objectSummary.getKey(), "UTF-8"));
}

// commonPrefixes解码。
for (String commonPrefixes: objectListing.getCommonPrefixes()) {
System.out.println("CommonPrefixes:" + URLDecoder.decode(commonPrefixes, "UTF-8"));
}

// nextMarker解码。
if (objectListing.getNextMarker() != null) {
nextMarker = URLDecoder.decode(objectListing.getNextMarker(), "UTF-8");
}
} while (objectListing.isTruncated());

// 关闭OSSClient。
ossClient.shutdown();
```

文件夹功能

OSS没有文件夹的概念，所有元素都是以文件来存储。创建文件夹本质上来说是创建了一个大小为0并以正斜线（/）结尾的文件。这个文件可以被上传和下载，控制台会对以正斜线（/）结尾的文件以文件夹的方式展示。

通过delimiter和prefix两个参数可以模拟文件夹功能：

- 如果设置prefix为某个文件夹名称，则会列举以此prefix开头的文件，即该文件夹下所有的文件和子文件夹（目录），显示为Objects。
- 如果再设置delimiter为正斜线（/），则只列举该文件夹下的文件和子文件夹（目录），该文件夹下的子文件夹（目录）则显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

文件夹详情请参见文件夹功能。创建文件的完整代码请参见GitHub。

假设存储空间中有4个文件： oss.jpg、fun/test.jpg、fun/movie/001.avi、fun/movie/007.avi，正斜线（/）作为文件夹的分隔符。下面的示例展示了如何模拟文件夹功能。

列举存储空间下所有文件

以下代码用于列举指定存储空间下的所有文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
```

```
// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 构造ListObjectsRequest请求。
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);

// 列出文件。
ObjectListing listing = ossClient.listObjects(listObjectsRequest);

// 遍历所有文件。
System.out.println("Objects:");
for (OSSObjectSummary objectSummary : listing.getObjectSummaries()) {
System.out.println(objectSummary.getKey());
}

// 遍历所有commonPrefix。
System.out.println("CommonPrefixes:");
for (String commonPrefix : listing.getCommonPrefixes()) {
System.out.println(commonPrefix);
}

// 关闭OSSClient。
ossClient.shutdown();
```

返回结果如下：

```
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
oss.jpg

CommonPrefixes:
```

列举指定目录下所有文件

以下代码用于列举指定目录下的所有文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 构造ListObjectsRequest请求。
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);
// 设置prefix参数来获取fun目录下的所有文件。
```

```
listObjectsRequest.setPrefix("fun/");

// 递归列出fun目录下的所有文件。
ObjectListing listing = ossClient.listObjects(listObjectsRequest);

// 遍历所有文件。
System.out.println("Objects:");
for (OSSObjectSummary objectSummary : listing.getObjectSummaries()) {
System.out.println(objectSummary.getKey());
}

// 遍历所有commonPrefix。
System.out.println("\nCommonPrefixes:");
for (String commonPrefix : listing.getCommonPrefixes()) {
System.out.println(commonPrefix);
}

// 关闭OSSClient。
ossClient.shutdown();
```

返回结果如下：

```
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg

CommonPrefixes:
```

列举目录下的文件和子目录

以下代码用于列举指定目录下的文件和子目录：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 构造ListObjectsRequest请求。
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);

// 设置正斜线（/）为文件夹的分隔符。
listObjectsRequest.setDelimiter("/");

// 列出fun目录下的所有文件和文件夹。
listObjectsRequest.setPrefix("fun/");
```

```
ObjectListing listing = ossClient.listObjects(listObjectsRequest);

// 遍历所有文件。
System.out.println("Objects:");
// objectSummaries的列表中给出的是fun目录下的文件。
for (OSSObjectSummary objectSummary : listing.getObjectSummaries()) {
System.out.println(objectSummary.getKey());
}

// 遍历所有commonPrefix。
System.out.println("\nCommonPrefixes:");
// commonPrefixs列表中给出的是fun目录下的所有子文件夹。fun/movie/001.avi和fun/movie/007.avi两个文件没有被列出来，因为它们属于fun文件夹下的movie目录。
for (String commonPrefix : listing.getCommonPrefixes()) {
System.out.println(commonPrefix);
}

// 关闭OSSClient。
ossClient.shutdown();
```

返回结果如下：

```
Objects:
fun/test.jpg

CommonPrefixes:
fun/movie/
```

删除文件

删除单个文件

以下代码用于删除单个文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);
```

```
// 删除文件。
ossClient.deleteObject(bucketName, objectName);

// 关闭OSSClient。
ossClient.shutdown();
```

删除多个文件

每次最多删除1000个文件。有两种返回模式：详细（verbose）模式和简单（quiet）模式：

- 详细模式：返回删除成功的文件列表。默认为详细模式。
- 简单模式：返回删除失败的文件列表。

DeleteObjectsRequest的参数如下：

参数	描述	方法
Keys	需要删除的文件。	setKeys(List<String>)
quiet	返回模式。true表示简单模式，false表示详细模式。默认为详细模式。	setQuiet(boolean)
encodingType	指定对返回的文件名称进行编码，目前仅支持url。	setEncodingType(String)

DeleteObjectsResult的参数如下：

参数	描述	方法
deletedObjects	删除结果。详细模式下为删除成功的文件列表，简单模式下为删除失败的文件列表。	List<String> getDeletedObjects()
encodingType	deletedObjects中文件名称的编码，为空表示没有编码。	getEncodingType()

以下代码用于批量删除文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 删除文件。
List<String> keys = new ArrayList<String>();
```

```
keys.add("key0");
keys.add("key1");
keys.add("key2");

DeleteObjectsResult deleteObjectsResult = ossClient.deleteObjects(new
DeleteObjectsRequest(bucketName).withKeys(keys));
List<String> deletedObjects = deleteObjectsResult.getDeletedObjects();

// 关闭OSSClient。
ossClient.shutdown();
```

批量删除文件的完整代码请参见GitHub。

拷贝文件

拷贝小文件

对于小于1GB的文件，您可以通过ossClient.copyObject方法将文件从一个存储空间（源存储空间）复制到同一地域的另一个存储空间（目标存储空间）。此方法有两种参数指定方式：

参数指定方式	描述
CopyObjectResult copyObject(String sourceBucketName, String sourceKey, String destinationBucketName, String destinationKey)	允许指定源存储空间和源文件，以及目标存储空间和目标文件。拷贝后，目标文件的内容及元信息与源文件相同，称为简单拷贝。
CopyObjectResult copyObject(CopyObjectRequest copyObjectRequest)	允许指定目标文件的元信息和拷贝的限制条件。如果拷贝操作的源文件地址和目标文件地址相同，则直接替换源文件的元信息。

CopyObjectRequest可设置的参数如下：

参数	描述	方法
sourceBucketName	源存储空间名称。	setSourceBucketName(String sourceBucketName)
sourceKey	源文件名称。	setSourceKey(String sourceKey)
destinationBucketName	目标存储空间名称。	setDestinationBucketName(String destinationBucketName)
destinationKey	目标文件名称。	setDestinationKey(String destinationKey)
newObjectMetadata	目标文件元信息。	setNewObjectMetadata(ObjectMetadata

		newObjectMetadata)
matchingETagConstraints	拷贝的限制条件。如果源文件的ETag值和提供的ETag值相等，则执行拷贝操作，否则返回错误。	setMatchingETagConstraints(List<String> matchingETagConstraints)
nonmatchingEtagConstraints	拷贝的限制条件。如果源文件的ETag值和用户提供的ETag值不相等，则执行拷贝操作，否则返回错误。	setNonmatchingETagConstraints(List<String> nonmatchingEtagConstraints)
unmodifiedSinceConstraint	拷贝的限制条件。如果传入参数中的时间等于或者晚于文件实际修改时间，则执行拷贝操作，否则返回错误。	setUnmodifiedSinceConstraint(Date unmodifiedSinceConstraint)
modifiedSinceConstraint	拷贝的限制条件。如果源文件在指定的时间之后被修改过，则执行拷贝操作，否则返回错误。	setModifiedSinceConstraint(Date modifiedSinceConstraint)

CopyObjectRequest的参数如下：

参数	**描述**	**方法**
etag	OSS文件唯一性标志	String getETag()
lastModified	文件最后修改时间	Date getLastModified()

此操作的限制条件如下：

- 用户有源文件的读写权限。
- 不支持跨地域拷贝。例如，不支持将杭州存储空间里的文件拷贝到青岛。
- 文件的大小不能超过1GB。

简单拷贝

以下代码用于简单拷贝：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

String sourceBucketName = "<yourSourceBucketName>";
String sourceObjectName = "<yourSourceObjectName>";
String destinationBucketName = "<yourDestinationBucketName>";
String destinationObjectName = "<yourDestinationObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);
```

```
// 拷贝文件。
CopyObjectResult result = ossClient.copyObject(sourceBucketName, sourceObjectName, destinationBucketName,
destinationObjectName);
System.out.println("ETag: " + result.getETag() + " LastModified: " + result.getLastModified());

// 关闭OSSClient。
ossClient.shutdown();
```

通过CopyObjectRequest拷贝

以下代码用于通过CopyObjectRequest拷贝文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

String sourceBucketName = "<yourSourceBucketName>";
String sourceObjectName = "<yourSourceObjectName>";
String destinationBucketName = "<yourDestinationBucketName>";
String destinationObjectName = "<yourDestinationObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 创建CopyObjectRequest对象。
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(sourceBucketName, sourceObjectName,
destinationBucketName, destinationObjectName);

// 设置新的文件元信息。
ObjectMetadata meta = new ObjectMetadata();
meta.setContentType("text/html");
copyObjectRequest.setNewObjectMetadata(meta);

// 复制文件。
CopyObjectResult result = ossClient.copyObject(copyObjectRequest);
System.out.println("ETag: " + result.getETag() + " LastModified: " + result.getLastModified());

// 关闭OSSClient。
ossClient.shutdown();
```

拷贝大文件

对于大于1GB的文件，需要使用分片拷贝（UploadPartCopy）。分片拷贝分为三步：

1. 通过ossClient.initiateMultipartUpload初始化分片拷贝任务。
2. 通过ossClient.uploadPartCopy进行分片拷贝。除最后一个分片外，其它分片都要大于100KB。
3. 通过ossClient.completeMultipartUpload提交分片拷贝任务。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

String sourceBucketName = "<yourSourceBucketName>";
String sourceObjectName = "<yourSourceObjectName>";
String destinationBucketName = "<yourDestinationBucketName>";
String destinationObjectName = "<yourDestinationObjectName>";

ObjectMetadata objectMetadata = ossClient.getObjectMetadata(sourceBucketName, sourceObjectName);
// 获取被拷贝文件的大小。
long contentLength = objectMetadata.getContentLength();

// 设置分片大小为10MB。
long partSize = 1024 * 1024 * 10;

// 计算分片总数。
int partCount = (int) (contentLength / partSize);
if (contentLength % partSize != 0) {
partCount++;
}
System.out.println("total part count:" + partCount);

// 初始化拷贝任务。可以通过InitiateMultipartUploadRequest指定目标文件元信息。
InitiateMultipartUploadRequest initiateMultipartUploadRequest = new
InitiateMultipartUploadRequest(destinationBucketName, destinationObjectName);
InitiateMultipartUploadResult initiateMultipartUploadResult =
ossClient.initiateMultipartUpload(initiateMultipartUploadRequest);
String uploadId = initiateMultipartUploadResult.getUploadId();

// 分片拷贝。
List<PartETag> partETags = new ArrayList<PartETag>();
for (int i = 0; i < partCount; i++) {
// 计算每个分片的大小。
long skipBytes = partSize * i;
long size = partSize < contentLength - skipBytes ? partSize : contentLength - skipBytes;

// 创建UploadPartCopyRequest。可以通过UploadPartCopyRequest指定限定条件。
UploadPartCopyRequest uploadPartCopyRequest =
new UploadPartCopyRequest(sourceBucketName, sourceObjectName, destinationBucketName,
destinationObjectName);
uploadPartCopyRequest.setUploadId(uploadId);
uploadPartCopyRequest.setPartSize(size);
uploadPartCopyRequest.setBeginIndex(skipBytes);
uploadPartCopyRequest.setPartNumber(i + 1);
UploadPartCopyResult uploadPartCopyResult = ossClient.uploadPartCopy(uploadPartCopyRequest);

// 将返回的分片ETag保存到partETags中。
partETags.add(uploadPartCopyResult.getPartETag());
}

// 提交分片拷贝任务。
```

```
CompleteMultipartUploadRequest completeMultipartUploadRequest = new CompleteMultipartUploadRequest(
destinationBucketName, destinationObjectName, uploadId, partETags);
ossClient.completeMultipartUpload(completeMultipartUploadRequest);

// 关闭OSSClient。
ossClient.shutdown();
```

分片拷贝的详细说明请参见UploadPartCopy。分片拷贝的完整代码请参见GitHub。

解冻归档文件

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。非归档类型的文件，不要调用restoreObject方法。

归档文件的状态变换过程如下：

1. 归档类型的文件初始时处于冷冻状态。
2. 提交解冻操作后，服务端执行解冻，文件处于解冻中的状态。
3. 完成解冻后，可以读取文件。解冻状态默认持续1天，最多延长7天，之后文件又回到冷冻状态。

以下代码用于解冻归档文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

ObjectMetadata objectMetadata = ossClient.getObjectMetadata(bucketName, objectName);

// 校验文件是否为归档文件。
StorageClass storageClass = objectMetadata.getObjectStorageClass();
if (storageClass == StorageClass.Archive) {
// 解冻文件。
ossClient.restoreObject(bucketName, objectName);

// 等待解冻完成。
do {
Thread.sleep(1000);
objectMetadata = ossClient.getObjectMetadata(bucketName, objectName);
} while (!objectMetadata.isRestoreCompleted());
```

```
}

// 获取解冻文件。
OSSObject ossObject = ossClient.getObject(bucketName, objectName);
ossObject.getObjectContent().close();

// 关闭OSSClient。
ossClient.shutdown();
```

归档存储类型的详细说明请参见存储类型介绍。解冻归档文件的完整代码请参见GitHUb。

管理符号链接

创建符号链接

符号链接是一种特殊的文件，它指向某个具体的文件，类似于Windows上使用的快捷方式。符号链接支持自定义元信息。

以下代码用于创建符号链接：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String symLink = "<yourSymLink>";
String destinationObjectName = "<yourDestinationObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 创建上传文件元信息。
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentType("text/plain");
// 设置自定义元信息property的值为property-value。
metadata.addUserMetadata("property", "property-value");

// 创建CreateSymlinkRequest。
CreateSymlinkRequest createSymlinkRequest = new CreateSymlinkRequest(bucketName, symLink, destinationObjectName);

// 设置元信息。
createSymlinkRequest.setMetadata(metadata);

// 创建符号链接。
```

```
ossClient.createSymlink(createSymlinkRequest);

// 关闭OSSClient。
ossClient.shutdown();
```

符号链接的详细信息请参见PutSymlink。

获取符号链接指向的文件内容

获取符号链接要求您对该符号链接有读权限。以下代码用于获取符号链接指向的文件内容：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String symLink = "<yourSymLink>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 获取符号链接。
OSSSymlink symbolicLink = ossClient.getSymlink(bucketName, symLink);
// 打印符号链接指向的文件内容。
System.out.println(symbolicLink.getSymlink());
System.out.println(symbolicLink.getTarget());
System.out.println(symbolicLink.getRequestId());

// 关闭OSSClient。
ossClient.shutdown();
```

符号链接的详细信息请参见GetSymlink。

数据安全性

Java SDK提供了数据完整性校验、服务器端加密、客户端加密等方法，来保证您在上传、下载和拷贝过程中数据的安全性。

数据完整性校验

Java SDK提供基于MD5和CRC的数据校验，确保上传、下载和拷贝过程中的数据完整性。

MD5

如果上传时设置了Content-MD5，OSS会根据接收的内容计算MD5。OSS计算的MD5值和上传提供的MD5值不一致时，则返回InvalidDigest。从而保证数据的完整性。

以下代码用于上传时进行MD5校验：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 上传字符串。
String content = "Hello OSS";

ObjectMetadata meta = new ObjectMetadata();
// 设置MD5校验。
String md5 = BinaryUtil.toBase64String(BinaryUtil.calculateMd5(content.getBytes()));
meta.setContentMD5(md5);

ossClient.putObject("<yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(content.getBytes()), meta);

// 关闭OSSClient。
ossClient.shutdown();
```

说明：putObject、getObject、appendObject、postObject、uploadPart支持MD5校验。

CRC64

上传、下载和拷贝文件时默认开启CRC数据校验，确保数据的完整性。

以下代码用于上传时查看CRC数据校验是否生效：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 上传字符串。
String content = "Hello OSS";
```

```
PutObjectResult result = ossClient.putObject("<yourBucketName>", "<yourObjectName>", new
ByteArrayInputStream(content.getBytes()));

// 判断是否开启CRC校验，true为开启，false为关闭。
if (ossClient.getClientConfiguration().isCrcCheckEnabled()) {
// 通过响应结果查看CRC校验是否生效。
OSSUtils.checkChecksum(result.getClientCRC(), result.getServerCRC(), result.getRequestId());
}
// 关闭OSSClient。
ossClient.shutdown();
```

说明：

- putObject、getObject、appendObject、uploadPart支持CRC64校验。
- CRC64校验会占用一定的CPU，上传、下载速度会有影响。

服务器端加密

上传文件时，设置元信息中serverSideEncryption参数为AES256或KMS，即可实现该文件在服务器端加密存储。更多信息请参见开发指南中的服务器端加密编码。

以下代码用于上传时设置服务器端加密：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 上传字符串。
String content = "Hello OSS";

ObjectMetadata meta = new ObjectMetadata();
// 指定此文件在服务器端的加密编码为AES256。
meta.setServerSideEncryption(ObjectMetadata.AES_256_SERVER_SIDE_ENCRYPTION);

ossClient.putObject("<yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(content.getBytes()),
meta);

// 关闭OSSClient。
ossClient.shutdown();
```

客户端加密

如需客户端加密，您可将数据在本地加密，然后上传到OSS，并在下载到本地后解密。详情请参见开发指南中

的客户端加密SDK介绍。

授权访问

使用STS进行临时授权

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。STS更详细的解释请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。

使用STS访问OSS的流程请参见开发指南中的RAM和STS应用场景实践。

使用STS凭证构造签名请求

以下代码用于使用STS凭证构造签名请求：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String securityToken = "<yourSecurityToken>";

// 用户拿到STS临时凭证后，通过其中的安全令牌（SecurityToken）和临时访问密钥（AccessKeyId和AccessKeySecret）生成OSSClient。
// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret, securityToken);

// OSS操作。

// 关闭OSSClient。
ossClient.shutdown();
```

使用签名URL进行临时授权

生成签名URL

您可以将生成的签名URL提供给访客进行临时访问。生成签名URL时，您可以指定URL的过期时间，来限制访客的访问时长。

生成以GET方法访问的签名URL

以下代码用于生成以GET方法访问的签名URL：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 设置URL过期时间为1小时。
Date expiration = new Date(new Date().getTime() + 3600 * 1000);
// 生成以GET方法访问的签名URL，访客可以直接通过浏览器访问相关内容。
URL url = ossClient.generatePresignedUrl(bucketName, objectName, expiration);

// 关闭OSSClient。
ossClient.shutdown();
```

生成以其他HTTP方法访问的签名URL

如果您想允许访客临时进行其他操作（比如上传或删除文件），需要生成对应的签名URL，例如：

```
// 生成以PUT方法访问的签名URL。
URL url = ossClient.generatePresignedUrl(bucketName, objectName, expiration, HttpMethod.PUT);
```

通过传入HttpMethod.PUT参数，访客可以使用生成的签名URL上传文件。

生成指定参数的签名URL

以下代码用于生成指定参数的签名URL：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
```

```
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 创建请求。
GeneratePresignedUrlRequest generatePresignedUrlRequest = new GeneratePresignedUrlRequest(bucketName,
objectName);
// HttpMethod为PUT。
generatePresignedUrlRequest.setMethod(HttpMethod.PUT);
// 添加用户自定义元信息。
generatePresignedUrlRequest.addUserMetadata("author", "baymax");
// 添加Content-Type。
generatePresignedUrlRequest.setContentType("application/octet-stream");
// 设置URL过期时间为1小时。
Date expiration = new Date(new Date().getTime() + 3600 * 1000);
generatePresignedUrlRequest.setExpiration(expiration);
// 生成签名URL。
URL url = ossClient.generatePresignedUrl(generatePresignedUrlRequest);

// 关闭OSSClient。
ossClient.shutdown();
```

使用签名URL上传/获取文件

使用签名URL获取文件

以下代码用于使用签名URL获取指定文件：

```


// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

Date expiration = DateUtil.parseRfc822Date("Wed, 18 Mar 2022 14:20:00 GMT");
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(bucketName, objectName,
HttpMethod.GET);
// 设置过期时间。
request.setExpiration(expiration);
// 生成签名URL（HTTP GET请求）。
URL signedUrl = ossClient .generatePresignedUrl(request);
System.out.println("signed url for getObject: " + signedUrl);

// 使用签名URL发送请求。
```

```
Map<String, String> customHeaders = new HashMap<String, String>();
// 添加GetObject请求头。
customHeaders.put("Range", "bytes=100-1000");
OSSObject object = ossClient.getObject(signedUrl,customHeaders);

// 关闭OSSClient。
ossClient.shutdown();
```

使用签名URL上传文件

以下代码用于使用签名URL上传文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 生成签名URL。
Date expiration = DateUtil.parseRfc822Date("Thu, 19 Mar 2019 18:00:00 GMT");
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(bucketName, objectName,
HttpMethod.PUT);
// 设置过期时间。
request.setExpiration(expiration);
// 设置Content-Type。
request.setContentType("application/octet-stream");
// 添加用户自定义元信息。
request.addUserMetadata("author", "aliy");
// 生成签名URL（HTTP PUT请求）。
URL signedUrl = ossClient.generatePresignedUrl(request);
System.out.println("signed url for putObject: " + signedUrl);

// 使用签名URL发送请求。
File f = new File("<yourLocalFile>");
FileInputStream fin = new FileInputStream(f);
// 添加PutObject请求头。
Map<String, String> customHeaders = new HashMap<String, String>();
customHeaders.put("Content-Type", "application/octet-stream");
customHeaders.put("x-oss-meta-author", "aliy");

PutObjectResult result = ossClient.putObject(signedUrl, fin, f.length(), customHeaders);

// 关闭OSSClient。
ossClient.shutdown();
```

生命周期管理

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。每条规则包含：

- 规则ID。用于标识一条规则，同一存储空间内规则ID不能重复。
- 策略。有以下两种设置方式。同一存储空间内仅支持一种设置方式。
    - 按前缀匹配。此种方式允许创建多条规则，前缀不能重复。
    - 配置到整个存储空间。此种方式只能创建一条规则。
- 过期时间。有两种指定方式：
    - 指定距文件最后修改时间N天后过期。
    - 指定日期创建前的文件过期，之后的不过期。
- 是否生效。

通过uploadPart方法上传的分片也支持设置生命周期规则。文件最后修改时间以初始化分片上传事件的时间为准。

更多关于生命周期的内容请参见管理对象生命周期。

设置生命周期规则

以下代码用于设置生命周期规则：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

SetBucketLifecycleRequest request = new SetBucketLifecycleRequest(bucketName);

// 设置规则ID和文件前缀。
String ruleId0 = "rule0";
String matchPrefix0 = "A0/";
String ruleId1 = "rule1";
String matchPrefix1 = "A1/";
String ruleId2 = "rule2";
String matchPrefix2 = "A2/";
String ruleId3 = "rule3";
String matchPrefix3 = "A3/";

// 距最后修改时间3天后过期。
```

```
request.AddLifecycleRule(new LifecycleRule(ruleId0, matchPrefix0, RuleStatus.Enabled, 3));

// 指定日期之前创建的文件过期。
LifecycleRule rule = new LifecycleRule(ruleId1, matchPrefix1, RuleStatus.Enabled);
rule.setCreatedBeforeDate(DateUtil.parseISO8601Date("2022-10-12T00:00:00.000Z"));
request.AddLifecycleRule(rule);

// 分片3天后过期。
rule = new LifecycleRule(ruleId2, matchPrefix2, RuleStatus.Enabled);
LifecycleRule.AbortMultipartUpload abortMultipartUpload = new LifecycleRule.AbortMultipartUpload();
abortMultipartUpload.setExpirationDays(3);
rule.setAbortMultipartUpload(abortMultipartUpload);
request.AddLifecycleRule(rule);

// 指定日期之前的分片过期。
rule = new LifecycleRule(ruleId3, matchPrefix3, RuleStatus.Enabled);
abortMultipartUpload = new LifecycleRule.AbortMultipartUpload();
abortMultipartUpload.setCreatedBeforeDate(DateUtil.parseISO8601Date("2022-10-12T00:00:00.000Z"));
rule.setAbortMultipartUpload(abortMultipartUpload);
request.AddLifecycleRule(rule);

ossClient.setBucketLifecycle(request);

// 关闭OSSClient。
ossClient.shutdown();
```

查看生命周期规则

以下代码用于查看生命周期规则：

```


// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

List<LifecycleRule> rules = ossClient.getBucketLifecycle(bucketName);
for (LifecycleRule rule : rules) {
System.out.println(rule.getId());
System.out.println(rule.getPrefix());
System.out.println(rule.getExpirationDays());
System.out.println(rule.getCreatedBeforeDate());
if(rule.hasAbortMultipartUpload()) {
System.out.println(rule.getAbortMultipartUpload().getExpirationDays());
System.out.println(rule.getAbortMultipartUpload().getCreatedBeforeDate());
}
}

// 关闭OSSClient。
```

```
ossClient.shutdown();
```

清空生命周期规则

以下代码用于清空生命周期规则：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

ossClient.deleteBucketLifecycle(bucketName);

// 关闭OSSClient。
ossClient.shutdown();
```

跨域资源共享

跨域资源共享（Cross-origin resource sharing，简称CORS）允许Web端的应用程序访问不属于本域的资源。OSS提供跨域资源共享接口，方便您控制跨域访问的权限。

更多关于跨域资源共享的介绍，请参见开发指南中的设置跨域访问和API参考中的PutBucketcors。

设置跨域资源共享规则

以下代码用于设置指定存储空间的跨域资源共享规则：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);
```

```
SetBucketCORSRequest request = new SetBucketCORSRequest(bucketName);

// 跨域资源共享规则的容器，每个存储空间最多允许10条规则。
ArrayList<CORSRule> putCorsRules = new ArrayList<CORSRule>();

CORSRule corRule = new CORSRule();

ArrayList<String> allowedOrigin = new ArrayList<String>();
// 指定允许跨域请求的来源。
allowedOrigin.add( "http://www.b.com");

ArrayList<String> allowedMethod = new ArrayList<String>();
// 指定允许的跨域请求方法(GET/PUT/DELETE/POST/HEAD)。
allowedMethod.add("GET");

ArrayList<String> allowedHeader = new ArrayList<String>();
// 是否允许预取指令（OPTIONS）中Access-Control-Request-Headers头中指定的Header。
allowedHeader.add("x-oss-test");

ArrayList<String> exposedHeader = new ArrayList<String>();
// 指定允许用户从应用程序中访问的响应头。
exposedHeader.add("x-oss-test1");
// AllowedOrigins和AllowedMethods最多支持一个星号（*）通配符。星号（*）表示允许所有的域来源或者操作。
corRule.setAllowedMethods(allowedMethod);
corRule.setAllowedOrigins(allowedOrigin);
// AllowedHeaders和ExposeHeaders不支持通配符。
corRule.setAllowedHeaders(allowedHeader);
corRule.setExposeHeaders(exposedHeader);
// 指定浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间，单位为秒。
corRule.setMaxAgeSeconds(10);

// 最多允许10条规则。
putCorsRules.add(corRule);
// 已存在的规则将被覆盖。
request.setCorsRules(putCorsRules);
ossClient.setBucketCORS(request);

// 关闭OSSClient。
ossClient.shutdown();
```

获取跨域资源共享规则

以下代码用于获取跨域资源共享规则：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);
```

```
ArrayList<CORSRule> corsRules;
// 获取跨域资源共享规则列表。
corsRules = (ArrayList<CORSRule>) ossClient.getBucketCORSRules(bucketName);
for (CORSRule rule : corsRules) {
for (String allowedOrigin1 : rule.getAllowedOrigins()) {
// 获取允许的跨域请求源。
System.out.println(allowedOrigin1);
}

for (String allowedMethod1 : rule.getAllowedMethods()) {
// 获取允许的跨域请求方法。
System.out.println(allowedMethod1);
}

if (rule.getAllowedHeaders().size() > 0){
for (String allowedHeader1 : rule.getAllowedHeaders()) {
// 获取允许的头部列表。
System.out.println(allowedHeader1);
}
}

if (rule.getExposeHeaders().size() > 0) {
for (String exposeHeader : rule.getExposeHeaders()) {
// 获取允许的头部。
System.out.println(exposeHeader);
}
}

if ( null != rule.getMaxAgeSeconds()) {
System.out.println(rule.getMaxAgeSeconds());
}
}

// 关闭OSSClient。
ossClient.shutdown();
```

删除跨域资源共享规则

以下代码用于删除指定存储空间的所有跨域资源共享规则：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 删除存储空间的跨域资源共享规则。
ossClient.deleteBucketCORSRules(bucketName);
```

```
// 关闭OSSClient。
ossClient.shutdown();
```

设置访问日志

您可以开启存储空间的访问日志记录功能，开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。日志文件的格式为：

<TargetPrefix><SourceBucket>-YYYY-mm-DD-HH-MM-SS-UniqueString

更多关于访问日志的介绍，请参见开发指南中的**设置访问日志记录**。

开启访问日志记录

以下代码用于开启存储空间的访问日志记录：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

SetBucketLoggingRequest request = new SetBucketLoggingRequest("<yourSourceBucketName>");
// 设置存放日志文件的存储空间。
request.setTargetBucket("<yourTargetBucketName>");
// 设置日志文件存放的目录。
request.setTargetPrefix("<yourTargetPrefix>");
ossClient.setBucketLogging(request);

// 关闭OSSClient。
ossClient.shutdown();
```

查看访问日志设置

以下代码用于查看存储空间的访问日志设置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
```

```
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

BucketLoggingResult result = ossClient.getBucketLogging("<yourSourceBucketName>");
System.out.println(result.getTargetBucket());
System.out.println(result.getTargetPrefix());

// 关闭OSSClient。
ossClient.shutdown();
```

关闭访问日志记录

以下代码用于关闭存储空间的访问日志记录：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

SetBucketLoggingRequest request = new SetBucketLoggingRequest("<yourSourceBucketName>");
request.setTargetBucket(null);
request.setTargetPrefix(null);
ossClient.setBucketLogging(request);

// 关闭OSSClient。
ossClient.shutdown();
```

静态网站托管

您可以将存储空间配置成静态网站托管模式。配置生效后，访问网站相当于访问存储空间，并且能够自动跳转至指定的索引页面和错误页面。

更多关于静态网站托管的介绍，请参见开发指南中的配置静态网站托管。

设置静态网站托管

以下代码用于设置静态网站托管：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

SetBucketWebsiteRequest request = new SetBucketWebsiteRequest("<yourBucketName>");
request.setIndexDocument("index.html");
request.setErrorDocument("error.html");
ossClient.setBucketWebsite(request);

// 关闭OSSClient。
ossClient.shutdown();
```

查看静态网站托管配置

以下代码用于查看静态网站托管配置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

BucketWebsiteResult result = ossClient.getBucketWebsite("<yourBucketName>");
System.out.println(result.getIndexDocument());
System.out.println(result.getErrorDocument());

// 关闭OSSClient。
ossClient.shutdown();
```

删除静态网站托管配置

以下代码用于删除静态网站托管配置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号
```

```
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

ossClient.deleteBucketWebsite("<yourBucketName>");

// 关闭OSSClient。
ossClient.shutdown();
```

防盗链

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。
- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

更多关于防盗链的介绍，请参见开发指南中的**设置防盗链**。

设置防盗链

以下代码用于设置防盗链：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

List<String> refererList = new ArrayList<String>();
// 添加Referer白名单。Referer参数支持通配符“*”和“？”。
refererList.add("http://www.aliyun.com");
refererList.add("http://www.*.com");
refererList.add("http://www.?.aliyuncs.com");
// 设置存储空间Referer列表。设为true表示Referer字段允许为空。
BucketReferer br = new BucketReferer(true, refererList);
ossClient.setBucketReferer(bucketName, br);

// 关闭OSSClient。
```

```
ossClient.shutdown();
```

获取防盗链信息

以下代码用于获取防盗链信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 获取存储空间Referer白名单列表。
BucketReferer br = ossClient.getBucketReferer(bucketName);
List<String> refererList = br.getRefererList();
for (String referer : refererList) {
System.out.println(referer);
}

// 关闭OSSClient。
ossClient.shutdown();
```

清空防盗链

以下代码用于清空防盗链：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 防盗链不能直接清空，需要新建一个允许空Referer的规则来覆盖之前的规则。
BucketReferer br = new BucketReferer();
ossClient.setBucketReferer(bucketName, br);

// 关闭OSSClient。
ossClient.shutdown();
```

跨区域复制

跨区域复制是在不同OSS地域之间自动、异步复制文件，将源存储空间中文件的改动（新建、覆盖、删除操作）同步到目标存储空间中。该功能用于满足异地容灾和数据复制的需求。

更多跨区域复制的内容请参见开发指南中的管理跨区域复制。

开启跨区域复制

以下代码用于开启跨区域复制：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

AddBucketReplicationRequest request = new AddBucketReplicationRequest(bucketName);
request.setReplicationRuleID("<yourRuleId>");
request.setTargetBucketName("<yourTargetBucketName>");
// 目标Endpoint以北京为例。
request.setTargetBucketLocation("oss-cn-beijing");
// 设置禁止同步历史数据。默认会同步历史数据。
request.setEnableHistoricalObjectReplication(false);
ossClient.addBucketReplication(request);

// 关闭OSSClient。
ossClient.shutdown();
```

查看跨区域复制

以下代码用于查看存储空间已开启的跨区域复制：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
```

```
// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

List<ReplicationRule> rules = ossClient.getBucketReplication(bucketName);
for (ReplicationRule rule : rules) {
System.out.println(rule.getReplicationRuleID());
System.out.println(rule.getTargetBucketLocation());
System.out.println(rule.getTargetBucketName());
}

// 关闭OSSClient。
ossClient.shutdown();
```

查看跨区域复制进度

跨区域复制进度分为历史数据同步进度和实时数据同步进度。

- 历史数据同步进度用百分比表示，仅对开启了历史数据同步的存储空间有效。
- 实时数据同步进度用新写入数据的时间点表示，代表这个时间点之前的数据已同步完成。

以下代码用于查看跨区域复制进度：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

BucketReplicationProgress process = ossClient.getBucketReplicationProgress(bucketName, "<yourRuleId>");
System.out.println(process.getReplicationRuleID());
// 是否开启了历史数据同步。
System.out.println(process.isEnableHistoricalObjectReplication());
// 历史数据同步进度。
System.out.println(process.getHistoricalObjectProgress());
// 实时数据同步进度。
System.out.println(process.getNewObjectProgress());

// 关闭OSSClient。
ossClient.shutdown();
```

查看可同步的目标地域

以下代码用于获取存储空间所能同步到的地域列表：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
```

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

List<String> locations = ossClient.getBucketReplicationLocation(bucketName);
for (String loc : locations) {
System.out.println(loc);
}

// 关闭OSSClient。
ossClient.shutdown();
```

关闭跨区域复制

以下代码用于关闭已开启的跨区域复制：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 关闭跨区域复制。关闭后目标存储空间内的文件依然存在，只是不再同步源存储空间内文件的所有改动。
ossClient.deleteBucketReplication(bucketName, "<yourRuleId>");

// 关闭OSSClient。
ossClient.shutdown();
```

图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

图片处理的详细信息请参见OSS图片处理指南。

图片处理功能

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放
- 图片裁剪
- 图片旋转
- 图片效果
- 图片水印，包括添加图片、文字、图文混合水印
- 自定义图片处理样式
- 级联处理，调用多个图片处理功能

图片处理使用

图片处理使用标准的HTTP GET请求。您可以在URL的QueryString中设置处理参数。

如果图片文件的访问权限为私有读写，必须通过授权才能进行访问。

匿名访问

您可以通过如下格式的三级域名匿名访问处理后的图片：

```
http://<yourBucketName>.<yourEndpoint>/<yourObjectName>?x-oss-
process=image/<yourAction>,<yourParamValue>
```

参数	描述
bucket	存储空间名称
endpoint	存储空间所在地域的访问域名
object	图片文件名称
image	图片处理的保留标志符
action	对图片做的操作，如缩放、裁剪、旋转等
param	对图片做的操作所对应的参数

基础操作

例如，将图缩略成宽度为100，高度按比例处理：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

自定义样式

使用如下格式的三级域名匿名访问图片处理：

```
http://<yourBucketName>.<yourEndpoint>/<yourObjectName>?x-oss-process=style/<yourStyleName>
```

- style：自定义样式的保留标志符。
- yourStyleName：自定义样式名称，即通过控制台自定义样式的规则名称。

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=style/oss-pic-style-w-100
```

级联处理

通过级联处理，可以对一张图片顺序进行多个操作，格式如下：

```
http://<yourBucketName>.<yourEndpoint>/<yourObjectName>?x-oss-
process=image/<yourAction1>,<yourParamValue1>/<yourAction2>,<yourParamValue2>/...
```

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100/rotate,90
```

支持HTTPS访问

图片服务支持HTTPS访问，例如：

```
https://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

授权访问

授权访问支持自定义样式、HTTPS和级联处理。

以下代码用于生成带签名的图片处理URL：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
```

```
// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 设置图片处理样式。
String style = "image/resize,m_fixed,w_100,h_100/rotate,90";
// 指定过期时间为10分钟。
Date expiration = new Date(new Date().getTime() + 1000 * 60 * 10 );

GeneratePresignedUrlRequest req = new GeneratePresignedUrlRequest(bucketName, objectName,
HttpMethod.GET);
req.setExpiration(expiration);
req.setProcess(style);

URL signedUrl = ossClient.generatePresignedUrl(req);
System.out.println(signedUrl);

// 关闭OSSClient。
ossClient.shutdown();
```

SDK访问

对于任意权限的图片文件，都可以直接使用SDK访问和处理。

SDK处理图片文件支持自定义样式、HTTPS和级联处理。图片处理的完整代码请参见：GitHub。

基础操作

以下代码展示了图片处理的基础操作：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 缩放
String style = "image/resize,m_fixed,w_100,h_100";
GetObjectRequest request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);

ossClient.getObject(request, new File("example-resize.jpg"));

// 裁剪
style = "image/crop,w_100,h_100,x_100,y_100,r_1";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);

ossClient.getObject(request, new File("example-crop.jpg"));
```

```
// 旋转
style = "image/rotate,90";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);

ossClient.getObject(request, new File("example-rotate.jpg"));

// 锐化
style = "image/sharpen,100";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);

ossClient.getObject(request, new File("example-sharpen.jpg"));

// 水印
style = "image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);

ossClient.getObject(request, new File("example-watermark.jpg"));

// 格式转换
style = "image/format,png";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);

ossClient.getObject(request, new File("example-format.png"));

// 获取图片信息
style = "image/info";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);

ossClient.getObject(request, new File("example-info.txt"));

// 关闭OSSClient。
ossClient.shutdown();
```

自定义样式

以下代码用于自定义图片样式：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);
```

```
// 自定义样式。
String style = "style/<yourCustomStyleName>";
GetObjectRequest request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);

ossClient.getObject(request, new File("example-new.jpg"));

// 关闭OSSClient。
ossClient.shutdown();
```

级联处理

以下代码用于级联处理图片：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 级联处理。
String style = "image/resize,m_fixed,w_100,h_100/rotate,90";
GetObjectRequest request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);

ossClient.getObject(request, new File("example-new.jpg"));

// 关闭OSSClient。
ossClient.shutdown();
```

图片处理工具

您可以通过可视化图片处理工具ImageStyleViever直观地看到OSS图片处理结果。

异常处理

OSS Java SDK包含两类异常，一类是客户端异常ClientException，另一类是服务器端异常OSSException，它们均继承自RuntimeException。

异常处理示例

以下代码用于展示异常处理：

```
try {
// OSS操作，例如上传文件
ossClient.putObject(...);
} catch (OSSException oe) {
System.out.println("Caught an OSSException, which means your request made it to OSS, "
+ "but was rejected with an error response for some reason.");
System.out.println("Error Message: " + oe.getErrorCode());
System.out.println("Error Code: " + oe.getErrorCode());
System.out.println("Request ID: " + oe.getRequestId());
System.out.println("Host ID: " + oe.getHostId());
} catch (ClientException ce) {
System.out.println("Caught an ClientException, which means the client encountered "
+ "a serious internal problem while trying to communicate with OSS, "
+ "such as not being able to access the network.");
System.out.println("Error Message: " + ce.getMessage());
} finally {
if (ossClient != null) {
ossClient.shutdown();
}
}
```

ClientException

ClientException指客户端尝试向OSS发送请求以及数据传输时遇到的异常。例如，当发送请求时网络连接不可用，则会抛出ClientException。当上传文件时发生IO异常，也会抛出ClientException。

OSSException

OSSException指服务器端异常，它来自于对服务器错误信息的解析。OSSException包含OSS返回的错误码和错误信息，便于定位问题，并做出适当的处理。

OSSException通常包含以下错误信息：

参数	描述
Code	OSS返回的错误码。
Message	OSS返回的详细错误信息。
RequestId	用于唯一标识该请求的UUID。当您无法解决问题时，可以提供RequestId来请求OSS开发工程师的帮助。
HostId	用于标识访问的OSS集群，与请求时使用的Host一致。

OSS常见错误码

错误码	描述	HTTP状态码
AccessDenied	拒绝访问	403
BucketAlreadyExists	存储空间已经存在	409
BucketNotEmpty	存储空间非空	409
EntityTooLarge	实体过大	400
EntityTooSmall	实体过小	400
FileGroupTooLarge	文件组过大	400
FilePartNotExist	文件分片不存在	400
FilePartStale	文件分片过时	400
InvalidArgument	参数格式错误	400
InvalidAccessKeyId	AccessKeyId不存在	403
InvalidBucketName	无效的存储空间名称	400
InvalidDigest	无效的摘要	400
InvalidObjectName	无效的文件名称	400
InvalidPart	无效的分片	400
InvalidPartOrder	无效的分片顺序	400
InvalidTargetBucketForLogging	Logging操作中有无效的目标存储空间	400
InternalError	OSS内部错误	500
MalformedXML	XML格式非法	400
MethodNotAllowed	不支持的方法	405
MissingArgument	缺少参数	411
MissingContentLength	缺少内容长度	411
NoSuchBucket	存储空间不存在	404
NoSuchKey	文件不存在	404
NoSuchUpload	分片上传ID不存在	404
NotImplemented	无法处理的方法	501
PreconditionFailed	预处理错误	412
RequestTimeTooSkewed	客户端本地时间和OSS服务器时间相差超过15分钟	403
RequestTimeout	请求超时	400
SignatureDoesNotMatch	签名错误	403

InvalidEncryptionAlgorithmError	指定的熵编码加密算法错误	400

常见问题

包冲突

错误原因

使用OSS Java SDK时，报类似如下错误，说明您的工程里可能有包冲突。

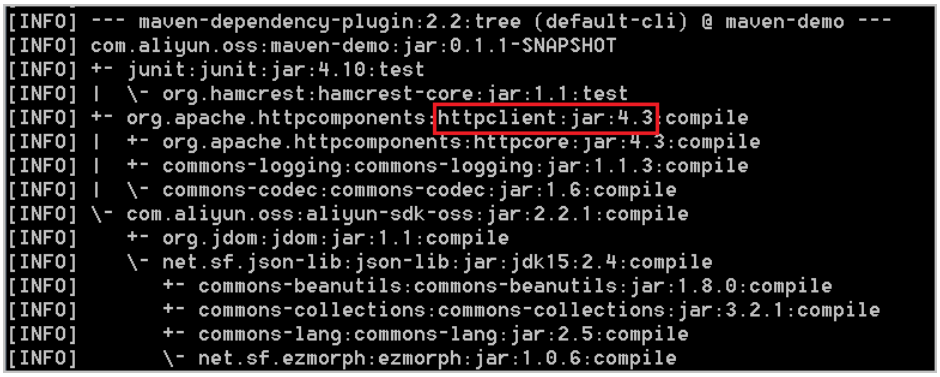
```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/http/ssl/TrustStrategy
at com.aliyun.oss.OSSClient.<init>(OSSClient.java:268)
at com.aliyun.oss.OSSClient.<init>(OSSClient.java:193)
at com.aliyun.oss.demo.HelloOSS.main(HelloOSS.java:77)
Caused by: java.lang.ClassNotFoundException: org.apache.http.ssl.TrustStrategy
at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
... 3 more
```

或

```
Exception in thread "main" java.lang.NoSuchFieldError: INSTANCE
at org.apache.http.impl.io.DefaultHttpRequestWriterFactory.<init>(DefaultHttpRequestWriterFactory.java:52)
at org.apache.http.impl.io.DefaultHttpRequestWriterFactory.<init>(DefaultHttpRequestWriterFactory.java:56)
at org.apache.http.impl.io.DefaultHttpRequestWriterFactory.<clinit>(DefaultHttpRequestWriterFactory.java:46)
at
org.apache.http.impl.conn.ManagedHttpClientConnectionFactory.<init>(ManagedHttpClientConnectionFactory.java
:82)
at
org.apache.http.impl.conn.ManagedHttpClientConnectionFactory.<init>(ManagedHttpClientConnectionFactory.java
:95)
at
org.apache.http.impl.conn.ManagedHttpClientConnectionFactory.<init>(ManagedHttpClientConnectionFactory.java
:104)
at
org.apache.http.impl.conn.ManagedHttpClientConnectionFactory.<clinit>(ManagedHttpClientConnectionFactory.ja
va:62)
at
```

```
org.apache.http.impl.conn.PoolingHttpClientConnectionManager$InternalConnectionFactory.<init>(PoolingHttpCli
entConnectionManager.java:572)
at
org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<init>(PoolingHttpClientConnectionManager.java:
174)
at
org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<init>(PoolingHttpClientConnectionManager.java:
158)
at
org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<init>(PoolingHttpClientConnectionManager.java:
149)
at
org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<init>(PoolingHttpClientConnectionManager.java:
125)
at
com.aliyun.oss.common.comm.DefaultServiceClient.createHttpClientConnectionManager(DefaultServiceClient.java:2
37)
at com.aliyun.oss.common.comm.DefaultServiceClient.<init>(DefaultServiceClient.java:78)
at com.aliyun.oss.OSSClient.<init>(OSSClient.java:268)
at com.aliyun.oss.OSSClient.<init>(OSSClient.java:193)
at OSSManagerImpl.upload(OSSManagerImpl.java:42)
at OSSManagerImpl.main(OSSManagerImpl.java:63)
```

错误原因是OSS Java SDK使用了Apache httpclient 4.4.1，而您的工程使用了与Apache httpclient 4.4.1冲突的Apache httpclient或commons-httpclient jar包。要查看工程使用的jar包及版本，请在您的工程目录下执行**mvn dependency:tree**。如下图所示，您的工程里使用了Apache httpclient 4.3：

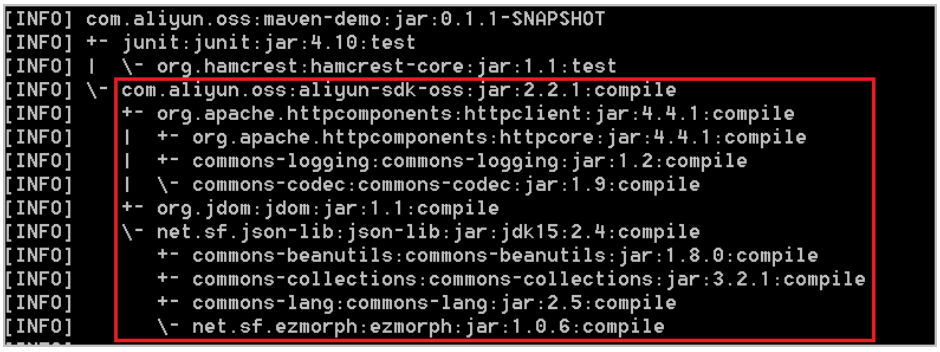


解决方法

包冲突有以下两种解决方法：

- 使用统一版本。如果您的工程使用与Apache httpclient 4.4.1冲突的版本，请您使用4.4.1版本，并在pom.xml删除其它版本的Apache httpclient依赖。如果您的工程使用了commons-httpclient，也可能存在冲突，请删除commons-httpclient。
- 解除依赖冲突。如果您的工程依赖多个第三方包，而第三方包又依赖不同版本的Apache httpclient，您的工程里会有依赖冲突，请使用exclusion解除。详细请参见Maven Guides。

OSS Java SDK依赖以下版本的包，冲突解决办法与httpclient类似。



缺少包

错误原因

使用OSS Java SDK时，报类似如下错误，说明您的工程中可能缺少编译或运行OSS Java SDK所必需的包。

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/http/auth/Credentials
at com.aliyun.oss.OSSClient.<init>(OSSClient.java:268)
at com.aliyun.oss.OSSClient.<init>(OSSClient.java:193)
at com.aliyun.oss.demo.HelloOSS.main(HelloOSS.java:76)
Caused by: java.lang.ClassNotFoundException: org.apache.http.auth.Credentials
at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
... 3 more
```

或

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/http/protocol/HttpContext
at com.aliyun.oss.OSSClient.<init>(OSSClient.java:268)
at com.aliyun.oss.OSSClient.<init>(OSSClient.java:193)
at com.aliyun.oss.demo.HelloOSS.main(HelloOSS.java:76)
Caused by: java.lang.ClassNotFoundException: org.apache.http.protocol.HttpContext
at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
... 3 more
```

或

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/jdom/input/SAXBuilder
at com.aliyun.oss.internal.ResponseParsers.getXmlRootElement(ResponseParsers.java:645)
at ... ...
at com.aliyun.oss.OSSClient.doesBucketExist(OSSClient.java:471)
at com.aliyun.oss.OSSClient.doesBucketExist(OSSClient.java:465)
at com.aliyun.oss.demo.HelloOSS.main(HelloOSS.java:82)
Caused by: java.lang.ClassNotFoundException: org.jdom.input.SAXBuilder
at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
... 11 more
```

OSS Java SDK依赖下列包：

- aliyun-sdk-oss-2.2.1.jar
- hamcrest-core-1.1.jar
- jdom-1.1.jar
- commons-codec-1.9.jar
- httpclient-4.4.1.jar
- commons-logging-1.2.jar
- httpcore-4.4.1.jar
- log4j-1.2.15.jar

其中log4j-1.2.15.jar是可选的，需要日志功能的时候加入该包，其它包都是必需的。

解决方法

在您的工程中加入OSS Java SDK依赖的包。加入方法如下：

- 如果您的工程在Eclipse中，请参见Java SDK使用手册中的安装方式二：在Eclipse项目中导入工程依赖的包。
- 如果您的工程在Ant中，请把OSS Java SDK依赖的包放入工程的lib目录中。
- 如果您直接使用.javac或.java文件，请使用**-classpath**或**-cp**命令指定OSS Java SDK依赖的包路径，或把OSS Java SDK依赖的包放入classpath路经下。

连接超时

错误原因

运行OSS Java SDK程序时，报类似如下错误，可能的原因是Endpoint错误或者网络不通。

```
com.aliyun.oss.ClientException: SocketException
at com.aliyun.oss.common.utils.ExceptionFactory.createNetworkException(ExceptionFactory.java:71)
```

```
at com.aliyun.oss.common.comm.DefaultServiceClient.sendRequestCore(DefaultServiceClient.java:116)
at com.aliyun.oss.common.comm.ServiceClient.sendRequestImpl(ServiceClient.java:121)
at com.aliyun.oss.common.comm.ServiceClient.sendRequest(ServiceClient.java:67)
at com.aliyun.oss.internal.OSSOperation.send(OSSOperation.java:92)
at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:140)
at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:111)
at com.aliyun.oss.internal.OSSBucketOperation.getBucketInfo(OSSBucketOperation.java:1152)
at com.aliyun.oss.OSSClient.getBucketInfo(OSSClient.java:1220)
at com.aliyun.oss.OSSClient.getBucketInfo(OSSClient.java:1214)
at com.aliyun.oss.demo.HelloOSS.main(HelloOSS.java:94)
Caused by: org.apache.http.conn.HttpHostConnectException: Connect to oss-test.oss-cn-hangzhou-
internal.aliyuncs.com:80 [oss-test.oss-cn-hangzhou-internal.aliyuncs.com/10.84.135.99] failed: Connection timed
out: connect
at
org.apache.http.impl.conn.DefaultHttpClientConnectionOperator.connect(DefaultHttpClientConnectionOperator.jav
a:151)
at
org.apache.http.impl.conn.PoolingHttpClientConnectionManager.connect(PoolingHttpClientConnectionManager.jav
a:353)
at org.apache.http.impl.execchain.MainClientExec.establishRoute(MainClientExec.java:380)
at org.apache.http.impl.execchain.MainClientExec.execute(MainClientExec.java:236)
at org.apache.http.impl.execchain.ProtocolExec.execute(ProtocolExec.java:184)
at org.apache.http.impl.execchain.RedirectExec.execute(RedirectExec.java:110)
at org.apache.http.impl.client.InternalHttpClient.doExecute(InternalHttpClient.java:184)
at org.apache.http.impl.client.CloseableHttpClient.execute(CloseableHttpClient.java:82)
at com.aliyun.oss.common.comm.DefaultServiceClient.sendRequestCore(DefaultServiceClient.java:113)
... 9 more
```

解决方法

您可以使用OSSProbe工具快速定位错误原因，从而解决问题。

org.apache.http.NoHttpResponseException: The target server failed to respond

错误原因

运行OSS Java SDK程序时，报类似如下错误：

使用过期的连接会导致上述错误。该错误仅在Java SDK 2.1.2前的版本出现。

解决方法

请升级OSS Java SDK到2.1.2及以后版本。

调用OSS Java SDK夯住

错误原因

调用OSS Java SDK夯住（hang）。通过**jstack -l pid**命令查看堆栈，夯在如下的位置：

```
"main" prio=6 tid=0x000000000291e000 nid=0xc40 waiting on condition [0x0000000002dae000]
java.lang.Thread.State: WAITING (parking)
at sun.misc.Unsafe.park(Native Method)
- parking to wait for <0x00000007d85697f8> (a
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
at java.util.concurrent.locks.LockSupport.park(LockSupport.java:186)
at
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:20
43)
at org.apache.http.pool.PoolEntryFuture.await(PoolEntryFuture.java:138)
at org.apache.http.pool.AbstractConnPool.getPoolEntryBlocking(AbstractConnPool.java:306)
at org.apache.http.pool.AbstractConnPool.access$000(AbstractConnPool.java:64)
at org.apache.http.pool.AbstractConnPool$2.getPoolEntry(AbstractConnPool.java:192)
at org.apache.http.pool.AbstractConnPool$2.getPoolEntry(AbstractConnPool.java:185)
at org.apache.http.pool.PoolEntryFuture.get(PoolEntryFuture.java:107)
at
org.apache.http.impl.conn.PoolingHttpClientConnectionManager.leaseConnection(PoolingHttpClientConnectionMa
nager.java:276)
at
org.apache.http.impl.conn.PoolingHttpClientConnectionManager$1.get(PoolingHttpClientConnectionManager.java:
263)
at org.apache.http.impl.execchain.MainClientExec.execute(MainClientExec.java:190)
at org.apache.http.impl.execchain.ProtocolExec.execute(ProtocolExec.java:184)
at org.apache.http.impl.execchain.RedirectExec.execute(RedirectExec.java:110)
at org.apache.http.impl.client.InternalHttpClient.doExecute(InternalHttpClient.java:184)
at org.apache.http.impl.client.CloseableHttpClient.execute(CloseableHttpClient.java:82)
at com.aliyun.oss.common.comm.DefaultServiceClient.sendRequestCore(DefaultServiceClient.java:113)
at com.aliyun.oss.common.comm.ServiceClient.sendRequestImpl(ServiceClient.java:123)
at com.aliyun.oss.common.comm.ServiceClient.sendRequest(ServiceClient.java:68)
at com.aliyun.oss.internal.OSSOperation.send(OSSOperation.java:94)
at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:146)
at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:113)
at com.aliyun.oss.internal.OSSObjectOperation.getObject(OSSObjectOperation.java:229)
at com.aliyun.oss.OSSClient.getObject(OSSClient.java:629)
at com.aliyun.oss.OSSClient.getObject(OSSClient.java:617)
at samples.HelloOSS.main(HelloOSS.java:49)
```

原因是连接池中连接泄漏，可能是使用ossObject后没有关闭。

解决方法

请检查您的程序，确保没有连接泄漏。关闭方法如下：

```
// 读取文件
OSSObject ossObject = ossClient.getObject(bucketName, objectName);
// OSS操作
// 关闭ossObject
ossObject.close();
```

问题排查的详细步骤，请参见OSS Java SDK夯住问题排查。

连接关闭

错误原因

如果您在使用ossClient.getObject时，报类似如下错误:

```
Exception in thread "main" org.apache.http.ConnectionClosedException: Premature end of Content-Length
delimited message body (expected: 11990526; received: 202880
at org.apache.http.impl.io.ContentLengthInputStream.read(ContentLengthInputStream.java:180)
at org.apache.http.impl.io.ContentLengthInputStream.read(ContentLengthInputStream.java:200)
at org.apache.http.impl.io.ContentLengthInputStream.close(ContentLengthInputStream.java:103)
at org.apache.http.impl.execchain.ResponseEntityProxy.streamClosed(ResponseEntityProxy.java:128)
at org.apache.http.conn.EofSensorInputStream.checkClose(EofSensorInputStream.java:228)
at org.apache.http.conn.EofSensorInputStream.close(EofSensorInputStream.java:174)
at java.io.FilterInputStream.close(FilterInputStream.java:181)
at java.io.FilterInputStream.close(FilterInputStream.java:181)
at com.aliyun.oss.event.ProgressInputStream.close(ProgressInputStream.java:147)
at java.io.FilterInputStream.close(FilterInputStream.java:181)
at samples.HelloOSS.main(HelloOSS.java:39)
```

原因是两次读取数据间隔时间超过1分钟。OSS会关闭超过1分钟没有发送或接收数据的连接。

解决方法

如果您每次读取部分数据进行处理，且处理数据的时间不固定，建议使用指定范围读取，避免数据读取时连接关闭。

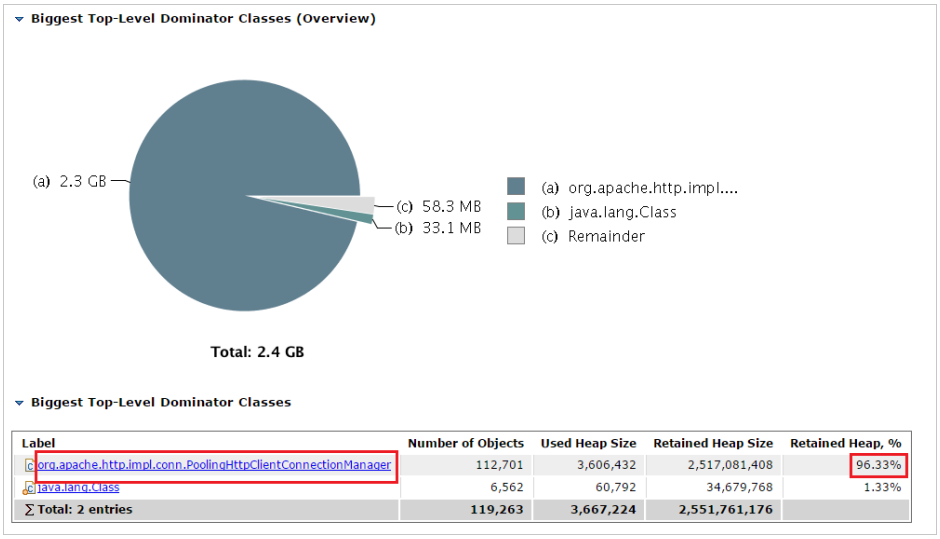
内存泄露

错误原因

调用OSS Java SDK的程序，运行一段时间（根据业务量，几小时到几天不等）后内存泄露。推荐使用Eclipse Memory Analyzer (MAT)分析内存使用情况，使用方法请参见使用MAT进行堆转储文件分析。

如果分析结果类似下图所示（PoolingHttpClientConnectionManager占96%的内存），原因是程序中可能多

次执行new OSSClient，但是没有调用ossClient.shutdown，造成内存泄漏。



解决方法

new OSSClient一定要和ossClient.shutdown成对使用。

调用ossClient.shutdown抛异常InterruptedException

错误原因

调用ossClient.shutdown抛如下异常：

```
java.lang.InterruptedException: sleep interrupted
at java.lang.Thread.sleep(Native Method)
at com.aliyun.oss.common.comm.IdleConnectionReaper.run(IdleConnectionReaper:76)
```

原因是ossClient后台线程IdleConnectionReaper会定时关闭闲置连接。IdleConnectionReaper在Sleep时，调用ossClient.shutdown，就会抛出上面的异常。

解决方法

OSS Java SDK 2.3.0已经修复该问题。

OSS Java SDK 2.3.0之前的版本，请使用如下代码，忽略该异常：

```
try {
ossClient.shutdown();
} catch(Exception e) {
}
```

其它错误

其它OSS返回错误的排查，请参见**常见错误及排查**。

Python-SDK

前言

环境准备

OSS Python SDK适用于Python 2.6、2.7、3.3、3.4、3.5版本。您可以根据**Python官网**的引导安装合适的Python版本。

不同系统下查看Python版本的方法如下：

Linux：

```
$ python -V
Python 2.7.10
```

上面的输出表明您已经成功安装了Python 2.7.10版本。

Windows：

```
C:\> python -V
Python 2.7.10
```

上面的输出表明您已经成功安装了Python 2.7.10版本。如果提示“不是内部或外部命令”，请在环境变量中编辑**Path**，增加Python和pip的安装路径。pip的安装路径一般为Python所在目录的Scripts文件夹。

说明：您可能需要重启电脑使环境变量生效。

SDK下载

- 通过GitHub下载
- 历史版本下载

更多信息请参见OSS API文档。

SDK安装

请参见OSS Python SDK安装。

示例程序

OSS Python SDK提供丰富的示例程序，方便您参考或直接使用。您可以通过以下两种方式获取示例程序：

- 通过GitHub查看和下载示例代码。
- 解压OSS Python SDK开发包，examples目录下为示例代码。

示例包括以下内容：

示例文件	示例内容
object_basic.py	快速入门，包括创建存储空间、上传、下载、列举、删除文件等
object_extra.py	上传文件和管理文件，包括设置中文、设置自定义元信息、拷贝文件、追加上传文件等
upload.py	上传文件，包括断点续传上传、分片上传等
download.py	下载文件，包括简单下载、范围下载、断点续传下载等
object_check.py	上传和下载时数据校验的用法，包括MD5和CRC
object_progress.py	带进度条的上传和下载
object_callback.py	上传文件中的上传回调
object_post.py	表单上传的相关操作
sts.py	STS的用法，包括角色扮演获取临时用户的密钥，并使用临时用户的密钥访问OSS
live_channel.py	LiveChannel的相关操作
image.py	图片处理的相关操作
bucket.py	管理存储空间，包括创建、删除、列举存储空间，以及设置静态网站托管，设置生命周期规则等

安装

安装方式

通过pip安装

执行命令如下：

```
pip install oss2
```

说明：如果您的环境尚未安装pip，请参见pip官网安装。

通过源码安装

从GitHub下载相应版本的SDK包，解压后进入目录，确认目录下有setup.py文件，然后执行命令如下：

```
python setup.py install
```

验证

验证SDK版本

1. 在命令行输入**python**并回车，进入Python环境。
2. 输入以下命令检查SDK版本：

```
>>> import oss2
>>> oss2.__version__
'2.0.0'
```

上面的输出表明您已经成功安装了Python SDK 2.0.0。

验证crcmod

OSS的CRC数据校验有两种方式：C扩展方式（通过crcmod模块计算CRC）和纯Python方式。前者的性能远优于后者。安装oss2时会自动安装crcmod。crcmod详情请参见crcmod introduction。

验证crcmod是否安装成功的方法如下：

1. 在命令行输入**python**并回车，进入Python环境。
2. 输入**import crcmod._crcfunext**并回车。
   - 如果没有出现错误提示，则表明crcmod安装成功，将使用C扩展方式。
   - 如果出现以下错误提示，则表明crcmod安装失败，将使用纯Python方式。

```
>>> import crcmod._crcfunext
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ImportError: No module named _crcfunext
```

说明：如果crcmod安装失败，建议关闭CRC数据校验，关闭的方法请参见**初始化**。

卸载SDK

如果安装失败，建议通过pip卸载然后重装。卸载命令如下：

```
pip uninstall oss2
```

初始化

Python SDK中的大部分操作都是通过oss2.Service和oss2.Bucket这两个类进行的。

- oss2.Service类用于列举存储空间。
- oss2.Bucket类用于上传、下载、删除文件以及对存储空间进行各种配置。

初始化这两个类时，需要指定Endpoint。其中oss2.Service类不支持自定义域名访问。有关Endpoint的更多信息，请参见开发人员指南中**访问域名和数据中心**和**自定义域名**。

初始化oss2.Service类

详情请参见**管理存储空间**中的列举存储空间。

初始化oss2.Bucket类

使用OSS域名初始化

下面的代码用于使用OSS域名初始化：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'http://oss-cn-hangzhou.aliyuncs.com'

bucket = oss2.Bucket(auth, endpoint, '<yourBucketName>')
```

使用自定义域名初始化

下面的代码用于使用自定义域名初始化：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')

# 自定义域名以my-domain.com为例。is_cname=True为开启CNAME。CNAME是指将自定义域名绑定到存储空间上。
cname = 'http://my-domain.com'
bucket = oss2.Bucket(auth, cname, '<yourBucketName>', is_cname=True)
```

设置连接超时时间

以下代码用于设置连接超时时间：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'http://oss-cn-hangzhou.aliyuncs.com'

# 设置连接超时时间设为30秒。
bucket = oss2.Bucket(auth, endpoint, '<yourBucketName>', connect_timeout=30)
```

关闭CRC数据校验

上传、下载文件时默认开启CRC数据校验，确保上传、下载过程的数据完整性。以下代码用于关闭CRC数据校验：

```
# -*- coding: utf-8 -*-
```

```
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'http://oss-cn-hangzhou.aliyuncs.com'

bucket = oss2.Bucket(auth, endpoint, '<yourBucketName>', enable_crc=False)
```

快速入门

本节介绍如何快速使用OSS Python SDK完成常见操作，如创建存储空间、上传文件、下载文件等。

创建存储空间

以下代码用于创建存储空间：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 设置存储空间为私有读写权限。
bucket.create_bucket(oss2.models.BUCKET_ACL_PRIVATE)
```

上传文件

以下代码用于上传文件至OSS：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
```

```
# <yourLocalFile>由本地文件路径加文件名包括后缀组成，例如/users/local/myfile.txt
bucket.put_object_from_file('<yourObjectName>', '<yourLocalFile>')
```

下载文件

以下代码用于将指定的OSS文件下载到本地文件：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# <yourLocalFile>由本地文件路径加文件名包括后缀组成，例如/users/local/myfile.txt
bucket.get_object_to_file('<yourObjectName>', '<yourLocalFile>')
```

列举文件

以下代码用于列举指定存储空间下的10个文件：

```
# -*- coding: utf-8 -*-
import oss2
from itertools import islice

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# oss2.ObjectIteratorr用于遍历文件。
for b in islice(oss2.ObjectIterator(bucket), 10):
print(b.key)
```

删除文件

以下代码用于删除指定文件：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
```

```
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

bucket.delete_object('<yourObjectName>')
```

管理存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。

创建存储空间

以下代码用于新建存储空间：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# 通过指定Endpoint和存储空间名称，您可以在指定的地域创建新的存储空间。Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 新建存储空间默认为标准存储类型，私有读写权限。
bucket.create_bucket()
```

存储空间的命名规范，请参见基本概念中的命名规范。

您可以在创建存储空间时指定存储空间的权限和存储类型。示例代码如下：

```
# 设置存储空间的存储类型为低频访问类型，访问权限为公共读。
bucket.create_bucket(oss2.BUCKET_ACL_PUBLIC_READ,
oss2.models.BucketCreateConfig(oss2.BUCKET_STORAGE_CLASS_IA))
```

列举存储空间

以下代码用于列举存储空间：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请
```

```
登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
service = oss2.Service(auth, 'http://oss-cn-hangzhou.aliyuncs.com')

print([b.name for b in oss2.BucketIterator(service)])
```

设置存储空间的访问权限

存储空间的访问权限（ACL）有以下三类：

访问权限	访问权限值
私有	oss2.BUCKET_ACL_PRIVATE
公共读	oss2.BUCKET_ACL_PUBLIC_READ
公共读写	oss2.BUCKET_ACL_PUBLIC_READ_WRITE

以下代码用于设置存储空间的访问权限：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 设置存储空间访问权限为私有读写。
bucket.put_bucket_acl(oss2.BUCKET_ACL_PRIVATE)
```

获取存储空间的访问权限

以下代码用于获取存储空间的访问权限：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

print(bucket.get_bucket_acl().acl)
```

获取存储空间的信息

以下代码用于获取存储空间的信息（Info）：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 获取存储空间相关信息
bucket_info = bucket.get_bucket_info()
print('name: ' + bucket_info.name)
print('storage class: ' + bucket_info.storage_class)
print('creation date: ' + bucket_info.creation_date)
print('intranet_endpoint: ' + bucket_info.intranet_endpoint)
print('extranet_endpoint ' + bucket_info.extranet_endpoint)
print('owner: ' + bucket_info.owner.id)
print('grant: ' + bucket_info.acl.grant)
```

删除存储空间

删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。

以下代码用于删除存储空间：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
try:
# 删除存储空间。
bucket.delete_bucket()
except oss2.exceptions.BucketNotEmpty:
print('bucket is not empty.')
except oss2.exceptions.NoSuchBucket:
print('bucket does not exist')
```

对于非空的存储空间，可以通过边列举边删除（对于分片上传则是终止上传）的方法清空存储空间，然后再删除。

上传文件

在OSS中，操作的基本数据单元是文件（Object）。OSS Python SDK提供了丰富的文件上传方式：

- 简单上传：文件最大不能超过5GB。
- 追加上传：文件最大不能超过5GB。
- 断点续传上传：支持并发、断点续传、自定义分片大小。大文件上传推荐使用断点续传。最大不能超过48.8TB。
- 分片上传：当文件较大时，可以使用分片上传，最大不能超过48.8TB。

说明：各种上传方式的适用场景请参见开发指南中的上传文件。

上传过程中，您还可以通过上传进度条功能查看上传进度。上传完成后，您还可以进行上传回调。

简单上传

通过bucket.put_object方法上传文件。上传方法支持多种类型的输入源，输入源有如下几种类型：

类型	上传方式
字符串	直接上传
Bytes	直接上传
Unicode	自动转换为UTF-8编码的Bytes进行上传
本地文件	文件对象（File Object），必须以二进制方式打开（如“rb”模式）
网络流	可迭代对象（Iterable），以Chunked Encoding的方式上传

上传字符串

以下代码用于上传字符串：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 返回值。
```

```
result = bucket.put_object('<yourObjectName>', 'content of object')
# HTTP返回码。
print('http status: {0}'.format(result.status))
# 请求ID。请求ID是请求的唯一标识，强烈建议在程序日志中添加此参数。
print('request_id: {0}'.format(result.request_id))
# ETag是put_object方法返回值特有的属性。
print('ETag: {0}'.format(result.etag))
# HTTP响应头部。
print('date: {0}'.format(result.headers['date']))
```

上传Bytes

以下代码用于上传Bytes：

```
# -*- coding: utf-8 -*-

import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

bucket.put_object('<yourObjectName>', b'content of object')
```

上传Unicode

以下代码用于上传Unicode：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

bucket.put_object('<yourObjectName>', u'content of object')
```

上传本地文件

以下代码用于上传本地文件：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
```

```
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 必须以二进制的方式打开文件，因为需要知道文件包含的字节数。
with open('<yourLocalFile>', 'rb') as fileobj:
# Seek方法用于指定从第1000个字节位置开始读写。上传时会从您指定的第1000个字节位置开始上传，直到文件结束。
fileobj.seek(1000, os.SEEK_SET)
# Tell方法用于返回当前位置。
current = fileobj.tell()
bucket.put_object('<yourObjectName>', fileobj)
```

Python SDK还提供了一个更加便捷的方法用于上传本地文件：

```
bucket.put_object_from_file('<yourObjectName>', '<yourLocalFile>')
```

上传网络流

以下代码用于上传网络流：

```
# -*- coding: utf-8 -*-
import oss2
import requests

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# requests.get返回的是一个可迭代对象（Iterable），此时Python SDK会通过Chunked Encoding方式上传。
input = requests.get('http://www.aliyun.com')
bucket.put_object('<yourObjectName>', input)
```

追加上传

以下代码用于追加上传文件：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 设置首次上传的追加位置（Position参数）为0。
result = bucket.append_object('<yourObjectName>', 0, 'content of first append')
```

```
# 如果不是首次上传，可以通过bucket.head_object方法或上次追加返回值的next_position属性，得到追加位置。
bucket.append_object('<yourObjectName>', result.next_position, 'content of second append')
```

如果文件已经存在，如下两种情况将会抛出异常：

- 不是可追加文件，则抛出ObjectNotAppendable异常。
- 是可追加文件，但设置的追加位置和文件当前长度不等，则抛出PositionNotEqualToLength异常。

断点续传上传

断点续传上传将要上传的文件分成若干个分片（Part）分别上传，所有分片都上传完成后，将所有分片合并成完整的文件，完成整个文件的上传。

您可以通过oss2.resumable_upload方法断点续传上传指定文件，该方法包含以下参数：

参数	描述	是否必需	默认值
bucket	存储空间名称	是	无
key	文件名称	是	无
filename	待上传的本地文件名称	是	无
store	指定保存断点信息的目录	否	HOME目录下建立的.py-oss-upload目录
headers	HTTP头部	否	无
multipart_threshold	文件长度大于该值时，则用分片上传	否	10MB
part_size	分片大小	否	自动计算
progress_callback	上传进度回调函数	否	无
num_threads	并发上传的线程数	否	1

以下代码用于断点续传上传：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 当文件长度大于或等于可选参数multipart_threshold（默认值为10MB）时，会使用分片上传。如未使用参数store指定目录，则会在HOME目录下建立.py-oss-upload目录来保存断点信息。
oss2.resumable_upload(bucket, '<yourObjectName>', '<yourLocalFile>')
```

Python SDK 2.1.0以上版本支持设置可选参数进行断点续传上传，代码如下：

```
# 如使用store指定了目录，则保存断点信息在指定目录中。如使用num_threads设置上传并发数，请将
oss2.defaults.connection_pool_size设成大于或等于线程数。默认线程数为1。
oss2.resumable_upload(bucket, '<yourObjectName>', '<yourLocalFile>',
store=oss2.ResumableStore(root='/tmp'),
multipart_threshold=100*1024,
part_size=100*1024,
num_threads=4)
```

断点续传详情请参见开发指南中的**断点续传**。

分片上传

分片上传（Multipart Upload）分为以下三个步骤：

- 初始化（bucket.init_multipart_upload）：获得Upload ID。
- 上传分片（bucket.upload_part）：上传分片数据。这一步可以并发进行。
- 完成上传（bucket.complete_multipart_upload）：所有分片上传完成后，合并分片，生成OSS文件。

以下代码用于分片上传文件：

```
# -*- coding: utf-8 -*-
import os
from oss2 import SizedFileAdapter, determine_part_size
from oss2.models import PartInfo
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请
登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

key = '<yourObjectName>'
filename = '<yourLocalFile>'

total_size = os.path.getsize(filename)
# determine_part_size方法用来确定分片大小。
part_size = determine_part_size(total_size, preferred_size=100 * 1024)

# 初始化分片。
upload_id = bucket.init_multipart_upload(key).upload_id
parts = []

# 逐个上传分片。
with open(filename, 'rb') as fileobj:
part_number = 1
offset = 0
while offset < total_size:
num_to_upload = min(part_size, total_size - offset)
# SizedFileAdapter(fileobj, size)方法会生成一个新的文件对象，重新计算起始追加位置。
```

```
result = bucket.upload_part(key, upload_id, part_number,
SizedFileAdapter(fileobj, num_to_upload))
parts.append(PartInfo(part_number, result.etag))

offset += num_to_upload
part_number += 1

# 完成分片上传。
bucket.complete_multipart_upload(key, upload_id, parts)

# 验证分片上传。
with open(filename, 'rb') as fileobj:
assert bucket.get_object(key).read() == fileobj.read()
```

上传进度条

进度条用于指示上传或下载的进度。下面的代码以bucket.put_object方法为例，介绍如何使用进度条。

```
# -*- coding: utf-8 -*-
from __future__ import print_function
import os, sys
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 当无法确定待上传的数据长度时，total_bytes的值为None。
def percentage(consumed_bytes, total_bytes):
if total_bytes:
rate = int(100 * (float(consumed_bytes) / float(total_bytes)))
print('\r{0}% '.format(rate), end='')

sys.stdout.flush()

# progress_callback为可选参数，用于实现进度条功能。
bucket.put_object('<yourObjectName>', 'a'*1024*1024, progress_callback=percentage)
```

进度条的完整示例代码请参见GitHub。

上传回调

以下代码用于上传回调：

```
# -*- coding: utf-8 -*-
import json
import base64
import os
import oss2
```

```
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 准备回调参数。
callback_dict = {}
callback_dict['callbackUrl'] = 'http://oss-demo.aliyuncs.com:23450'
callback_dict['callbackHost'] = 'oss-cn-hangzhou.aliyuncs.com'
callback_dict['callbackBody'] = 'filename=${object}&size=${size}&mimeType=${mimeType}'
callback_dict['callbackBodyType'] = 'application/x-www-form-urlencoded'
# 回调参数是Json格式，并且需要Base64编码。
callback_param = json.dumps(callback_dict).strip()
base64_callback_body = base64.b64encode(callback_param)
# 回调参数编码后放在Header中发送给OSS。
headers = {'x-oss-callback': base64_callback_body}

# 上传并回调。
result = bucket.put_object('<yourObjectName>', 'a'*1024*1024, headers)
```

put_object、put_object_from_file、complete_multipart_upload支持上传回调功能。上传回调的详细说明请参见API参考中的**上传回调**。上传回调的完整示例代码请参见**GitHub**。

下载文件

OSS Python SDK提供了丰富的文件下载方式：

- 流式下载
- 下载到本地文件
- 范围下载
- 断点续传下载

下载过程中，您还可以通过**下载进度条**功能查看下载进度。

流式下载

以下代码用于流式下载文件：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
```

```
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# bucket.get_object的返回值是一个类文件对象（File-Like Object），同时也是一个可迭代对象（Iterable）。
object_stream = bucket.get_object('<yourObjectName>')
print(object_stream.read())
```

以下代码用于将数据下载到本地文件中：

```
import shutil

# object_stream是类文件对象，您可以使用shutil.copyfileobj方法，将数据下载到本地文件中。
object_stream = bucket.get_object('<yourObjectName>')
with open('<yourLocalFile>', 'wb') as local_fileobj:
shutil.copyfileobj(object_stream, local_fileobj)
```

以下代码用于流式拷贝到另一个文件中：

```
# object_stream是一个可迭代对象，您可以将它流式拷贝到另一个文件中。
object_stream = bucket.get_object('<yourObjectName>')
bucket.put_object('<yourBackupObjectName>', object_stream)
```

下载到本地文件

以下代码用于将指定OSS文件下载到本地文件中：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 下载OSS文件到本地文件。如果指定的本地文件存在会覆盖，不存在则新建。
bucket.get_object_to_file('<yourObjectName>', '<yourLocalFile>')
```

范围下载

如果仅需要文件中的部分数据，您可以使用范围下载，下载指定范围内的数据，代码如下：

```
# -*- coding: utf-8 -*-
import oss2
```

```
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 获取0~99字节范围内的数据，包括0和99，共100个字节的数据。如果指定的范围无效（比如开始或结束位置的指定值为负数，或指定值大于文件大小），则下载整个文件。
object_stream = bucket.get_object('<yourObjectName>', byte_range=(0, 99))
```

断点续传下载

当下载大文件时，如果网络不稳定或者程序异常退出，会导致下载失败，甚至重试多次仍无法完成下载。为此OSS提供了断点续传下载功能。

断点续传下载的流程如下：

1. 在本地创建一个临时文件，文件名由原文件名加上一个随机的后缀组成。
2. 通过指定HTTP请求的Range头，按照范围读取OSS文件，并写入到临时文件里相应的位置。
3. 下载完成之后，把临时文件重命名为目标文件。如目标文件已存在会覆盖，不存在则新建。

您可以通过oss2.resumable_download方法断点续传下载，该方法中包含以下参数：

参数	描述	是否必需	默认值
bucket	存储空间名称。	是	无
key	OSS文件名称。	是	无
filename	本地文件。OSS文件将下载到该文件中。	是	无
store	记录本地分片下载结果的文件。下载过程中，断点信息会保存在此文件中，如果下载中断了，再次下载时会根据文件中记录的点继续下载。	否	HOME目录下建立的.py-oss-download目录。
multipart_threshold	文件长度大于该值时，则使用断点续传下载。	否	10MB
part_size	分片大小。	否	自动计算
progress_callback	下载进度回调函数。	否	无
num_threads	并发下载的线程数。	否	1

以下代码用于断点续传下载：

```
# -*- coding: utf-8 -*-
import oss2
```

```
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

oss2.resumable_download(bucket, '<yourObjectName>', '<yourLocalFile>')
```

Python SDK 2.1.0以上版本支持设置可选参数进行断点续传下载，代码如下：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 请将oss2.defaults.connection_pool_size设成大于或等于线程数，并将part_size参数设成大于或等于oss2.defaults.multiget_part_size。
oss2.resumable_download(bucket, '<yourObjectName>', '<yourLocalFile>',
store=oss2.ResumableDownloadStore(root='/tmp'),
multiget_threshold=20*1024*1024,
part_size=10*1024*1024,
num_threads=3)
```

说明：避免多个程序（线程）同时调用该方法下载同一个源文件到同一个目标文件中。因为断点信息会在本地磁盘上互相覆盖，且临时文件名可能会冲突。

下载进度条

进度条用于指示上传或下载的进度。下面的代码以bucket.get_object_to_file方法为例，介绍如何使用进度条。

```
# -*- coding: utf-8 -*-
from __future__ import print_function
import os, sys
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 当HTTP响应头部没有Content-Length时，total_bytes的值为None。
def percentage(consumed_bytes, total_bytes):
if total_bytes:
rate = int(100 * (float(consumed_bytes) / float(total_bytes)))
```

```
print('\r{0}% '.format(rate), end='')

sys.stdout.flush()

# progress_callback是可选参数，用于实现进度条功能。
bucket.get_object_to_file('<yourObjectName>', '<yourLocalFile>', progress_callback=percentage)
```

进度条的完整示例代码请参见GitHub。

管理文件

您可以通过一系列的接口管理存储空间（Bucket）下的文件（Object），包括以下操作：

- 判断文件是否存在
- 管理文件访问权限
- 管理文件元信息
- 列举文件
- 删除文件
- 拷贝文件
- 解冻归档文件
- 管理符号链接

判断文件是否存在

以下代码用于判断指定的文件是否存在：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

exist = bucket.object_exists('<yourObjectName>')
# 返回值为true表示文件存在，false表示文件不存在。
if exist:
print('object exist')
else:
print('object not eixst')
```

管理文件访问权限

文件的访问权限（ACL）有以下四种：

访问权限	描述
继承Bucket	文件遵循存储空间的访问权限。
私有	文件是私有资源。文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。
公共读	文件是公共读资源。文件拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。
公共读写	文件是公共读写资源。所有用户都有该文件的读写权限。

文件的访问权限优先级高于存储空间的访问权限。例如存储空间的访问权限是私有读写，而文件的访问权限是公共读写，则所有用户都有该文件的读写权限。如果某个文件没有设置过访问权限，则遵循存储空间的访问权限。

设置文件访问权限

您可以通过bucket.put_object_acl设置文件的访问权限。

访问权限	访问权限值
继承Bucket	oss2.OBJECT_ACL_DEFAULT
私有	oss2.OBJECT_ACL_PRIVATE
公共读	oss2.OBJECT_ACL_PUBLIC_READ
公共读写	oos2.OBJECT_ACL_PUBLIC_READ_WRITE

以下代码用于设置指定文件的访问权限：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 设置文件的访问权限。
bucket.put_object_acl('<yourObjectName>', oss2.OBJECT_ACL_PUBLIC_READ)
```

获取文件访问权限

以下代码用于获取文件访问权限：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

print(bucket.get_object_acl('<yourObjectName>').acl)
```

管理文件元信息

文件元信息（Object Meta）详情请参见文件元信息。

设置HTTP header

OSS允许自定义HTTP header。

以下代码用于设置HTTP header：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

bucket.put_object('<yourObjectName>', '{"age": 1}', headers={'Content-Type': 'application/json; charset=utf-8'})
```

HTTP header详情请参见RFC2616。

设置自定义元信息

您可以自定义文件的元信息来对文件进行描述。

以下代码用于设置文件的自定义元信息：

```
# -*- coding: utf-8 -*-

import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
```

```
bucket.put_object('<yourObjectName>', 'a novel', headers={'x-oss-meta-author': 'O. Henry'})
```

修改元信息

以下代码用于修改文件元信息：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

bucket.update_object_meta('<yourObjectName>', {'x-oss-meta-author': 'O. Henry'})
# 每次调用bucket.update_object_meta都会清空之前设置的值。
bucket.update_object_meta('<yourObjectName>', {'x-oss-meta-price': '100 dollar'})
```

以下代码用于更改Content-Type等元信息：

```
bucket.update_object_meta('<yourObjectName>', {'x-oss-meta-author': 'O. Henry'})
# 每次调用bucket.update_object_meta都会清空之前设置的值。
bucket.update_object_meta('<yourObjectName>', {'Content-Type': 'text/plain'})
```

列举文件

OSS文件按照字母顺序排列。Python SDK提供了一系列列举文件的方法。

简单列举

以下代码用于列举指定存储空间下的10个文件：

```
# -*- coding: utf-8 -*-
import oss2
from itertools import islice

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

for b in islice(oss2.ObjectIterator(bucket), 10):
print(b.key)
```

列举指定前缀的文件

以下代码用于列举包含指定前缀（prefix）的文件：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 列举包含指定前缀的文件。默认列举100个文件。
for obj in oss2.ObjectIterator(bucket, prefix = 'img-'):
print(obj.key)
```

列举存储空间下所有文件

列举文件夹操作比较低效，不建议使用。

以下代码用于列举指定存储空间下的所有文件：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 设置Delimiter参数为正斜线（/）。
for obj in oss2.ObjectIterator(bucket, delimiter = '/'):
# 通过is_prefix方法判断obj是否为文件夹。
if obj.is_prefix(): # 文件夹
print('directory: ' + obj.key)
else: # 文件
print('file: ' + obj.key)
```

删除文件

删除单个文件

以下代码用于删除单个文件：

```
# -*- coding: utf-8 -*-
import oss2
```

```
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

bucket.delete_object('<yourObjectName>')
```

删除多个文件

以下代码用于批量删除文件：

```
# 批量删除3个文件。每次最多删除1000个文件。
result = bucket.batch_delete_objects(['<yourObjectName-a>', '<yourObjectName-b>', '<yourObjectName-c>'])
# 打印成功删除的文件名。
print('\n'.join(result.deleted_keys))
```

拷贝文件

您可以通过拷贝方法将文件从一个存储空间（源存储空间）复制到同一地域的另一个存储空间（目标存储空间）中。

拷贝时限制条件如下：

- 用户有源文件的读写权限。
- 不支持跨地域拷贝。例如，不支持将杭州存储空间里的文件拷贝到青岛。

简单拷贝

对于小于1GB的文件，您可以使用简单拷贝：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourDestinationBucketName>')

bucket.copy_object('<yourSourceBucketName>', '<yourSourceObjectName>', '<yourDestinationObjectName>')
```

拷贝大文件

对于大于1GB的文件，需要使用分片拷贝。分片拷贝分为三步：

1. 通过bucket.init_multipart_upload初始化分片拷贝任务。
2. 通过bucket.upload_part_copy进行分片拷贝。除最后一个分片外，其它分片都要大于100KB。
3. 通过bucket.complete_multipart_copy提交分片拷贝任务。

以下代码用于分片拷贝：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import PartInfo
from oss2 import determine_part_size

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

src_object = '<yourSourceObjectName>'
dst_object = '<yourDestinationObjectName>'

bucket.put_object(src_object, 'a' * (1024 * 1024 + 100))

total_size = bucket.head_object(src_object).content_length
part_size = determine_part_size(total_size, preferred_size=100 * 1024)

# 初始化分片。
upload_id = bucket.init_multipart_upload(dst_object).upload_id
parts = []

# 逐个分片拷贝。
part_number = 1
offset = 0
while offset < total_size:
num_to_upload = min(part_size, total_size - offset)
byte_range = (offset, offset + num_to_upload - 1)

result = bucket.upload_part_copy(bucket.bucket_name, src_object, byte_range,dst_object, upload_id, part_number)
parts.append(PartInfo(part_number, result.etag))

offset += num_to_upload
part_number += 1

# 完成分片拷贝。
bucket.complete_multipart_upload(dst_object, upload_id, parts)
```

解冻归档文件

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。非归档类型的文件，不要调用restore_object方法。

归档文件的状态变换过程如下：

1. 归档类型的文件初始时处于冷冻状态。
2. 提交解冻操作后，服务端执行解冻，文件处于解冻中的状态。
3. 完成解冻后，可以读取文件。解冻状态默认持续1天，最多延长7天，之后文件又回到冷冻状态。

以下代码用于解冻归档文件：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
objectName = '<yourObjectName>'

bucket.restore_object(objectName)
```

归档存储类型的详细说明请参见存储类型介绍。

管理符号链接

创建符号链接

符号链接是一种特殊的文件，它指向某个具体的文件，类似于Windows上使用的快捷方式。

以下代码用于创建符号链接：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

objectName = '<yourObjectName>'
symlink = "<yourSymlink>";

bucket.put_symlink(objectName, symlink)
```

符号链接的详细信息请参见PutSymlink。

获取符号链接指向的文件内容

获取符号链接要求您对该符号链接有读权限。以下代码用于获取符号链接指向的文件内容：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

symlink = "<yourSymlink>";

bucket.get_symlink(symlink)
```

符号链接的详细信息请参见GetSymlink。

授权访问

使用签名URL下载文件

您可以将生成的签名URL提供给访客进行临时访问。生成签名URL时，您可以指定URL的过期时间，来限制访客的访问时长。

以下代码用于使用签名URL下载文件：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 设置此签名URL在60秒内有效。
print(bucket.sign_url('GET', '<yourObjectName>', 60))
```

使用STS进行临时授权

您可以通过STS（Security Token Service）进行临时授权访问。更多有关STS的内容请参见访问控制API参考（STS）中的简介。关于账号及授权的详细信息请参见最佳实践中的STS临时授权访问。

首先您需要安装官方的Python STS客户端：

```
pip install aliyun-python-sdk-sts
```

请使用2.0.6及以上版本。以下代码用于使用STS临时授权上传文件：

```
# -*- coding: utf-8 -*-

from aliyunsdkcore import client
from aliyunsdksts.request.v20150401 import AssumeRoleRequest
import json
import oss2

# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'oss-cn-hangzhou.aliyuncs.com'
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
bucket_name = '<yourBucketName>'
# role_arn是角色的资源名称。
role_arn = '<yourRoleArn>'

clt = client.AcsClient(access_key_id, access_key_secret, 'cn-hangzhou')
req = AssumeRoleRequest.AssumeRoleRequest()

# 设置返回值格式为JSON。
req.set_accept_format('json')
req.set_RoleArn(role_arn)
req.set_RoleSessionName('session-name')
body = clt.do_action(req)

# 使用RAM账号的AccessKeyId和AccessKeySecret向STS申请临时token。
token = json.loads(body)

# 使用临时token中的认证信息初始化StsAuth实例。
auth = oss2.StsAuth(token['Credentials']['AccessKeyId'],
token['Credentials']['AccessKeySecret'],
token['Credentials']['SecurityToken'])

# 使用StsAuth实例初始化存储空间。
bucket = oss2.Bucket(auth, endpoint, bucket_name)

# 上传一个字符串。
bucket.put_object('object-name.txt', b'hello world')
```

STS应用完整的示例代码请参见GitHub。

客户端加密

客户端加密是指将数据发送到OSS之前在用户本地进行加密。数据加密密钥的保存方式有如下两种：

- 用户自主管理（RSA）
- KMS托管

使用以上两种加密方式能够有效的避免密钥的泄漏，保护客户端数据安全，即使数据泄漏其他人也无法解密得到原始数据。

客户端加密详细信息请参见开发指南中的客户端加密SDK介绍，完整的示例代码请参见GitHub。

加密元信息

参数	描述	是否必需
x-oss-meta-oss-crypto-key	加密后的密钥。 经过RSA加密后再经过base64编码的字符串。	是
x-oss-meta-oss-crypto-start	随机产生的加密数据的初始值。经过RSA加密后再经过base64编码的字符串。	是
x-oss-meta-oss-cek-alg	数据的加密算法，取值为AES/GCM/NoPadding。	是
x-oss-meta-oss-wrap-alg	数据密钥的加密算法。取值为rsa和kms。	是
x-oss-meta-oss-matdesc	内容加密密钥（CEK）描述，JSON格式。暂未生效。	否
x-oss-meta-unencrypted-content-length	加密前的数据长度。如未指定content-length则不生成该参数。	否
x-oss-meta-unencrypted-content-md5	加密前的数据的MD5。如未指定MD5则不生成该参数。	否

使用用户自主管理方式上传和下载文件

您可以使用用户自主管理方式上传和下载文件。代码如下：

```
# -*- coding: utf-8 -*-
import os
import oss2
from oss2.crypto import LocalRsaProvider

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
```

```
# 创建存储空间，使用用户自主管理（RSA）方式加密，此方式只支持文件整体上传下载操作。
bucket = oss2.CryptoBucket(auth,'<yourEndpoint>', '<yourBucketName>', crypto_provider=LocalRsaProvider())

key = 'motto.txt'
content = b'a' * 1024 * 1024
filename = 'download.txt'


# 上传文件
bucket.put_object(key, content, headers={'content-length': str(1024 * 1024)})

# 下载文件
result = bucket.get_object(key)

# 验证
content_got = b''
for chunk in result:
content_got += chunk
assert content_got == content

# 下载OSS文件到本地文件
result = bucket.get_object_to_file(key, filename)

# 验证
with open(filename, 'rb') as fileobj:
assert fileobj.read() == content

os.remove(filename)
```

使用KMS托管方式上传和下载文件

前提条件

1. 已开通阿里云密钥管理服务。
2. 已生成您所在地域的密钥ID。
3. 已创建自定义授权策略（操作路径：**控制台** > **访问控制** > **策略管理** > **自定义授权策略** > **新建授权策略**），并为对应用户关联此策略。

参考以下格式创建自定义策略：

```
{
"Version": "1",
"Statement": [
{
"Effect": "Allow",
"Action": [
"kms:CreateKey",
"kms:GenerateDataKey",
"kms:ListKeys",
"kms:Encrypt",
"kms:Decrypt"
```

```
],
"Resource": [
"acs:kms:<yourRegion>:<yourloginUserId>:key/*"
]
}
```

示例代码

```
# -*- coding: utf-8 -*-
import os
import oss2
from oss2.crypto import AliKMSProvider

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')

# 创建存储空间，使用KMS托管方式加密，此方式只支持文件整体上传下载操作。
bucket = oss2.CryptoBucket(auth,'<yourEndpoint>',
'liusiman123456',crypto_provider=AliKMSProvider('<yourAccessKeyId>', '<yourAccessKeySecret>', '<yourRegion>',
'<yourCMK>', '1234'))

key = 'motto.txt'
content = b'a' * 1024 * 1024
filename = 'download.txt'

# 上传文件
bucket.put_object(key, content, headers={'content-length': str(1024 * 1024)})

# 下载文件
result = bucket.get_object(key)

# 验证
content_got = b''
for chunk in result:
content_got += chunk
assert content_got == content

# 下载OSS文件到本地文件
result = bucket.get_object_to_file(key, filename)

# 验证
with open(filename, 'rb') as fileobj:
assert fileobj.read() == content

os.remove(filename)
```

静态网站托管

您可以将存储空间配置成静态网站托管模式。配置生效后，访问网站相当于访问存储空间，并且能够自动跳转至指定的索引页面和错误页面。

更多关于静态网站托管的介绍，请参见开发指南中的配置静态网站托管。

设置静态网站托管

以下代码用于设置静态网站托管：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import BucketWebsite

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 开启静态网站托管模式，并把索引页面设置为index.html，404错误页面设置为error.html。
bucket.put_bucket_website(BucketWebsite('index.html', 'error.html'))
```

查看静态网站托管配置

以下代码用于查看静态网站托管配置：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

try:
# 当静态网站托管模式没有开启时，get_bucket_website会抛出NoSuchWebsite异常。
website = bucket.get_bucket_website()
print('Index file is {0}, error file is {1}'.format(website.index_file, website.error_file))
except oss2.exceptions.NoSuchWebsite as e:
print('Website is not configured, request_id={0}'.format(e.request_id))
```

删除静态网站托管配置

以下代码用于删除静态网站托管配置：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

bucket.delete_bucket_website()
```

生命周期管理

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。每条规则包含：

- 规则ID。用于标识一条规则，同一存储空间内规则ID不能重复。
- 策略。有以下两种设置方式。同一存储空间内仅支持一种设置方式。
  - 按前缀匹配。此种方式允许创建多条规则，前缀不能重复。
  - 配置到整个存储空间。此种方式只能创建一条规则。
- 过期时间。有两种指定方式：
  - 指定距文件最后修改时间N天后过期。
  - 指定日期创建前的文件过期，之后的不过期。
- 是否生效。

通过upload_part方法上传的分片也支持设置生命周期规则。文件最后修改时间以初始化分片上传事件的时间为准。

更多关于生命周期的内容请参见**管理对象生命周期**。

设置生命周期规则

以下代码用于设置生命周期规则：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import LifecycleExpiration, LifecycleRule, BucketLifecycle,AbortMultipartUpload
import datetime

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
```

```
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 距最后修改时间3天后过期。
rule1 = LifecycleRule('rule1', 'tests/',
status=LifecycleRule.ENABLED,
expiration=LifecycleExpiration(days=3))

# 指定日期之前创建的文件过期。
rule2 = LifecycleRule('rule2', 'logging-',
status=LifecycleRule.DISABLED,
expiration=LifecycleExpiration(created_before_date=datetime.date(2018, 12, 12)))

# 分片3天后过期。
rule3 = LifecycleRule('rule3', 'tests1/',
status=LifecycleRule.ENABLED,
abort_multipart_upload=AbortMultipartUpload(days=3))

# 指定日期之前的分片过期。
rule4 = LifecycleRule('rule4', 'logging1-',
status=LifecycleRule.DISABLED,
abort_multipart_upload = AbortMultipartUpload(created_before_date=datetime.date(2018, 12, 12)))

lifecycle = BucketLifecycle([rule1, rule2,rule3,rule4])

bucket.put_bucket_lifecycle(lifecycle)
```

查看生命周期规则

以下代码用于查看生命周期规则：

```


# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

lifecycle = bucket.get_bucket_lifecycle()

for rule in lifecycle.rules:
if rule.abort_multipart_upload is None:
print('id={0}, prefix={1}, status={2}, days={3}, created_before_date={4}'
.format(rule.id, rule.prefix, rule.status,
rule.expiration.days,
rule.expiration.created_before_date))
else:
print('id={0}, prefix={1}, status={2}, days={3}, created_before_date={4}'
.format(rule.id, rule.prefix, rule.status,
rule.abort_multipart_upload.days,
rule.abort_multipart_upload.created_before_date))
```

清空生命周期规则

以下代码用于清空生命周期规则：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

bucket.delete_bucket_lifecycle()

# 再次查看生命周期规则会抛出异常。
try:
lifecycle = bucket.get_bucket_lifecycle()
except oss2.exceptions.NoSuchLifecycle:
print('lifecycle is not configured')
```

跨域资源共享

跨域资源共享（Cross-origin resource sharing，简称CORS）允许Web端的应用程序访问不属于本域的资源。OSS提供跨域资源共享接口，方便您控制跨域访问的权限。

更多关于跨域资源共享的介绍，请参见开发指南中的设置跨域访问和API参考中的PutBucketcors。

设置跨域资源共享规则

以下代码用于设置指定存储空间的跨域资源共享规则：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import BucketCors, CorsRule

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

rule = CorsRule(allowed_origins=['*'],
allowed_methods=['GET', 'HEAD'],
```

```
allowed_headers=['*'],
max_age_seconds=1000)

# 已存在的规则将被覆盖。
bucket.put_bucket_cors(BucketCors([rule]))
```

获取跨域资源共享规则

以下代码用于获取跨域资源共享规则：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

try:
cors = bucket.get_bucket_cors()
except oss2.exceptions.NoSuchCors:
print('cors is not set')
else:
for rule in cors.rules:
print('AllowedOrigins={0}'.format(rule.allowed_origins))
print('AllowedMethods={0}'.format(rule.allowed_methods))
print('AllowedHeaders={0}'.format(rule.allowed_headers))
print('ExposeHeaders={0}'.format(rule.expose_headers))
print('MaxAgeSeconds={0}'.format(rule.max_age_seconds))
```

删除跨域资源共享规则

以下代码用于删除指定存储空间的所有跨域资源共享规则：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

bucket.delete_bucket_cors()
```

设置访问日志

您可以开启存储空间的访问日志记录功能，开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。日志文件的格式为：

```
<TargetPrefix><SourceBucket>-YYYY-mm-DD-HH-MM-SS-UniqueString
```

更多关于访问日志的介绍，请参见开发指南中的设置访问日志。

开启访问日志记录

以下代码用于开启存储空间的访问日志记录：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import BucketLogging

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 开启日志记录。把日志保存在当前存储空间，设置日志文件存放的目录为 `logging/`。
bucket.put_bucket_logging(BucketLogging(bucket.bucket_name, 'logging/'))
```

查看访问日志设置

以下代码用于查看存储空间的访问日志设置：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

logging = bucket.get_bucket_logging()
print('TargetBucket={0}, TargetPrefix={1}'.format(logging.target_bucket, logging.target_prefix))
```

关闭访问日志记录

以下代码用于关闭存储空间的访问日志记录：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

bucket.delete_bucket_logging()
```

防盗链

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。
- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

关于防盗链的更多介绍，请参见开发指南中的**设置防盗链**。

设置防盗链

以下代码用于设置防盗链：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import BucketReferer

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 设置允许空Referer（True表示允许空Referer，False表示不允许空Referer），并设置Referer白名单。
bucket.put_bucket_referer(BucketReferer(True, ['http://aliyun.com', 'http://*.aliyuncs.com']))
```

获取防盗链信息

以下代码用于获取防盗链信息：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

config = bucket.get_bucket_referer()
print('allow empty referer={0}, referers={1}'.format(config.allow_empty_referer, config.referers))
```

清空防盗链

以下代码用于清空防盗链：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

# 不能直接清空防盗链，需要新建一个允许空Referer的规则来覆盖之前的规则。
bucket.put_bucket_referer(BucketReferer(True, []))
```

图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

图片处理的详细信息请参见OSS图片处理指南。

图片处理功能

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放
- 图片裁剪
- 图片旋转
- 图片效果
- 图片水印，包括添加图片、文字、图文混合水印
- 自定义图片处理样式
- 级联处理，调用多个图片处理功能

图片处理使用

图片处理使用标准的HTTP GET请求。您可以在URL的QueryString中设置处理参数。

如果图片文件的访问权限为私有读写，必须通过授权才能进行访问。

匿名访问

您可以通过如下格式的三级域名匿名访问处理后的图片：

```
http://<yourBucketName>.<yourEndpoint>/<yourObjectName>?x-oss-
process=image/<yourAction>,<yourParameValue>
```

参数	描述
bucket	存储空间名称
endpoint	存储空间所在地域的访问域名
object	图片文件名称
image	图片处理的保留标志符
action	对图片做的操作，如缩放、裁剪、旋转等
param	对图片做的操作所对应的参数

基础操作

例如，将图缩略成宽度为100，高度按比例处理：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

自定义样式

使用如下格式的三级域名匿名访问图片处理：

```
http://<yourBucketName>.<yourEndpoint>/<yourObjectName>?x-oss-process=style/<yourStyleName>
```

- style：自定义样式的保留标志符。
- yourStyleName：自定义样式名称，即通过控制台自定义样式的规则名称。

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=style/oss-pic-style-w-100
```

级联处理

通过级联处理，可以对一张图片顺序进行多个操作，格式如下：

```
http://<yourBucketName>.<yourEndpoint>/<yourObjectName>?x-oss-
process=image/<yourAction1>,<yourParameValue1>/<yourAction2>,<yourParameValue2>/...
```

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100/rotate,90
```

支持HTTPS访问

图片服务支持HTTPS访问，例如：

```
https://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

授权访问

授权访问支持自定义样式、HTTPS和级联处理。

以下代码用于生成带签名的图片处理URL：

```
# -*- coding: utf-8 -*-
import oss2

# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'http://oss-cn-hangzhou.aliyuncs.com'
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
bucket_name = '<yourBucketName>'
key = 'example.jpg'

# 创建存储空间实例，所有文件相关的方法都需要通过存储空间实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
```

```
# 上传示例图片。
bucket.put_object_from_file(key, 'example.jpg')

# 生成带签名的URL，并指定过期时间为10分钟。过期时间单位是秒。
style = 'image/resize,m_fixed,w_100,h_100/rotate,90'
url = bucket.sign_url('GET', key, 10 * 60, params={'x-oss-process': style})
print(url)
```

SDK访问

对于任意权限的图片文件，都可以直接使用SDK访问和处理。

SDK处理图片文件支持自定义样式、HTTPS和级联处理。图片处理的完整代码请参见：GitHub。

基础操作

get_object和get_object_to_file支持图片处理。

以下代码展示了图片处理的基础操作：

```
# -*- coding: utf-8 -*-
import os
import oss2

# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'http://oss-cn-hangzhou.aliyuncs.com'
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
bucket_name = '<yourBucketName>'
key = 'example.jpg'
new_pic = 'example-new.jpg'

# 创建存储空间实例，所有文件相关的方法都需要通过存储空间实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)

# 上传示例图片。
bucket.put_object_from_file(key, 'example.jpg')

# 缩放
style = 'image/resize,m_fixed,w_100,h_100'
bucket.get_object_to_file(key, new_pic, process=style)

# 裁剪
style = 'image/crop,w_100,h_100,x_100,y_100,r_1'
bucket.get_object_to_file(key, new_pic, process=style)

# 旋转
style = 'image/rotate,90'
bucket.get_object_to_file(key, new_pic, process=style)

# 锐化
```

```
style = 'image/sharpen,100'
bucket.get_object_to_file(key, new_pic, process=style)

# 水印
style = 'image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ'
bucket.get_object_to_file(key, new_pic, process=style)

# 格式转换
style = 'image/format,png'
bucket.get_object_to_file(key, new_pic, process=style)

# 删除示例图片。
bucket.delete_object(key)
# 清除本地文件。
os.remove(new_pic)
```

自定义样式

以下代码用于自定义图片样式：

```
# -*- coding: utf-8 -*-
import os
import oss2

# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'http://oss-cn-hangzhou.aliyuncs.com'
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
bucket_name = '<yourBucketName>'
key = 'example.jpg'
new_pic = 'example-new.jpg'

# 创建存储空间实例，所有文件相关的方法都需要通过存储空间实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)

# 上传示例图片。
bucket.put_object_from_file(key, 'example.jpg')

# 自定义样式。
style = 'style/oss-pic-style-w-100'

# 图片处理。
bucket.get_object_to_file(key, new_pic, process=style)

# 删除示例图片。
bucket.delete_object(key)
# 清除本地文件。
os.remove(new_pic)
```

级联处理

以下代码用于级联处理图片：

```
# -*- coding: utf-8 -*-
import os
import oss2

# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'http://oss-cn-hangzhou.aliyuncs.com'
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
bucket_name = '<yourBucketName>'
key = 'example.jpg'
new_pic = 'example-new.jpg'

# 创建存储空间实例，所有文件相关的方法都需要通过存储空间实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)

# 上传示例图片。
bucket.put_object_from_file(key, 'example.jpg')

# 级联处理。
style = 'image/resize,m_fixed,w_100,h_100/rotate,90'

# 图片处理。
bucket.get_object_to_file(key, new_pic, process=style)

# 删除示例图片。
bucket.delete_object(key)
# 清除本地文件。
os.remove(new_pic)
```

图片处理工具

您可以通过可视化图片处理工具ImageStyleViever直观地看到OSS图片处理结果。

中文和时间

中文

在Python代码中如果使用了中文字符，运行时会出错。因此，您需要在代码的开头部分加入字符编码的声明，例如：

```
# -*- coding: utf-8 -*-
```

数据类型

Python 2.x中以下数据类型支持中文：

数据类型	描述
str	字符串。
unicode	unicode流。其长度是字符数，如u'中文'的长度是2。

Python 3.x中以下数据类型支持中文：

数据类型	描述
str	字符串。
bytes	字节流。其长度是字节数，如b'中文'的长度取决于编码，如果是UTF-8编码，则为6。

输入、输出类型约定

输入类型约定如下：

输入	类型	备注
OSS文件名	str	如为bytes，要求是UTF-8编码。
本地文件名	str, unicode	如为bytes，要求是UTF-8编码，例如bucket.get_object_to_file里的youLocalFile参数。
输入数据流	bytes	例如bucket.put_object里的data参数。

输出类型约定如下：

输出	类型	备注
解析XML得到的结果	str	例如通过bucket.list_object得到结果中的字符串
下载内容	bytes	Python SDK默认bytes类型经过UTF-8编码，请确保Python源文件也是UTF-8编码。

类型转换函数

Python SDK提供了三个用于类型转换的函数：

函数	描述
to_bytes	- Python 2.x中，把unicode转换为str。其他类型则原值返回。 - Python 3.x中，把str转换为bytes。其他类型则原值返回。
to_unicode	- Python 2.x中，把str转换为unicode。其他类型则原值返回。 - Python 3.x中，把bytes转换为str。其他类型则原值返回。
to_string	Python 2.x中相当于to_bytes。Python 3.x中相当于to_unicode。

时间

Python SDK会把从服务器获得的时间戳字符串（datetime.datetime类型的时间）都转换为Unix Time类型的时间，即自1970年1月1日UTC零点以来的秒数。例如bucket.get_object方法返回结果中的last_modified就是一个int类型的Unix Time。

您可以通过datetime.datetime.fromtimestamp()方法进行时间转换，得到时间戳字符串。

异常处理

OSS Python SDK异常（OssError）分为三类：ClientError、RequestError和ServerError，这些异常定义在oss2.exceptions子模块中。

异常的成员变量如下：

变量	类型	描述
status	int	- ServerError：HTTP状态码 - ClientError和RequestError：固定值
request_id	str	- ServerError：OSS服务器返回的请求ID - ClientError和RequestError：空字符串
code和message	str	对应OSS的错误响应格式里的Code和Message两个XML Tag中的文本。

异常处理示例

下面的代码展示了下载一个不存在文件时的异常处理，并打印出错误信息的HTTP状态码和请求ID。

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')

try:
stream = bucket.get_object('random-key.txt')
except oss2.exceptions.NoSuchKey as e:
print('status={0}, request_id={1}'.format(e.status, e.request_id))
```

ClientError

ClientError是由于客户端输入有误引起的。例如，使用bucket.batch_delete_objects方法时，如果收到空的文件列表，会抛出该异常。ClientError的status值是oss2.exceptions.OSS_CLIENT_ERROR_STATUS。

RequestError

当HTTP库抛出异常时，Python SDK会将其转换为RequestError。RequestError的status值是oss2.exceptions.OSS_REQUEST_ERROR_STATUS。

ServerError

当OSS服务器返回HTTP错误码时，Python SDK会将其转换为ServerError。ServerError根据HTTP状态码和OSS错误码派生出多个子类。其中NotFound子类对应所有404异常，Conflict子类对应所有409异常。

下表列出了常见的错误码：

异常类	HTTP状态码	OSS错误码	描述
NotModified	304	空	没有修改
AccessDenied	403	AccessDenied	拒绝访问
NoSuchBucket	404	NoSuchBucket	存储空间不存在
NoSuchKey	404	NoSuchKey	文件不存在
NoSuchUpload	404	NoSuchUpload	分片上传不存在
NoSuchWebsite	404	NoSuchWebsiteConfiguration	静态网站托管未配置

NoSuchLifecycle	404	NoSuchLifecycle	生命周期规则未配置
NoSuchCors	404	NoSuchCORSConfiguration	跨域资源共享未配置
BucketNotEmpty	409	BucketNotEmpty	存储空间非空
PositionNotEqualToLength	409	PositionNotEqualToLength	设置的追加位置和文件长度不等
ObjectNotAppendable	409	ObjectNotAppendable	不是可追加文件

Android-SDK

前言

简介

本文档主要介绍OSS Android SDK的安装和使用。本文档假设您已经开通了阿里云OSS 服务，并创建了AccessKeyId 和AccessKeySecret。文中的ID 指的是AccessKeyId，KEY 指的是AccessKeySecret。如果您还没有开通或者还不了解OSS，请登录OSS产品主页获取更多的帮助。

环境要求

- Android系统版本：2.3及以上
- 必须注册有阿里云账户，并开通OSS服务。

SDK下载

- Android SDK开发包
    - Android Studio环境下:

```
dependencies {
compile 'com.aliyun.dpa:oss-android-sdk:+'
}
```

- github地址：点击查看
- sample地址：点击查看

安装

Android Studio方式（推荐） Maven依赖

```
dependencies {
compile 'com.aliyun.dpa:oss-android-sdk:+'
compile 'com.squareup.okhttp3:okhttp:3.4.1'
compile 'com.squareup.okio:okio:1.9.0'
}
```

直接引入jar包

- 将aliyun-oss-sdk-android-x.x.x.jar、okhttp-3.x.x.jar 和 okio-1.x.x.jar 3 个 jar 包导入 libs 目录。
  其中aliyun-oss-sdk-android-x.x.x.jar参考如下

源码编译jar包

可以clone下工程源码之后，运行gradle命令打包：

```
# clone工程
$ git clone https://github.com/aliyun/aliyun-oss-android-sdk.git

# 进入目录
$ cd aliyun-oss-android-sdk/oss-android-sdk/

# 执行打包脚本，要求jdk 1.7
$ ../gradlew releaseJar

# 进入打包生成目录，jar包生成在该目录下
$ cd build/libs && ls
```

权限设置

以下是 OSS Android SDK 所需要的 Android 权限，请确保您的 AndroidManifest.xml 文件中已经配置了这些权限，否则，SDK 将无法正常工作。

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

混淆设置

在混淆配置中加入：

```
-keep class com.alibaba.sdk.android.oss.** { *; }
-dontwarn okio.**
-dontwarn org.apache.commons.codec.binary.**
```

初始化

OSSClient 是 OSS 服务的 Android 客户端，它为调用者提供了一系列的方法，可以用来操作，管理存储空间（bucket）和文件（object）等。在使用 SDK 发起对 OSS 的请求前，您需要初始化一个 OSSClient 实例，并对它进行一些必要设置。

OSSClient 生命周期和应用生命周期保持一致即可。在应用启动时创建一个全局的ossclient，在应用结束时销毁即可。

确定 Endpoint

Endpoint 是阿里云 OSS 服务在各个区域的地址，目前支持以下两种形式:

Endpoint类型	解释
OSS区域地址	使用OSS Bucket所在区域地址，各个区域Endpoint参考这里
用户自定义域名	用户自定义域名，且CNAME指向OSS域名

关于Endpoint，可以参考：点击查看。

OSS区域地址

使用OSS Bucket所在区域地址，Endpoint查询可以有下面两种方式：

- 查询Endpoint与区域对应关系详情，可以参考：点击查看。
- 您可以登录 阿里云OSS控制台，进入Bucket概览页，Bucket域名的后缀部分：如bucket-1.oss-cn-

hangzhou.aliyuncs.com的oss-cn-hangzhou.aliyuncs.com部分为该Bucket的外网Endpoint。

Cname

您可以将自己拥有的域名通过Cname绑定到某个存储空间（bucket）上，然后通过自己域名访问存储空间内的文件。

比如您要将域名new-image.xxxxx.com绑定到深圳区域的名称为image的存储空间上：您需要到您的域名xxxxx.com托管商那里设定一个新的域名解析，将http://new-image.xxxxx.com 解析到 http://image.oss-cn-shenzhen.aliyuncs.com ，类型为CNAME。

设置EndPoint和凭证

移动终端是一个不受信任的环境，把AccessKeyId和AccessKeySecret直接保存在终端用来加签请求，**存在极高的风险**。推荐使用***STS鉴权模式***或***自签名模式***，详细请参考：访问控制、移动端直传。

如果用STS鉴权模式，推荐使用OSSAuthCredentialsProvider方式直接访问鉴权应用服务器，token过期后可以自动更新。

更多信息可查看可查看sample 点击查看。

设置EndPoint和CredentialProvider示例如下：

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

String stsServer = "STS应用服务器地址，例如http://abc.com"
//推荐使用OSSAuthCredentialsProvider。token过期可以及时更新
OSSCredentialProvider credentialProvider = new OSSAuthCredentialsProvider(stsServer);

//该配置类如果不设置，会有默认配置，具体可看该类
ClientConfiguration conf = new ClientConfiguration();
conf.setConnectionTimeout(15 * 1000); // 连接超时，默认15秒
conf.setSocketTimeout(15 * 1000); // socket超时，默认15秒
conf.setMaxConcurrentRequest(5); // 最大并发请求数，默认5个
conf.setMaxErrorRetry(2); // 失败后最大重试次数，默认2次

OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider);
```

提示：

- 更多鉴权方式参考：访问控制。

设置EndPoint为cname

如果您已经在bucket上绑定cname，将该cname直接设置到endPoint即可。如：

```
String endpoint = "http://new-image.xxxxx.com";
...
OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider);
```

更多鉴权方式参考：访问控制

启用日志

移动端的使用环境比较复杂。会出现部分区域或某一个时段无法正常使用oss sdk的情况。为了进一步方便开发者定位问题，oss sdk 在开启日志记录功能后，会将一些log信息记录在本地。如需开启需要在OSSClient使用之前进行初始化，调用如下方法即可。

补充说明：

- 日志文件位置在内置sd卡路径\OSSLog\logs.csv。
- 使用者可以自行选择将文件上传至服务器，进一步追踪问题。
- 也可以接入阿里云SLS日志服务进行日志文件上传, 日志服务传送门。

```
//日志的样式
//通过调用OSSLog.enableLog()开启可以在控制台看到日志，
//并且会支持写入手机sd卡中的一份日志文件位置在内置sd卡路径\OSSLog\logs.csv 默认不开启
//日志会记录oss操作行为中的请求数据，返回数据，异常信息
//例如requestId,response header等,下边是一个日志记录case
//android_version：5.1 android版本
//mobile_model：XT1085 android手机型号
//network_state：connected 网络状况
//network_type：WIFI 网络连接类型
//具体的操作行为信息:
//[2017-09-05 16:54:52] - Encounter local execpiton: //java.lang.IllegalArgumentException: The bucket name is invalid.
//A bucket name must:
//1) be comprised of lower-case characters, numbers or dash(-);
//2) start with lower case or numbers;
//3) be between 3-63 characters long.
//------>end of log
OSSLog.enableLog(); //调用此方法即可开启日志
```

设置网络参数

也可以在初始化的时候设置详细的ClientConfiguration：

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

// 在移动端建议使用STS的方式初始化OSSClient，更多信息参考：[访问控制]
OSSCredentialProvider credentialProvider = new OSSStsTokenCredentialProvider("<StsToken.AccessKeyId>", "<StsToken.SecretKeyId>", "<StsToken.SecurityToken>");


ClientConfiguration conf = new ClientConfiguration();
conf.setConnectionTimeout(15 * 1000); // 连接超时，默认15秒
conf.setSocketTimeout(15 * 1000); // socket超时，默认15秒
conf.setMaxConcurrentRequest(5); // 最大并发请求书，默认5个
conf.setMaxErrorRetry(2); // 失败后最大重试次数，默认2次
```

```
OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider, conf);
```

对 SDK 中同步接口、异步接口的一些说明

考虑到移动端开发场景下不允许在UI线程执行网络请求的编程规范，SDK大多数接口都提供了同步、异步两种调用方式，同步接口调用后会阻塞等待结果返回，而异步接口需要在请求时传入回调函数，请求的执行结果将在回调中处理。

同步接口不能在UI线程调用。遇到异常时，将直接抛出ClientException或者ServiceException异常，前者指本地遇到的异常如网络异常、参数非法等；后者指OSS返回的服务异常，如鉴权失败、服务器错误等。

异步请求遇到异常时，异常会在回调函数中处理。

此外，调用异步接口时，函数会直接返回一个Task，Task可以取消、等待直到完成、或者直接获取结果。如：

```
OSSAsyncTask task = oss.asyncGetObejct(...);
task.cancel(); // 可以取消任务
task.waitUntilFinished(); // 等待直到任务完成
GetObjectResult result = task.getResult(); // 阻塞等待结果返回
```

接口支持同步和异步两种调用方式，考虑到简洁性，本文档中只有部分重要接口会同时提供同步、异步两种调用的示例，其他接口暂时以异步调用的示例为主。

设置是否开启DNS配置

```
ClientConfiguration conf = new ClientConfiguration();
conf.setHttpDnsEnable(true);//默认是true 开启，需要关闭可以设置false
```

设置自定义user-agent

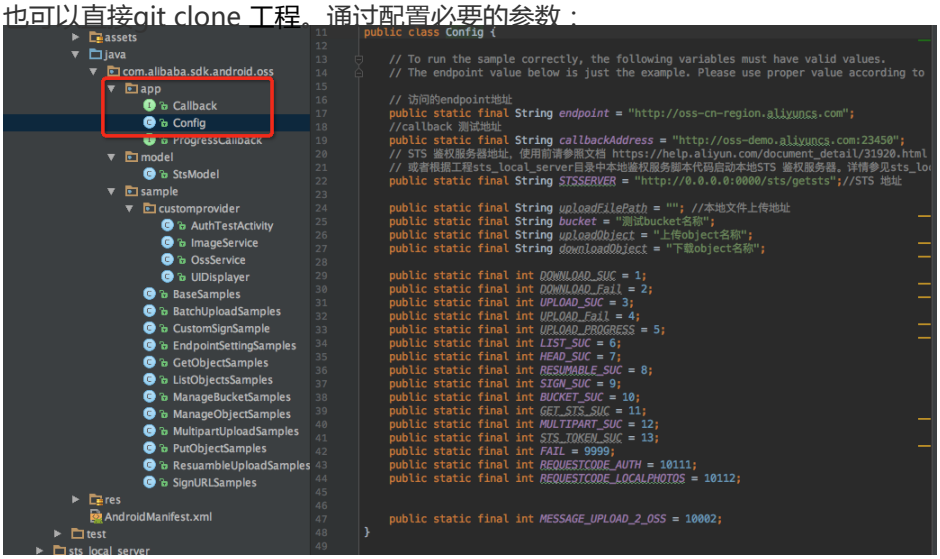
```
ClientConfiguration conf = new ClientConfiguration();
//设置的ua会默认添加到sdk默认ua的最后边，取得时候自行去最后一个即可。
conf.setUserAgentMark("customUserAgent");
```

快速入门

以下演示了上传、下载文件的基本流程。更多细节用法可以参考本工程的：

sample目录(包含上传本地文件，下载文件，断点续传，设置callback回调等示例)。点击查看。

也可以直接git clone 工程，通过配置必要的参数：



然后运行工程，demo 如下：

STEP-1. 初始化OSSClient

初始化主要完成Endpoint设置、鉴权方式设置、Client参数设置。其中，鉴权方式包含自签名模式、STS鉴权

模式。如果要使用STS鉴权请先阅读访问控制章节了解RAM的基础知识。以下内容假设您已开通RAM服务并了解RAM相关内容。了解如何获取子账户AccessKeyId，SecretKeyId以及RoleArn信息。

完善脚本文件中AccessKeyId，SecretKeyId以及RoleArn参数信息。通过python可以启动一个本机http服务。在客户端代码中访问本地服务从而获得StsToken.AccessKeyId,StsToken.SecretKeyId以及StsToken.SecurityToken。

更多信息可查看sample 中sts 使用方式点击查看。

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

// 在移动端建议使用STS方式初始化OSSClient。
// 更多信息可查看sample 中 sts 使用方式(https://github.com/aliyun/aliyun-oss-android-sdk/tree/master/app/src/main/java/com/alibaba/sdk/android/oss/app)
OSSCredentialProvider credentialProvider = new OSSStsTokenCredentialProvider("<StsToken.AccessKeyId>", "<StsToken.SecretKeyId>", "<StsToken.SecurityToken>");

//该配置类如果不设置，会有默认配置，具体可看该类
ClientConfiguration conf = new ClientConfiguration();
conf.setConnectionTimeout(15 * 1000); // 连接超时，默认15秒
conf.setSocketTimeout(15 * 1000); // socket超时，默认15秒
conf.setMaxConcurrentRequest(5); // 最大并发请求数，默认5个
conf.setMaxErrorRetry(2); // 失败后最大重试次数，默认2次

//开启可以在控制台看到日志，并且会支持写入手机sd卡中的一份日志文件位置在SDCard_path\OSSLog\logs.csv 默认不开启
//日志会记录oss操作行为中的请求数据，返回数据，异常信息
//例如requestId,response header等
//android_version：5.1 android版本
//mobile_model：XT1085 android手机型号
//network_state：connected 网络状况
//network_type：WIFI 网络连接类型
//具体的操作行为信息:
//[2017-09-05 16:54:52] - Encounter local execpiton: //java.lang.IllegalArgumentException: The bucket name is invalid.
//A bucket name must:
//1) be comprised of lower-case characters, numbers or dash(-);
//2) start with lower case or numbers;
//3) be between 3-63 characters long.
//------>end of log
OSSLog.enableLog();

OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider);
```

通过OSSClient发起上传、下载请求是线程安全的，您可以并发执行多个任务。

STEP-2. 上传文件

这里假设您已经在控制台上拥有自己的Bucket，这里演示如何从把一个本地文件上传到OSS:

```
// 构造上传请求
PutObjectRequest put = new PutObjectRequest("<bucketName>", "<objectKey>", "<uploadFilePath>");
```

```
// 异步上传时可以设置进度回调
put.setProgressCallback(new OSSProgressCallback<PutObjectRequest>() {
@Override
public void onProgress(PutObjectRequest request, long currentSize, long totalSize) {
Log.d("PutObject", "currentSize: " + currentSize + " totalSize: " + totalSize);
}
});

OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<PutObjectRequest, PutObjectResult>() {
@Override
public void onSuccess(PutObjectRequest request, PutObjectResult result) {
Log.d("PutObject", "UploadSuccess");
}

@Override
public void onFailure(PutObjectRequest request, ClientException clientExcepion, ServiceException serviceException)
{
// 请求异常
if (clientExcepion != null) {
// 本地异常如网络异常等
clientExcepion.printStackTrace();
}
if (serviceException != null) {
// 服务异常
Log.e("ErrorCode", serviceException.getErrorCode());
Log.e("RequestId", serviceException.getRequestId());
Log.e("HostId", serviceException.getHostId());
Log.e("RawMessage", serviceException.getRawMessage());
}
}
});

// task.cancel(); // 可以取消任务

// task.waitUntilFinished(); // 可以等待直到任务完成
```

STEP-3. 下载指定文件

下载一个指定object，返回数据的输入流，需要自行处理:

```
// 构造下载文件请求
GetObjectRequest get = new GetObjectRequest("<bucketName>", "<objectKey>");

OSSAsyncTask task = oss.asyncGetObject(get, new OSSCompletedCallback<GetObjectRequest, GetObjectResult>()
{
@Override
public void onSuccess(GetObjectRequest request, GetObjectResult result) {
// 请求成功
Log.d("Content-Length", "" + getResult.getContentLength());

InputStream inputStream = result.getObjectContent();

byte[] buffer = new byte[2048];
```

```
int len;

try {
while ((len = inputStream.read(buffer)) != -1) {
// 处理下载的数据
}
} catch (IOException e) {
e.printStackTrace();
}
}

@Override
public void onFailure(GetObjectRequest request, ClientException clientExcepion, ServiceException serviceException)
{
// 请求异常
if (clientExcepion != null) {
// 本地异常如网络异常等
clientExcepion.printStackTrace();
}
if (serviceException != null) {
// 服务异常
Log.e("ErrorCode", serviceException.getErrorCode());
Log.e("RequestId", serviceException.getRequestId());
Log.e("HostId", serviceException.getHostId());
Log.e("RawMessage", serviceException.getRawMessage());
}
}
});

// task.cancel(); // 可以取消任务

// task.waitUntilFinished(); // 如果需要等待任务完成
```

访问控制

访问控制

移动终端是一个不受信任的环境。为此，SDK提供了两种依赖于您的业务Server的鉴权模式： STS鉴权模式 和 自签名模式 。

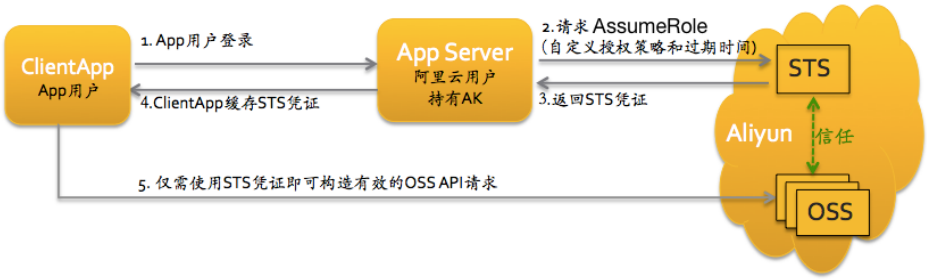
STS鉴权模式

介绍

OSS可以通过阿里云STS服务，临时进行授权访问。阿里云STS (Security Token Service) 是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或联邦用户（用户身份由您自己管理）颁发一个自定义时效和权限的访问凭证，App端称为FederationToken。第三方应用或联邦用户可以使用该访问凭证直接调用阿里云产品API，或者使用阿里云产品提供的SDK来访问云产品API。

- 您不需要透露您的长期密钥(AccessKey)给第三方应用，只需要生成一个访问令牌并将令牌交给第三方应用即可。这个令牌的访问权限及有效期限都可以由您自定义。
- 您不需要关心权限撤销问题，访问令牌过期后就自动失效。

以APP应用为例，交互流程如下图：



方案的详细描述如下：

1. App用户登录。App用户身份是您自己管理。您可以自定义身份管理系统，也可以使用外部Web账号或OpenID。对于每个有效的App用户来说，AppServer可以确切地定义出每个App用户的最小访问权限。
2. AppServer请求STS服务获取一个安全令牌(SecurityToken)。在调用STS之前，AppServer需要确定App用户的最小访问权限（用Policy语法描述）以及授权的过期时间。然后通过调用STS的AssumeRole(扮演角色)接口来获取安全令牌。角色管理与使用相关内容请参考RAM使用指南中的角色管理。
3. STS返回给AppServer一个有效的访问凭证，App端称为FederationToken，包括一个安全令牌(SecurityToken)、临时访问密钥(AccessKeyId, AccessKeySecret)以及过期时间。
4. AppServer将FederationToken返回给ClientApp。ClientApp可以缓存这个凭证。当凭证失效时，ClientApp需要向AppServer申请新的有效访问凭证。比如，访问凭证有效期为1小时，那么ClientApp可以每30分钟向AppServer请求更新访问凭证。
5. ClientApp使用本地缓存的FederationToken去请求Aliyun Service API。云服务会感知STS访问凭证，并会依赖STS服务来验证访问凭证，并正确响应用户请求。

STS安全令牌详情，请参考《RAM使用指南》中的角色管理。关键是调用STS服务接口AssumeRole来获取有效访问凭证即可。也可以直接使用STS SDK来调用该方法，点击查看。

使用这种模式授权需要先开通阿里云RAM服务。

STS使用手册：点击查看

OSS授权策略配置：点击查看

直接设置StsToken

您可以在APP中，预先通过某种方式(如通过网络请求从您的业务Server上)获取一对StsToken，然后用它来初始化SDK。采取这种使用方式，您需要格外关注StsToken的过期时间，在StsToken即将过期时，需要您主动更新新的StsToken到SDK中。

初始化代码为：

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

OSSCredentialProvider credentialProvider = new OSSStsTokenCredentialProvider("<StsToken.AccessKeyId>",
"<StsToken.SecretKeyId>", "<StsToken.SecurityToken>");

OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider);
```

在您判断到Token即将过期时，您可以重新构造新的OSSClient，也可以通过如下方式更新CredentialProvider:

```
oss.updateCredentialProvider(new OSSStsTokenCredentialProvider("<StsToken.AccessKeyId>",
"<StsToken.SecretKeyId>", "<StsToken.SecurityToken>"));
```

实现获取StsToken回调

如果您期望SDK能自动帮您管理Token的更新，那么，您需要告诉SDK如何获取Token。在SDK的应用中，您需要实现一个回调，这个回调通过您实现的方式去获取一个Federation Token(即StsToken)，然后返回。SDK会利用这个Token来进行加签处理，并在需要更新时主动调用这个回调获取Token，如图示：

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

OSSCredentialProvider credentialProvider = new OSSFederationCredentialProvider() {

@Override
public OSSFederationToken getFederationToken() {
// 您需要在这里实现获取一个FederationToken，并构造成OSSFederationToken对象返回
// 如果因为某种原因获取失败，可直接返回nil

OSSFederationToken * token;
// 下面是一些获取token的代码，比如从您的server获取
...
return token;
}
};

OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider);
```

此外，如果您已经通过别的方式拿到token所需的各个字段，也可以在这个回调中直接返回。如果这么做的话，您需要自己处理token的更新，更新后重新设置该OSSClient实例的OSSCredentialProvider。

使用示例：

假设您搭建的server地址为: http://localhost:8080/distribute-token.json ，并假设访问这个地址，返回的数

据如下：

```
{
"StatusCode": 200,
"AccessKeyId":"STS.iA645eTOXEqP3cg3VeHf",
"AccessKeySecret":"rV3VQrpFQ4BsyHSAvi5NVLpPIVffDJv4LojUBZCf",
"Expiration":"2015-11-03T09:52:59Z",
"SecurityToken":"CAES7QIIARKAAZPlqaN9ILiQZPS+JDkS/GSZN45RLx4YS/p3OgaUC+oJl3XSlbJ7StKpQ...."}
```

那么，您可以这么实现一个OSSFederationCredentialProvider实例：

```
OSSCredentialProvider credetialProvider = new OSSFederationCredentialProvider() {
@Override
public OSSFederationToken getFederationToken() {
try {
URL stsUrl = new URL("http://localhost:8080/distribute-token.json");
HttpURLConnection conn = (HttpURLConnection) stsUrl.openConnection();
InputStream input = conn.getInputStream();
String jsonText = IOUtils.readStreamAsString(input, OSSConstants.DEFAULT_CHARSET_NAME);
JSONObject jsonObjs = new JSONObject(jsonText);
String ak = jsonObjs.getString("AccessKeyId");
String sk = jsonObjs.getString("AccessKeySecret");
String token = jsonObjs.getString("SecurityToken");
String expiration = jsonObjs.getString("Expiration");
return new OSSFederationToken(ak, sk, token, expiration);
} catch (Exception e) {
e.printStackTrace();
}
return null;
}
};
```

自签名模式

您可以把AccessKeyId/AccessKeySecret保存在您的业务server，然后在SDK实现回调，将需要加签的合并好的签名串POST到server，您在业务server对这个串按照OSS规定的签名算法签名之后，返回给该回调函数，再由回调返回。

签名算法参考：**点击查看**

content是已经根据请求各个参数拼接后的字符串，所以算法为：

```
signature = "OSS " + AccessKeyId + ":" + base64(hmac-sha1(AccessKeySecret, content))
```

代码如下：

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

credentialProvider = new OSSCustomSignerCredentialProvider() {
@Override
```

```
public String signContent(String content) {
// 您需要在这里依照OSS规定的签名算法，实现加签一串字符内容，并把得到的签名传拼接上AccessKeyId后返回
// 一般实现是，将字符内容post到您的业务服务器，然后返回签名
// 如果因为某种原因加签失败，描述error信息后，返回nil

// 以下是用本地算法进行的演示
return "OSS " + AccessKeyId + ":" + base64(hmac-sha1(AccessKeySecret, content));
}
};

OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider);
```

特别注意： 无论是STS鉴权模式，还是自签名模式，您实现的回调函数，都需要保证调用时返回结果。所以，如果您在其中实现了向业务server获取token、signature的网络请求，建议调用网络库的同步接口。回调都是在SDK具体请求的时候，在请求的子线程中执行，所以不会阻塞主线程。

上传文件

OSS 移动端SDK 上传文件的方式可以分为：简单上传，追加上传，分片上传和断点续传。

简单上传本地文件

调用同步接口上传：

```
// 构造上传请求
PutObjectRequest put = new PutObjectRequest("<bucketName>", "<objectKey>", "<uploadFilePath>");

// 文件元信息的设置是可选的
// ObjectMetadata metadata = new ObjectMetadata();
// metadata.setContentType("application/octet-stream"); // 设置content-type
// metadata.setContentMD5(BinaryUtil.calculateBase64Md5(uploadFilePath)); // 校验MD5
// put.setMetadata(metadata);

try {

PutObjectResult putResult = oss.putObject(put);

Log.d("PutObject", "UploadSuccess");

Log.d("ETag", putResult.getETag());
Log.d("RequestId", putResult.getRequestId());
} catch (ClientException e) {
// 本地异常如网络异常等
e.printStackTrace();
} catch (ServiceException e) {
// 服务异常
```

```
Log.e("RequestId", e.getRequestId());
Log.e("ErrorCode", e.getErrorCode());
Log.e("HostId", e.getHostId());
Log.e("RawMessage", e.getRawMessage());
}
```

注意：在Android中，不能在UI线程调用同步接口，只能在子线程调用，否则将出现异常。如果希望直接在UI线程中上传，请使用异步接口。

调用异步接口上传：

```
// 构造上传请求
PutObjectRequest put = new PutObjectRequest("<bucketName>", "<objectKey>", "<uploadFilePath>");

// 异步上传时可以设置进度回调
put.setProgressCallback(new OSSProgressCallback<PutObjectRequest>() {
@Override
public void onProgress(PutObjectRequest request, long currentSize, long totalSize) {
Log.d("PutObject", "currentSize: " + currentSize + " totalSize: " + totalSize);
}
});

OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<PutObjectRequest, PutObjectResult>() {
@Override
public void onSuccess(PutObjectRequest request, PutObjectResult result) {
Log.d("PutObject", "UploadSuccess");

Log.d("ETag", result.getETag());
Log.d("RequestId", result.getRequestId());
}

@Override
public void onFailure(PutObjectRequest request, ClientException clientExcepion, ServiceException serviceException)
{
// 请求异常
if (clientExcepion != null) {
// 本地异常如网络异常等
clientExcepion.printStackTrace();
}
if (serviceException != null) {
// 服务异常
Log.e("ErrorCode", serviceException.getErrorCode());
Log.e("RequestId", serviceException.getRequestId());
Log.e("HostId", serviceException.getHostId());
Log.e("RawMessage", serviceException.getRawMessage());
}
}
});

// task.cancel(); // 可以取消任务
// task.waitUntilFinished(); // 可以等待任务完成
```

简单上传二进制byte[]数组

```
byte[] uploadData = new byte[100 * 1024];
new Random().nextBytes(uploadData);

// 构造上传请求
PutObjectRequest put = new PutObjectRequest(testBucket, testObject, uploadData);

try {
PutObjectResult putResult = oss.putObject(put);

Log.d("PutObject", "UploadSuccess");

Log.d("ETag", putResult.getETag());
Log.d("RequestId", putResult.getRequestId());
} catch (ClientException e) {
// 本地异常如网络异常等
e.printStackTrace();
} catch (ServiceException e) {
// 服务异常
Log.e("RequestId", e.getRequestId());
Log.e("ErrorCode", e.getErrorCode());
Log.e("HostId", e.getHostId());
Log.e("RawMessage", e.getRawMessage());
}
```

上传到文件目录

OSS服务是没有文件夹这个概念的，所有元素都是以文件来存储，但给用户提供了创建模拟文件夹的方式。创建模拟文件夹本质上来说是创建了一个名字以“/”结尾的文件，这个文件可以上传下载，只是控制台会对以“/”结尾的文件以文件夹的方式展示。

在上传文件时，如果把ObjectKey写为"folder/subfolder/file"，即是模拟了把文件上传到folder/subfolder/下的file文件。

> 注意：OSS路径默认是”根目录”，不需要以‘/’开头。

上传Content-Type设置

在Web服务中Content-Type用来设定文件的类型，决定以什么形式、什么编码读取这个文件。某些情况下，对于上传的文件需要设定Content-Type，否则文件不能以自己需要的形式和编码来读取。使用SDK上传文件时，如果不指定Content-Type，SDK会帮您根据后缀自动添加Content-Type。

```
// 构造上传请求
PutObjectRequest put = new PutObjectRequest("<bucketName>", "<objectKey>", "<uploadFilePath>");

ObjectMetadata metadata = new ObjectMetadata();
```

```
// 指定Content-Type
metadata.setContentType("application/octet-stream");
// user自定义metadata
metadata.addUserMetadata("x-oss-meta-name1", "value1");
put.setMetadata(metadata);

OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<PutObjectRequest, PutObjectResult>() {
...
});
```

Append Object 追加上传

Append Object以追加写的方式上传文件。通过Append Object操作创建的Object类型为Appendable Object，而通过Put Object上传的Object是Normal Object。

```
AppendObjectRequest append = new AppendObjectRequest(testBucket, testObject, uploadFilePath);

ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentType("application/octet-stream");
append.setMetadata(metadata);

// 设置追加位置
append.setPosition(0);

append.setProgressCallback(new OSSProgressCallback<AppendObjectRequest>() {
@Override
public void onProgress(AppendObjectRequest request, long currentSize, long totalSize) {
Log.d("AppendObject", "currentSize: " + currentSize + " totalSize: " + totalSize);
}
});

OSSAsyncTask task = oss.asyncAppendObject(append, new OSSCompletedCallback<AppendObjectRequest,
AppendObjectResult>() {
@Override
public void onSuccess(AppendObjectRequest request, AppendObjectResult result) {
Log.d("AppendObject", "AppendSuccess");
Log.d("NextPosition", "" + result.getNextPosition());
}

@Override
public void onFailure(AppendObjectRequest request, ClientException clientExcepion, ServiceException
serviceException) {
// 异常处理
}
});
```

用户使用Append方式上传，关键要对Position这个参数进行正确的设置：

当用户创建一个Appendable Object时，追加位置设为0。

当对Appendable Object进行内容追加时，追加位置设为Object当前长度。有两种方式获取该

Object长度：

- 通过上传追加后的返回内容获取。
- 通过head object获取文件长度。

上传后回调通知

客户端在上传Object时可以指定OSS服务端在处理完上传请求后，通知您的业务服务器，在该服务器确认接收了该回调后将回调的结果返回给客户端。因为加入了回调请求和响应的过程，相比简单上传，使用回调通知机制一般会导致客户端花费更多的等待时间。

具体说明参考：Callback

代码示例：

```
PutObjectRequest put = new PutObjectRequest(testBucket, testObject, uploadFilePath);

put.setCallbackParam(new HashMap<String, String>() {
{
put("callbackUrl", "110.75.82.106/callback");
put("callbackHost", "oss-cn-hangzhou.aliyuncs.com");
put("callbackBodyType", "application/json");
put("callbackBody", "{\"mimeType\":${mimeType},\"size\":${size}}");
}
});

OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<PutObjectRequest, PutObjectResult>() {
@Override
public void onSuccess(PutObjectRequest request, PutObjectResult result) {
Log.d("PutObject", "UploadSuccess");

// 只有设置了servercallback，这个值才有数据
String serverCallbackReturnJson = result.getServerCallbackReturnBody();

Log.d("servercallback", serverCallbackReturnJson);
}

@Override
public void onFailure(PutObjectRequest request, ClientException clientExcepion, ServiceException serviceException)
{
// 异常处理
}
});
```

如果需要支持自定义参数，参考如下设置：

```
put.setCallbackParam(new HashMap<String, String>() {
{
put("callbackUrl", "http://182.92.192.125/leibin/notify.php");
put("callbackHost", "oss-cn-hangzhou.aliyuncs.com");
put("callbackBodyType", "application/json");
```

```
put("callbackBody", "{\"object\":${object},\"size\":${size},\"my_var1\":${x:var1},\"my_var2\":${x:var2}}");
}
});

put.setCallbackVars(new HashMap<String, String>() {
{
put("x:var1", "value1");
put("x:var2", "value2");
}
});
```

分片上传

因为篇幅的原因，分片上传参考：分片上传

断点续传

特别注意：

> 断点续传暂时只支持上传本地文件。
>
> 对于移动端来说，如果不是比较大的文件，不建议使用这种方式上传，因为断点续传是通过分片上传实现的，上传单个文件需要进行多次网络请求，效率不高。**

在无线网络下，上传比较大的文件持续时间长，可能会遇到因为网络条件差、用户切换网络等原因导致上传中途失败，整个文件需要重新上传。为此，SDK提供了断点上传功能。

在上传前，可以指定断点记录的保存文件夹。若不进行此项设置，断点上传只在本次上传生效，某个分片因为网络原因等上传失败时会进行重试，避免整个大文件重新上传，节省重试时间和耗用流量。如果设置了断点记录的保存文件夹，如果任务失败，在下次重新启动任务，上传同一文件到同一Bucket、Object时，如果用户设置取消时不删除断点记录。再次上传将从断点记录处继续上传。详见随后的范例。

断点续传失败时，如果同一任务一直得不到续传，可能会在OSS上积累无用碎片。对这种情况，可以为Bucket设置lifeCycle规则，定时清理碎片。参考：生命周期管理。

说明：

> 断点续传的实现依赖InitMultipartUpload/UploadPart/ListParts/CompleteMultipartUpload/AbortMultipartUpload，如果采用STS鉴权模式，请注意加上这些API所需的权限。
>
> 断点续传也支持上传后回调通知，用法和上述普通上传回调通知一致。

> 断点续传已经默认开启每个分片上传时的Md5校验，请勿重复在request中设置Content-Md5头部。

不在本地持久保存断点记录的调用方式：

```
// 创建断点上传请求
ResumableUploadRequest request = new ResumableUploadRequest("<bucketName>", "<objectKey>",
"<uploadFilePath>");
// 设置上传过程回调
request.setProgressCallback(new OSSProgressCallback<ResumableUploadRequest>() {
@Override
public void onProgress(ResumableUploadRequest request, long currentSize, long totalSize) {
Log.d("resumableUpload", "currentSize: " + currentSize + " totalSize: " + totalSize);
}
});
// 异步调用断点上传
OSSAsyncTask resumableTask = oss.asyncResumableUpload(request, new
OSSCompletedCallback<ResumableUploadRequest, ResumableUploadResult>() {
@Override
public void onSuccess(ResumableUploadRequest request, ResumableUploadResult result) {
Log.d("resumableUpload", "success!");
}

@Override
public void onFailure(ResumableUploadRequest request, ClientException clientExcepion, ServiceException
serviceException) {
// 异常处理
}
});

// resumableTask.waitUntilFinished(); // 可以等待直到任务完成
```

在本地持久保存断点记录的调用方式：

```
String recordDirectory = Environment.getExternalStorageDirectory().getAbsolutePath() + "/oss_record/";

File recordDir = new File(recordDirectory);

// 要保证目录存在，如果不存在则主动创建
if (!recordDir.exists()) {
recordDir.mkdirs();
}

// 创建断点上传请求，参数中给出断点记录文件的保存位置，需是一个文件夹的绝对路径
ResumableUploadRequest request = new ResumableUploadRequest("<bucketName>", "<objectKey>",
"<uploadFilePath>", recordDirectory);
// 设置上传过程回调
request.setProgressCallback(new OSSProgressCallback<ResumableUploadRequest>() {
@Override
public void onProgress(ResumableUploadRequest request, long currentSize, long totalSize) {
Log.d("resumableUpload", "currentSize: " + currentSize + " totalSize: " + totalSize);
}
```

```
});


OSSAsyncTask resumableTask = oss.asyncResumableUpload(request, new
OSSCompletedCallback<ResumableUploadRequest, ResumableUploadResult>() {
@Override
public void onSuccess(ResumableUploadRequest request, ResumableUploadResult result) {
Log.d("resumableUpload", "success!");
}

@Override
public void onFailure(ResumableUploadRequest request, ClientException clientExcepion, ServiceException
serviceException) {
// 异常处理
}
});

// resumableTask.waitUntilFinished();
```

断点续传功能实现：

```
//调用OSSAsyncTask cancel()方法时是否需要删除断点记录文件的设置
String recordDirectory = Environment.getExternalStorageDirectory().getAbsolutePath() + "/oss_record/";

File recordDir = new File(recordDirectory);

// 要保证目录存在，如果不存在则主动创建
if (!recordDir.exists()) {
recordDir.mkdirs();
}

// 创建断点上传请求，参数中给出断点记录文件的保存位置，需是一个文件夹的绝对路径
ResumableUploadRequest request = new ResumableUploadRequest("<bucketName>", "<objectKey>",
"<uploadFilePath>", recordDirectory);
//设置false,取消时，不删除断点记录文件，如果不进行设置，默认true，是会删除断点记录文件，下次再进行上传时会重新上
传。
request.setDeleteUploadOnCancelling(false);
// 设置上传过程回调
request.setProgressCallback(new OSSProgressCallback<ResumableUploadRequest>() {
@Override
public void onProgress(ResumableUploadRequest request, long currentSize, long totalSize) {
Log.d("resumableUpload", "currentSize: " + currentSize + " totalSize: " + totalSize);
}
});


OSSAsyncTask resumableTask = oss.asyncResumableUpload(request, new
OSSCompletedCallback<ResumableUploadRequest, ResumableUploadResult>() {
@Override
public void onSuccess(ResumableUploadRequest request, ResumableUploadResult result) {
Log.d("resumableUpload", "success!");
}

@Override
public void onFailure(ResumableUploadRequest request, ClientException clientExcepion, ServiceException
```

```
serviceException) {
// 异常处理
}
});

// resumableTask.waitUntilFinished();
```

数据完整性校验

因为移动端网络环境的复杂性，OSS SDK提供了基于MD5和CRC64的端到端的数据完整性验证功能。

MD5校验

需要在上传文件时提供文件的Content-MD5值，OSS服务器会帮助用户进行MD5校验，只有在OSS服务器计算接收到的文件得到的MD5值和上传提供的MD5一致时才可以上传成功，从而保证上传数据的一致性。

```
// 构造上传请求
PutObjectRequest put = new PutObjectRequest("<bucketName>", "<objectKey>", "<uploadFilePath>");

ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentType("application/octet-stream");
try {
// 设置Md5以便校验
metadata.setContentMD5(BinaryUtil.calculateBase64Md5("<uploadFilePath>")); // 如果是从文件上传
// metadata.setContentMD5(BinaryUtil.calculateBase64Md5(byte[])); // 如果是上传二进制数据
} catch (IOException e) {
e.printStackTrace();
}
put.setMetadata(metadata);

OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<PutObjectRequest, PutObjectResult>() {
...
});
```

CRC校验

与MD5相比，CRC64可以边上传边计算CRC值。

```
// 构造上传请求
PutObjectRequest put = new PutObjectRequest("<bucketName>", "<objectKey>", "<uploadFilePath>");
// 开启crc效验后。如果在传输中数据不一致，会直接抛出ClientException异常。提示InconsistentException: inconsistent
object
put.setCRC64(OSSRequest.CRC64Config.YES);

OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<PutObjectRequest, PutObjectResult>() {
@Override
public void onSuccess(PutObjectRequest request, PutObjectResult result) {
//.....
}

@Override
public void onFailure(PutObjectRequest request, ClientException clientExcepion, ServiceException serviceException)
```

```
{
//.....
if (clientExcepion != null) {
// client side exception, such as network exception
// 检查是否有InconsistentException: inconsistent object信息
clientExcepion.getMessage();
}
}
});
```

下载文件

简单下载

下载指定文件，下载后将获得文件的输入流，此操作要求用户对该Object有读权限。同步调用：

```
//构造下载文件请求
GetObjectRequest get = new GetObjectRequest("<bucketName>", "<objectKey>");

//设置下载进度回调
get.setProgressListener(new OSSProgressCallback<GetObjectRequest>() {
@Override
public void onProgress(GetObjectRequest request, long currentSize, long totalSize) {
OSSLog.logDebug("getobj_progress: " + currentSize+" total_size: " + totalSize, false);
}
});

try {
// 同步执行下载请求，返回结果
GetObjectResult getResult = oss.getObject(get);

Log.d("Content-Length", "" + getResult.getContentLength());

// 获取文件输入流
InputStream inputStream = getResult.getObjectContent();

byte[] buffer = new byte[2048];
int len;

while ((len = inputStream.read(buffer)) != -1) {
// 处理下载的数据，比如图片展示或者写入文件等
}

// 下载后可以查看文件元信息
ObjectMetadata metadata = getResult.getMetadata();
Log.d("ContentType", metadata.getContentType());
```

```
} catch (ClientException e) {
// 本地异常如网络异常等
e.printStackTrace();
} catch (ServiceException e) {
// 服务异常
Log.e("RequestId", e.getRequestId());
Log.e("ErrorCode", e.getErrorCode());
Log.e("HostId", e.getHostId());
Log.e("RawMessage", e.getRawMessage());
} catch (IOException e) {
e.printStackTrace();
}
```

异步调用：

```
GetObjectRequest get = new GetObjectRequest("<bucketName>", "<objectKey>");
//设置下载进度回调
get.setProgressListener(new OSSProgressCallback<GetObjectRequest>() {
@Override
public void onProgress(GetObjectRequest request, long currentSize, long totalSize) {
OSSLog.logDebug("getobj_progress: " + currentSize+" total_size: " + totalSize, false);
}
});
OSSAsyncTask task = oss.asyncGetObject(get, new OSSCompletedCallback<GetObjectRequest, GetObjectResult>()
{
@Override
public void onSuccess(GetObjectRequest request, GetObjectResult result) {
// 请求成功
InputStream inputStream = result.getObjectContent();

byte[] buffer = new byte[2048];
int len;

try {
while ((len = inputStream.read(buffer)) != -1) {
// 处理下载的数据
}
} catch (IOException e) {
e.printStackTrace();
}
}

@Override
public void onFailure(GetObjectRequest request, ClientException clientExcepion, ServiceException serviceException)
{
// 请求异常
if (clientExcepion != null) {
// 本地异常如网络异常等
clientExcepion.printStackTrace();
}
if (serviceException != null) {
// 服务异常
Log.e("ErrorCode", serviceException.getErrorCode());
Log.e("RequestId", serviceException.getRequestId());
```

```
Log.e("HostId", serviceException.getHostId());
Log.e("RawMessage", serviceException.getRawMessage());
}
}
});
```

图片处理

OSS图片处理，是OSS对外提供的海量、安全、低成本、高可靠的图片处理服务。用户将原始图片上传保存到OSS，通过简单的 RESTful 接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。图片处理提供图片处理接口，图片上传请使用上传接口。基于OSS图片处理，用户可以搭建自己的图片处理服务。

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放、裁剪、旋转
- 图片效果
- 图片添加图片、文字、图文混合水印
- 自定义图片处理样式，在控制台的 图片处理 -> 样式管理 中定义
- 通过级联处理调用多个图片处理功能

SDK中使用时，只需要在下载图片时，调用request.setxOssProcess()方法设置处理参数。示例：

```
// 构造图片下载请求
GetObjectRequest get = new GetObjectRequest("<bucketName>", "example.jpg");

// 图片处理
request.setxOssProcess("image/resize,m_fixed,w_100,h_100");

OSSAsyncTask task = oss.asyncGetObject(get, new OSSCompletedCallback<GetObjectRequest, GetObjectResult>()
{
@Override
public void onSuccess(GetObjectRequest request, GetObjectResult result) {
// 请求成功
InputStream inputStream = result.getObjectContent();

byte[] buffer = new byte[2048];
int len;

try {
while ((len = inputStream.read(buffer)) != -1) {
// 处理下载的数据
}
} catch (IOException e) {
e.printStackTrace();
}
}

@Override
public void onFailure(GetObjectRequest request, ClientException clientExcepion, ServiceException serviceException)
```

```
{
// 处理异常
}
});
```

需要对图片进行其它处理，只要替换request.setxOssProcess()的参数就可以了。需要帮助，请参考：

- 可视化图片处理工具 ImageStyleViever ，可以直观的看到OSS图片处理的结果
- OSS图片处理的功能、使用演示 页面

指定范围下载

您可以在下载文件时指定一段范围，对于较大的Object适于使用此功能；如果在请求头中使用Range参数，则返回消息中会包含整个文件的长度和此次返回的范围。如：

```
GetObjectRequest get = new GetObjectRequest("<bucketName>", "<objectKey>");

// 设置范围
get.setRange(new Range(0, 99)); // 下载0到99字节共100个字节，文件范围从0开始计算
// get.setRange(new Range(100, Range.INFINITE)); // 下载从100个字节到结尾

OSSAsyncTask task = oss.asyncGetObject(get, new OSSCompletedCallback<GetObjectRequest, GetObjectResult>()
{
@Override
public void onSuccess(GetObjectRequest request, GetObjectResult result) {
// 请求成功
InputStream inputStream = result.getObjectContent();

byte[] buffer = new byte[2048];
int len;

try {
while ((len = inputStream.read(buffer)) != -1) {
// 处理下载的数据
}
} catch (IOException e) {
e.printStackTrace();
}
}

@Override
public void onFailure(GetObjectRequest request, ClientException clientExcepion, ServiceException serviceException)
{
// 请求异常
if (clientExcepion != null) {
// 本地异常如网络异常等
clientExcepion.printStackTrace();
}
if (serviceException != null) {
// 服务异常
Log.e("ErrorCode", serviceException.getErrorCode());
Log.e("RequestId", serviceException.getRequestId());
Log.e("HostId", serviceException.getHostId());
```

```
Log.e("RawMessage", serviceException.getRawMessage());
}
}
});
```

只获取文件元信息

通过headObject方法可以只获文件元信息而不获取文件的实体。代码如下：

```
// 创建同步获取文件元信息请求
HeadObjectRequest head = new HeadObjectRequest("<bucketName>", "<objectKey>");

OSSAsyncTask task = oss.asyncHeadObject(head, new OSSCompletedCallback<HeadObjectRequest,
HeadObjectResult>() {
@Override
public void onSuccess(HeadObjectRequest request, HeadObjectResult result) {
Log.d("headObject", "object Size: " + result.getMetadata().getContentLength());
Log.d("headObject", "object Content Type: " + result.getMetadata().getContentType());
}

@Override
public void onFailure(HeadObjectRequest request, ClientException clientExcepion, ServiceException
serviceException) {
// 请求异常
if (clientExcepion != null) {
// 本地异常如网络异常等
clientExcepion.printStackTrace();
}
if (serviceException != null) {
// 服务异常
Log.e("ErrorCode", serviceException.getErrorCode());
Log.e("RequestId", serviceException.getRequestId());
Log.e("HostId", serviceException.getHostId());
Log.e("RawMessage", serviceException.getRawMessage());
}
}
});

// task.waitUntilFinished();
```

数据完整性校验

因为移动端网络环境的复杂性，数据在客户端和服务器之间传输时有可能会出错。因此OSS SDK提供了基于CRC的端到端的数据完整性效验。

CRC校验在读取下载数据流的时候，如果开启了crc效验，会在数据流读取完毕后自动验证数据完整性。

```
GetObjectRequest request = new GetObjectRequest(OSSTestConfig.ANDROID_TEST_BUCKET, testFile);
//开启CRC效验
request.setCRC64(OSSRequest.CRC64Config.YES);
```

```
//....
try{
GetObjectResult result = oss.getObject(request);
InputStream in = result.getObjectContent() ;
ByteArrayOutputStream output = new ByteArrayOutputStream();
byte[] buffer = new byte[BUFFER_SIZE];
int len;
while ((len = in.read(buffer)) > -1) {
output.write(buffer, 0, len);
}
output.flush();
in.close();
}catch(ClientException e){
//...
}catch(InconsistentException e){
//....
}
```

授权访问

SDK支持签名出特定有效时长或者公开的URL，用于转给第三方实现授权访问。

签名私有资源的指定有效时长的访问URL

如果Bucket或Object不是公共可读的，那么需要调用以下接口，获得签名后的URL：

```
String url = oss.presignConstrainedObjectURL("<bucketName>", "<objectKey>", 30 * 60);
```

签名公开的访问URL

如果Bucket或Object是公共可读的，那么调用一下接口，获得可公开访问Object的URL：

```
String url = oss.presignPublicObjectURL("<bucketName>", "<objectKey>");
```

分片上传

下面演示通过分片上传文件的整个流程。

初始化分片上传

```
String uploadId;

InitiateMultipartUploadRequest init = new InitiateMultipartUploadRequest("<bucketName>", "<objectKey>");
InitiateMultipartUploadResult initResult = oss.initMultipartUpload(init);

uploadId = initResult.getUploadId();
```

- 我们用InitiateMultipartUploadRequest来指定上传文件的名字和所属存储空间（Bucket）。
- 在InitiateMultipartUploadRequest中，您也可以设置ObjectMeta，但是不必指定其中的ContentLength 。
- initiateMultipartUpload 的返回结果中含有UploadId ，它是区分分片上传事件的唯一标识，在后面的操作中，我们将用到它。

上传分片

```
long partSize = 128 * 1024; // 设置分片大小

int currentIndex = 1; // 上传分片编号，从1开始

File uploadFile = new File("<uploadFilePath>"); // 需要分片上传的文件

InputStream input = new FileInputStream(uploadFile);
long fileLength = uploadFile.length();

long uploadedLength = 0;
List<PartETag> partETags = new ArrayList<PartETag>(); // 保存分片上传的结果
while (uploadedLength < fileLength) {

int partLength = (int)Math.min(partSize, fileLength - uploadedLength);
byte[] partData = IOUtils.readStreamAsBytesArray(input, partLength); // 按照分片大小读取文件的一段内容

UploadPartRequest uploadPart = new UploadPartRequest("<bucketName>", "<objectKey>", uploadId,
currentIndex);
uploadPart.setPartContent(partData); // 设置分片内容
UploadPartResult uploadPartResult = oss.uploadPart(uploadPart);
partETags.add(new PartETag(currentIndex, uploadPartResult.getETag())); // 保存分片上传成功后的结果

uploadedLength += partLength;
currentIndex++;
}
```

上面程序的核心是调用UploadPart方法来上传每一个分片，但是要注意以下几点：

- UploadPart 方法要求除最后一个Part以外，其他的Part大小都要大于100KB。但是Upload Part接口并不会立即校验上传Part的大小（因为不知道是否为最后一块）；只有当Complete Multipart Upload的时候才会校验。
- OSS会将服务器端收到Part数据的MD5值放在ETag头内返回给用户。

- Part号码的范围是1~10000。如果超出这个范围，OSS将返回InvalidArgument的错误码。
- 每次上传part时都要把流定位到此次上传片开头所对应的位置。
- 每次上传part之后，OSS的返回结果会包含一个分片的ETag，您需要将它和块编号组合成PartEtag，保存在list中，后续完成分片上传需要用到。

MD5校验设置

因为移动端网络环境的复杂性，OSS提供了基于端到端的文件验证功能。如果要校验分片上传到OSS的文件和本地文件是否一致，可以在上传分片时携带分片的Content-MD5值，OSS服务器会帮助用户进行MD5校验，只有在OSS服务器接收到的分片MD5值和Content-MD5一致时才可以上传成功，从而保证上传分片的一致性。

```
UploadPartRequest uploadPart = new UploadPartRequest("<bucketName>", "<objectKey>", uploadId,
currentIndex);
uploadPart.setPartContent(partData); // 设置分片内容
//设置MD5内容
uploadPart.setMd5Digest(BinaryUtil.calculateBase64Md5(data));
```

完成分片上传

```
CompleteMultipartUploadRequest complete = new CompleteMultipartUploadRequest("<bucketName>",
"<objectKey>", uploadId, partETags);
CompleteMultipartUploadResult completeResult = oss.completeMultipartUpload(complete);

Log.d("multipartUpload", "multipart upload success! Location: " + completeResult.getLocation());
```

上面代码中的 partETags 就是进行分片上传中保存的partETag的列表，OSS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，OSS会将这些part组合成一个完整的文件。

完成分片上传（设置ServerCallback）

```
CompleteMultipartUploadRequest complete = new CompleteMultipartUploadRequest("<bucketName>",
"<objectKey>", uploadId, partETags);
CompleteMultipartUploadResult completeResult = oss.completeMultipartUpload(complete);
complete.setCallbackParam(new HashMap<String, String>() {
{
put("callbackUrl", "<server address>");
put("callbackBody", "<test>");
}
});
Log.d("multipartUploadWithServerCallback", completeResult.getServerCallbackReturnBody());
```

完成分片上传请求可以设置Server Callback参数，请求完成后会向指定的Server Adress发送回调请求。

删除分片上传事件

我们可以用 abortMultipartUpload 方法取消分片上传。

```
AbortMultipartUploadRequest abort = new AbortMultipartUploadRequest("<bucketName>", "<objectKey>",
uploadId);
oss.abortMultipartUpload(abort); // 若无异常抛出说明删除成功
```

罗列分片

我们可以用 listParts 方法获取某个上传事件所有已上传的分片。

```
ListPartsRequest listParts = new ListPartsRequest("<bucketName>", "<objectKey>", uploadId);

ListPartsResult result = oss.listParts(listParts);

for (int i = 0; i < result.getParts().size(); i++) {
Log.d("listParts", "partNum: " + result.getParts().get(i).getPartNumber());
Log.d("listParts", "partEtag: " + result.getParts().get(i).getETag());
Log.d("listParts", "lastModified: " + result.getParts().get(i).getLastModified());
Log.d("listParts", "partSize: " + result.getParts().get(i).getSize());
}
```

- 默认情况下，如果存储空间中的分片上传事件的数量大于1000，则只会返回1000个Multipart Upload信息，且返回结果中 IsTruncated 为false，并返回 NextPartNumberMarker作为下此读取的起点。
- 若想增大返回分片上传事件数目，可以修改 MaxParts 参数，或者使用 PartNumberMarker 参数分次读取。

管理文件

罗列Bucket所有文件

```
 ListObjectsRequest listObjects = new ListObjectsRequest("<bucketName>");

// 设定前缀
listObjects.setPrefix("file");

// 设置成功、失败回调，发送异步罗列请求
OSSAsyncTask task = oss.asyncListObjects(listObjects, new OSSCompletedCallback<ListObjectsRequest,
ListObjectsResult>() {
    @Override
```

```
    public void onSuccess(ListObjectsRequest request, ListObjectsResult result) {
        Log.d("AyncListObjects", "Success!");
        for (int i = 0; i < result.getObjectSummaries().size(); i++) {
            Log.d("AyncListObjects", "object: " + result.getObjectSummaries().get(i).getKey() + " "
                    + result.getObjectSummaries().get(i).getETag() + " "
                    + result.getObjectSummaries().get(i).getLastModified());
        }
    }

    @Override
    public void onFailure(ListObjectsRequest request, ClientException clientExcepion, ServiceException
serviceException) {
        // 请求异常
        if (clientExcepion != null) {
            // 本地异常如网络异常等
            clientExcepion.printStackTrace();
        }
        if (serviceException != null) {
            // 服务异常
            Log.e("ErrorCode", serviceException.getErrorCode());
            Log.e("RequestId", serviceException.getRequestId());
            Log.e("HostId", serviceException.getHostId());
            Log.e("RawMessage", serviceException.getRawMessage());
        }
    }
});
task.waitUntilFinished();
```

上述代码列出了bucket中以”file”为前缀的所有文件。具体可以设置的参数名称和作用如下：

名称	作用
delimiter	用于对Object名字进行分组的字符。所有名字包含指定的前缀且第一次出现delimiter字符之间的object作为一组元素: CommonPrefixes。
marker	设定结果从marker之后按字母排序的第一个开始返回。
maxkeys	限定此次返回object的最大数，如果不设定，默认为100，maxkeys取值不能大于1000。
prefix	限定返回的object key必须以prefix作为前缀。注意使用prefix查询时，返回的key中仍会包含prefix。

检查文件是否存在

SDK提供了方便的同步接口检测某个指定Object是否存在OSS上：

```
try {
    if (oss.doesObjectExist("<bucketName>", "<objectKey>")) {
        Log.d("doesObjectExist", "object exist.");
    } else {
```

```
            Log.d("doesObjectExist", "object does not exist.");
        }
    } catch (ClientException e) {
        // 本地异常如网络异常等
        e.printStackTrace();
    } catch (ServiceException e) {
        // 服务异常
        Log.e("ErrorCode", e.getErrorCode());
        Log.e("RequestId", e.getRequestId());
        Log.e("HostId", e.getHostId());
        Log.e("RawMessage", e.getRawMessage());
    }
```

复制文件

```
// 创建copy请求
CopyObjectRequest copyObjectRequest = new CopyObjectRequest("<srcBucketName>", "<srcObjectKey>",
        "<destBucketName>", "<destObjectKey>");

// 可选设置copy文件元信息
// ObjectMetadata objectMetadata = new ObjectMetadata();
// objectMetadata.setContentType("application/octet-stream");
// copyObjectRequest.setNewObjectMetadata(objectMetadata);

// 异步copy
OSSAsyncTask copyTask = oss.asyncCopyObject(copyObjectRequest, new
OSSCompletedCallback<CopyObjectRequest, CopyObjectResult>() {
    @Override
    public void onSuccess(CopyObjectRequest request, CopyObjectResult result) {
            Log.d("copyObject", "copy success!");
    }

    @Override
    public void onFailure(CopyObjectRequest request, ClientException clientExcepion, ServiceException
serviceException) {
        // 请求异常
        if (clientExcepion != null) {
            // 本地异常如网络异常等
            clientExcepion.printStackTrace();
        }
        if (serviceException != null) {
            // 服务异常
            Log.e("ErrorCode", serviceException.getErrorCode());
            Log.e("RequestId", serviceException.getRequestId());
            Log.e("HostId", serviceException.getHostId());
            Log.e("RawMessage", serviceException.getRawMessage());
        }
    }
});
```

上述代码实现了CopyObject，注意：

- 源Object和目标Object必须属于同一个数据中心。

- 如果拷贝操作的源Object地址和目标Object地址相同，可以修改已有Object的meta信息。
- 拷贝文件大小不能超过1G，超过1G需使用Multipart Upload操作。

删除文件

```
// 创建删除请求
DeleteObjectRequest delete = new DeleteObjectRequest("<bucketName>", "<objectKey>");
// 异步删除
OSSAsyncTask deleteTask = oss.asyncDeleteObject(delete, new OSSCompletedCallback<DeleteObjectRequest,
DeleteObjectResult>() {
    @Override
    public void onSuccess(DeleteObjectRequest request, DeleteObjectResult result) {
        Log.d("asyncCopyAndDelObject", "success!");
    }

    @Override
    public void onFailure(DeleteObjectRequest request, ClientException clientExcepion, ServiceException
serviceException) {
        // 请求异常
        if (clientExcepion != null) {
            // 本地异常如网络异常等
            clientExcepion.printStackTrace();
        }
        if (serviceException != null) {
            // 服务异常
            Log.e("ErrorCode", serviceException.getErrorCode());
            Log.e("RequestId", serviceException.getRequestId());
            Log.e("HostId", serviceException.getHostId());
            Log.e("RawMessage", serviceException.getRawMessage());
        }
    }

});
```

上述代码实现了删除Object，注意：

- DeleteObject要求对所在的Bucket有写权限。

只获取文件的元信息

下面的代码用于获取文件的元信息：

```
// 创建同步获取文件元信息请求
HeadObjectRequest head = new HeadObjectRequest("<bucketName>", "<objectKey>");

OSSAsyncTask task = oss.asyncHeadObject(head, new OSSCompletedCallback<HeadObjectRequest,
HeadObjectResult>() {
    @Override
    public void onSuccess(HeadObjectRequest request, HeadObjectResult result) {
        Log.d("headObject", "object Size: " + result.getMetadata().getContentLength()); // 获取文件长度
        Log.d("headObject", "object Content Type: " + result.getMetadata().getContentType()); // 获取文件类型
```

```
    }

    @Override
    public void onFailure(HeadObjectRequest request, ClientException clientExcepion, ServiceException
serviceException) {
        // 请求异常
        if (clientExcepion != null) {
            // 本地异常如网络异常等
            clientExcepion.printStackTrace();
        }
        if (serviceException != null) {
            // 服务异常
            Log.e("ErrorCode", serviceException.getErrorCode());
            Log.e("RequestId", serviceException.getRequestId());
            Log.e("HostId", serviceException.getHostId());
            Log.e("RawMessage", serviceException.getRawMessage());
        }
    }
});

// task.waitUntilFinished();
```

管理Bucket

Bucket 是 OSS 上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体；Bucket 名称在整个 OSS 服务中具有全局唯一性，且不能修改；存储在 OSS 上的每个 Object 必须都包含在某个 Bucket 中。

新建Bucket

以下代码可以新建一个Bucket:

```
CreateBucketRequest createBucketRequest = new CreateBucketRequest("<bucketName>");
createBucketRequest.setBucketACL(CannedAccessControlList.PublicRead); // 指定Bucket的ACL权限
createBucketRequest.setLocationConstraint("oss-cn-hangzhou"); // 指定Bucket所在的数据中心
OSSAsyncTask createTask = oss.asyncCreateBucket(createBucketRequest, new
OSSCompletedCallback<CreateBucketRequest, CreateBucketResult>() {
@Override
public void onSuccess(CreateBucketRequest request, CreateBucketResult result) {
Log.d("locationConstraint", request.getLocationConstraint());
}

@Override
public void onFailure(CreateBucketRequest request, ClientException clientException, ServiceException
serviceException) {
// 请求异常
if (clientException != null) {
```

```
// 本地异常如网络异常等
clientException.printStackTrace();
}
if (serviceException != null) {
// 服务异常
Log.e("ErrorCode", serviceException.getErrorCode());
Log.e("RequestId", serviceException.getRequestId());
Log.e("HostId", serviceException.getHostId());
Log.e("RawMessage", serviceException.getRawMessage());
}
}
});
```

上述代码在创建bucket时，指定了Bucket的ACL和所在的数据中心。

- 每个用户的Bucket数量不能超过30个。
- 每个Bucket的名字全局唯一，也就是说创建的Bucket不能和其他用户已经在使用的Bucket同名，否则会创建失败。
- 创建的时候可以选择Bucket ACL权限，如果不设置ACL，默认是private。
- 创建成功结果返回Bucket所在数据中心。

获取Bucket ACL权限

以下代码可以获取Bucket ACL：

```
GetBucketACLRequest getBucketACLRequest = new GetBucketACLRequest("<bucketName>");
OSSAsyncTask getBucketAclTask = oss.asyncGetBucketACL(getBucketACLRequest, new
OSSCompletedCallback<GetBucketACLRequest, GetBucketACLResult>() {
@Override
public void onSuccess(GetBucketACLRequest request, GetBucketACLResult result) {
Log.d("BucketAcl", result.getBucketACL());
Log.d("Owner", result.getBucketOwner());
Log.d("ID", result.getBucketOwnerID());
}

@Override
public void onFailure(GetBucketACLRequest request, ClientException clientException, ServiceException
serviceException) {
// 请求异常
if (clientException != null) {
// 本地异常如网络异常等
clientException.printStackTrace();
}
if (serviceException != null) {
// 服务异常
Log.e("ErrorCode", serviceException.getErrorCode());
Log.e("RequestId", serviceException.getRequestId());
Log.e("HostId", serviceException.getHostId());
Log.e("RawMessage", serviceException.getRawMessage());
}
}
});
```

上述代码在获取Bucket的ACL权限。

- 目前Bucket有三种访问权限：public-read-write，public-read和private。
- 只有Bucket的拥有者才能使用Get Bucket ACL这个接口。
- 获取的结果中返回Bucket拥有者ID、拥有者名称（和ID保持一致）和权限。

删除Bucket

以下代码删除了一个Bucket：

```
DeleteBucketRequest deleteBucketRequest = new DeleteBucketRequest("<bucketName>");
OSSAsyncTask deleteBucketTask = oss.asyncDeleteBucket(deleteBucketRequest, new
OSSCompletedCallback<DeleteBucketRequest, DeleteBucketResult>() {
@Override
public void onSuccess(DeleteBucketRequest request, DeleteBucketResult result) {
Log.d("DeleteBucket", "Success!");
}

@Override
public void onFailure(DeleteBucketRequest request, ClientException clientException, ServiceException
serviceException) {
// 请求异常
if (clientException != null) {
// 本地异常如网络异常等
clientException.printStackTrace();
}
if (serviceException != null) {
// 服务异常
Log.e("ErrorCode", serviceException.getErrorCode());
Log.e("RequestId", serviceException.getRequestId());
Log.e("HostId", serviceException.getHostId());
Log.e("RawMessage", serviceException.getRawMessage());
}
});
```

注：

- 为了防止误删除的发生，OSS不允许用户删除一个非空的Bucket。
- 只有Bucket的拥有者才能删除这个Bucket。

异常响应

OSS Android SDK 中有两种异常 ClientException 以及 ServiceException ，它们都是受检异常。

ClientException

ClientException指SDK内部出现的异常，比如参数错误，网络无法到达，主动取消等等。

ServiceException

OSSException指服务器端错误，它来自于对服务器错误信息的解析。OSSException一般有以下几个成员：

- Code： OSS返回给用户的错误码。
- Message： OSS给出的详细错误信息。
- RequestId： 用于唯一标识该次请求的UUID。当您无法解决问题时，可以凭这个RequestId来请求OSS开发工程师的帮助。
- HostId： 用于标识访问的OSS集群。
- rawMessage： HTTP响应的原始Body文本。

下面是OSS中常见的异常：

错误码	描述
AccessDenied	拒绝访问
BucketAlreadyExists	Bucket已经存在
BucketNotEmpty	Bucket不为空
EntityTooLarge	实体过大
EntityTooSmall	实体过小
FileGroupTooLarge	文件组过大
FilePartNotExist	文件Part不存在
FilePartStale	文件Part过时
InvalidArgument	参数格式错误
InvalidAccessKeyId	AccessKeyId不存在
InvalidBucketName	无效的Bucket名字
InvalidDigest	无效的摘要
InvalidObjectName	无效的Object名字
InvalidPart	无效的Part
InvalidPartOrder	无效的part顺序
InvalidTargetBucketForLogging	Logging操作中有无效的目标bucket
InternalError	OSS内部发生错误
MalformedXML	XML格式非法
MethodNotAllowed	不支持的方法
MissingArgument	缺少参数
MissingContentLength	缺少内容长度

NoSuchBucket	Bucket不存在
NoSuchKey	文件不存在
NoSuchUpload	Multipart Upload ID不存在
NotImplemented	无法处理的方法
PreconditionFailed	预处理错误
RequestTimeTooSkewed	发起请求的时间和服务器时间超出15分钟
RequestTimeout	请求超时
SignatureDoesNotMatch	签名错误
TooManyBuckets	用户的Bucket数目超过限制

iOS-SDK

前言

简介

本文档主要介绍OSS iOS SDK的安装和使用。本文档假设您已经开通了阿里云OSS 服务，并创建了AccessKeyId 和AccessKeySecret。文中的ID 指的是AccessKeyId，KEY 指的是AccessKeySecret。如果您还没有开通或者还不了解OSS，请登录OSS产品主页获取更多的帮助。

环境要求

- iOS系统版本：iOS 8.0及以上
- 必须注册有Aliyun.com用户账户，并开通OSS服务。

SDK下载

- github地址：https://github.com/aliyun/aliyun-oss-ios-sdk
- pod依赖：`pod ‘AliyunOSSiOS’
- demo地址：https://github.com/aliyun/aliyun-oss-ios-sdk
- SDK API文档: APIDOC

- 版本迭代： 请参考链接

安装

直接引入Framework

如何获取OSS iOS SDK framework参见链接。

在Xcode中，直接把framework拖入您对应的Target下即可，在弹出框勾选Copy items if needed。

Pod依赖

如果工程是通过pod管理依赖，那么在Podfile中加入以下依赖即可，不需要再导入framework：

```
pod 'AliyunOSSiOS'
```

Cocoapods是一个非常优秀的依赖管理工具，推荐参考文档: CocoaPods安装和使用教程。

直接引入Framework和Pod依赖，两种方式选其一即可。

工程中引入头文件

```
#import <AliyunOSSiOS/OSSService.h>
```

注意，引入Framework后，需要在工程Build Settings的Other Linker Flags中加入-ObjC。如果工程此前已经设置过-force_load选项，那么，需要加入-force_load <framework path>/AliyunOSSiOS。

兼容IPv6-Only网络

OSS移动端SDK为了解决无线网络下域名解析容易遭到劫持的问题，已经引入了HTTPDNS进行域名解析，直接使用IP请求OSS服务端。在IPv6-Only的网络下，可能会遇到兼容性问题。而APP官方近期发布了关于IPv6-only网络环境兼容的APP审核要求，为此，SDK从2.5.0版本开始已经做了兼容性处理。在新版本中，除了-ObjC的设置，还需要引入两个系统库：

```
libresolv.tbd
CoreTelephony.framework
SystemConfiguration.framework
```

关于苹果ATS政策

WWDC 2016开发者大会上，苹果宣布从2017年1月1日起，苹果App Store中的所有App都必须启用 App Transport Security(ATS) 安全功能。也就是说，所有的新提交 app 默认是不允许使用NSAllowsArbitraryLoads来绕过 ATS 限制的。我们最好保证 app 的所有网络请求都是 HTTPS 加密的，否则可能会在应用审核时遇到麻烦。

本SDK在2.6.0以上版本中对此做出支持，其中，SDK不会自行发出任何非HTTPS请求，同时，SDK支持https://前缀的Endpoint，只需要设置正确的HTTPS Endpoint，就能保证发出的网络请求都是符合要求的。

所以，用户需要注意：

- 设置Endpoint时，需要使用https://前缀的URL。
- 在实现加签、获取STSToken等回调时，需要确保自己不会发出 非HTTPS 的请求。

初始化

OSSClient是OSS服务的iOS客户端，它为调用者提供了一系列的方法，可以用来操作，管理存储空间（bucket）和文件（object）等。在使用SDK发起对OSS的请求前，您需要初始化一个OSSClient实例，并对它进行一些必要设置。

OSSClient的生命周期和应用程序的生命周期保持一致即可。在应用程序启动时创建一个ossClient，在应用程序结束时销毁即可。

确定Endpoint

Endpoint是阿里云OSS服务在各个区域的地址，目前支持两种形式

Endpoint类型	解释
OSS区域地址	使用OSS Bucket所在区域地址，各个区域Endpoint参考这里
用户自定义域名	用户自定义域名，且CNAME指向OSS域名

关于Endpoint，可以参考：点击查看。

OSS区域地址

使用OSS Bucket所在区域地址，Endpoint查询可以有下面两种方式：

- 查询Endpoint与区域对应关系详情，可以参考：点击查看。
- 您可以登录 阿里云OSS控制台，进入Bucket概览页，Bucket域名的后缀部分：如bucket-1.oss-cn-hangzhou.aliyuncs.com的oss-cn-hangzhou.aliyuncs.com部分为该Bucket的外网Endpoint。

Cname

您可以将自己拥有的域名通过Cname绑定到某个存储空间（bucket）上，然后通过自己域名访问存储空间内的文件。

比如您要将域名new-image.xxxxx.com绑定到深圳区域的名称为image的存储空间上：您需要到您的域名xxxxx.com托管商那里设定一个新的域名解析，将http://new-image.xxxxx.com 解析到 http://image.oss-cn-shenzhen.aliyuncs.com ，类型为CNAME。

设置EndPoint和凭证

移动终端是一个不受信任的环境，把AccessKeyId和AccessKeySecret直接保存在终端用来加签请求，**存在极高的风险**。建议只在测试时使用明文设置模式，业务应用推荐使用***STS鉴权模式***或***自签名模式***，详细请参考：访问控制、移动端直传。

如果用STS鉴权模式，推荐使用OSSAuthCredentialProvider方式直接访问鉴权应用服务器，token过期后可以自动更新。

更多信息可查看可查看sample 点击查看。

设置EndPoint和CredentialProvider示例如下：

```
NSString *endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

//直接访问鉴权服务器（推荐，token过期后可以自动更新）
id<OSSCredentialProvider> credential = [[OSSAuthCredentialProvider alloc] initWithAuthServerUrl:@"鉴权服务器地址，例如http://abc.com"];
client = [[OSSClient alloc] initWithEndpoint:endPoint credentialProvider:credential];
```

提示：

- 更多鉴权方式参考：访问控制。

设置EndPoint为cname

如果您已经在bucket上绑定cname，将该cname直接设置到endPoint即可。如：

```
NSString *endpoint = "http://new-image.xxxxx.com";
...
client = [[OSSClient alloc] initWithEndpoint:endpoint credentialProvider:credential];
```

注意，苹果要求支持ATS标准后，所有Endpoint URL都必须为HTTPS URL，而cname域名暂不支持证书设置，所以暂时不能用cname设置Endpoint。

更多鉴权方式参考：访问控制

启用日志

移动端的使用环境比较复杂。会出现部分区域或某一个时段无法正常使用oss sdk的情况。为了进一步方便开发

者定位问题，oss sdk 在开启日志记录功能后，会将一些log信息记录在本地。如需开启需要在OSSClient使用之前进行初始化，调用如下方法即可。

补充说明：

- 文件存储位置在沙盒的Caches/OSSLogs文件夹内。
- 使用者可以自行选择将文件上传至服务器，进一步追踪问题。
- 也可以接入阿里云SLS日志服务进行日志文件上传, 日志服务传送门。

```
//日志的样式
//2017/10/25 11:05:43:863 [Debug]: 第17次：<NSThread: 0x7f8099108580>{number = 3, name = (null)}
//2017/10/25 11:05:43:863 [Debug]: 第15次：<NSThread: 0x7f80976052c0>
//2017/10/25 11:05:43:863 [Debug]: ----------TestDebug------------
[OSSLog enableLog];//执行该方法，开启日志记录
```

设置网络参数

也可以在初始化的时候设置详细的ClientConfiguration：

```
NSString *endpoint = "https://oss-cn-hangzhou.aliyuncs.com";

//直接访问鉴权服务器（推荐，token过期后可以自动更新）
id<OSSCredentialProvider> credential = [[OSSAuthCredentialProvider alloc] initWithAuthServerUrl:@"鉴权服务器地址，例如http://abc.com"];
client = [[OSSClient alloc] initWithEndpoint:endPoint credentialProvider:credential];

OSSClientConfiguration * conf = [OSSClientConfiguration new];
conf.maxRetryCount = 3; // 网络请求遇到异常失败后的重试次数
conf.timeoutIntervalForRequest = 30; // 网络请求的超时时间
conf.timeoutIntervalForResource = 24 * 60 * 60; // 允许资源传输的最长时间

client = [[OSSClient alloc] initWithEndpoint:endpoint credentialProvider:credential clientConfiguration:conf];
```

对于OSSTask的一些说明

所有调用api的操作，都会立即获得一个OSSTask，如：

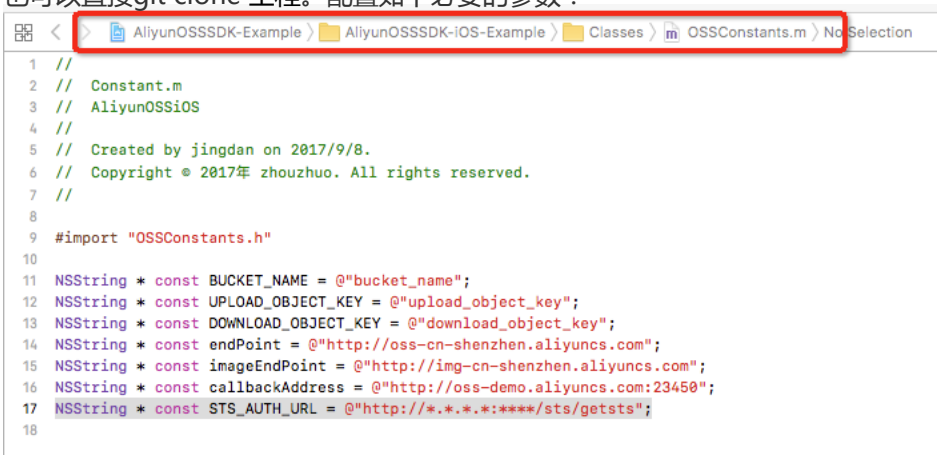
```
OSSTask * task = [client getObject:get];
```

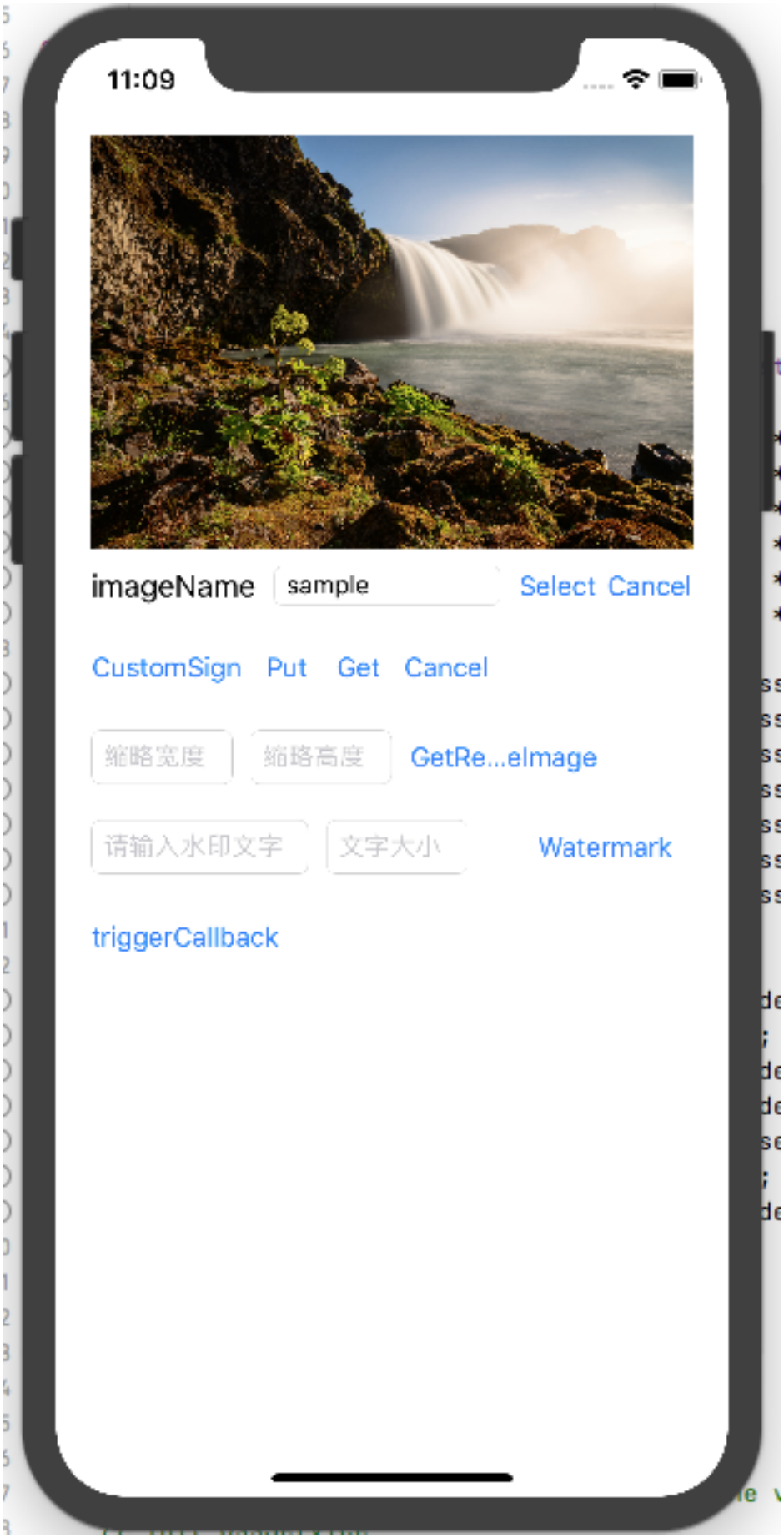
可以为这个Task设置一个延续(continution)，以实现异步回调，如：

```
[task continueWithBlock: ^(OSSTask *task) {
// do something
...

return nil;
```

```
}];
```

也可以等待这个Task完成，以实现同步等待，如：

```objc[task waitUntilFinished];

...

# 快速入门

以下演示了上传、下载文件的基本流程。更多细节用法可以参考本工程的：

ios demo示例: 点击查看

mac demo示例: 点击查看

swift demo示例: 点击查看

测试用例(可以参看api使用方式)：点击查看

也可以直接git clone 工程。配置如下必要的参数：




然后运行工程，demo如下：
```


11:09
imageName
sample
Select
Cancel
CustomSign
Put
Get
Cancel
缩略宽度
缩略高度
GetRe...eImage
请输入水印文字
文字大小
Watermark
triggerCallback

示例

IOS平台

需要添加的引用

```
#import <AliyunOSSiOS/OSSService.h>
```

STEP-1. 初始化OSSClient

初始化主要完成Endpoint设置、鉴权方式设置、Client参数设置。其中，鉴权方式包含明文设置模式、自签名模式、STS鉴权模式。如果要使用STS鉴权请先阅读访问控制章节了解RAM的基础知识。以下内容假设您已开通RAM服务并了解RAM相关内容。了解如何获取子账户AccessKeyId，SecretKeyId以及RoleArn信息。

完善脚本文件中AccessKeyId，SecretKeyId以及RoleArn参数信息。通过python可以启动一个本机http服务。在客户端代码中访问本地服务从而获得StsToken.AccessKeyId,StsToken.SecretKeyId以及StsToken.SecurityToken。

更多信息可查看sample中STS使用说明点击查看。

```
NSString *endpoint = "https://oss-cn-hangzhou.aliyuncs.com";

// 移动端建议使用STS方式初始化OSSClient。可以通过sample中STS使用说明了解更多(https://github.com/aliyun/aliyun-oss-ios-sdk/tree/master/DemoByOC)
id<OSSCredentialProvider> credential = [[OSSStsTokenCredentialProvider alloc]
initWithAccessKeyId:@"AccessKeyId" secretKeyId:@"AccessKeySecret" securityToken:@"SecurityToken"];

client = [[OSSClient alloc] initWithEndpoint:endpoint credentialProvider:credential];
```

通过OSSClient发起上传、下载请求是线程安全的，您可以并发执行多个任务。

STEP-2. 上传文件

这里假设您已经在控制台上拥有自己的Bucket。SDK的所有操作，都会返回一个OSSTask，您可以为这个task设置一个延续动作，等待其异步完成，也可以通过调用waitUntilFinished阻塞等待其完成。

```
OSSPutObjectRequest * put = [OSSPutObjectRequest new];

put.bucketName = @"<bucketName>";
put.objectKey = @"<objectKey>";

put.uploadingData = <NSData *>; // 直接上传NSData

put.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpectedToSend) {
NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};
```

```
OSSTask * putTask = [client putObject:put];

[putTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
NSLog(@"upload object success!");
} else {
NSLog(@"upload object failed, error: %@" , task.error);
}
return nil;
}];

// 可以等待任务完成
// [putTask waitUntilFinished];
```

STEP-3. 下载指定文件

下载一个指定object为NSData:

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
request.bucketName = @"<bucketName>";
request.objectKey = @"<objectKey>";

request.downloadProgress = ^(int64_t bytesWritten, int64_t totalBytesWritten, int64_t totalBytesExpectedToWrite) {
NSLog(@"%lld, %lld, %lld", bytesWritten, totalBytesWritten, totalBytesExpectedToWrite);
};

OSSTask * getTask = [client getObject:request];

[getTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
NSLog(@"download object success!");
OSSGetObjectResult * getResult = task.result;
NSLog(@"download result: %@", getResult.downloadedData);
} else {
NSLog(@"download object failed, error: %@" ,task.error);
}
return nil;
}];

// 如果需要阻塞等待任务完成
// [task waitUntilFinished];
```

MAC平台

MAC平台和ios平台示例一致，但是引入方式不一样。

```
import <AliyunOSSOSX/AliyunOSSiOS.h>
```

访问控制

移动终端是一个不受信任的环境。为此，SDK提供了两种鉴权模式：STS鉴权模式和自签名模式 。

STS鉴权模式

介绍

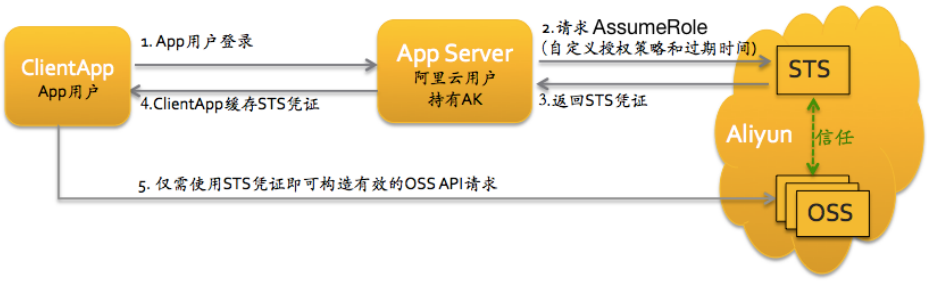
OSS可以通过阿里云STS服务，进行临时授权访问。阿里云STS (Security Token Service) 是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或联邦用户（用户身份由您自己管理）颁发一个自定义时效和权限的访问凭证，App端称为FederationToken。第三方应用或联邦用户可以使用该访问凭证直接调用阿里云产品API，或者使用阿里云产品提供的SDK来访问云产品API。

使用STS的好处：

您无需透露您的长期密钥(AccessKey)给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用即可。这个令牌的访问权限及有效期限都可以由您自定义。

您无需关心权限撤销问题，访问令牌过期后就自动失效。

以APP应用为例，交互流程如下图：



使用STS详细步骤

App用户登录。

App用户由您自己管理。您可以自定义身份管理系统，也可以使用外部Web账号或OpenID。对于每个有效的App用户来说，AppServer是可以确切地定义出每个App用户的最小访问权限。

AppServer请求STS服务获取一个安全令牌(SecurityToken)。

在调用STS之前，AppServer需要确定App用户的最小访问权限（用Policy语法描述）以及授权的过期时间。

通过调用STS的AssumeRole(扮演角色)接口来获取安全令牌。角色管理与使用相关内容请参考RAM使用指南中的角色管理。

STS返回给AppServer一个有效的访问凭证，包括一个安全令牌(SecurityToken)、临时访问密钥(AccessKeyId, AccessKeySecret)以及过期时间。

AppServer将访问凭证返回给ClientApp。

ClientApp可以缓存这个凭证。当凭证失效时，ClientApp需要向AppServer申请新的有效访问凭证。比如，访问凭证有效期为1小时，那么ClientApp可以每30分钟向AppServer请求更新访问凭证。

ClientApp使用本地缓存的访问凭证去请求Aliyun Service API。云服务会感知STS访问凭证，并会依赖STS服务来验证访问凭证，并正确响应用户请求。

注意：使用这种模式授权需要先开通阿里云RAM服务：**单击查看**

STS安全令牌详情请参考RAM使用指南中的**角色管理**

调用STS服务接口**AssumeRole**来获取有效访问凭证即可

直接使用STS SDK来调用该方法**单击查看**

STS使用手册：**单击查看**

OSS授权策略配置：**单击查看**

实战指南：**单击查看**

直接设置STSToken

您可以在APP或其他移动终端中，预先通过某种方式(如通过网络请求从您的业务Server上)获取一对STSToken，然后用它来初始化SDK。采取这种使用方式，您需要格外关注STSToken的过期时间，在STSToken即将过期时，需要您主动更新新的STSToken到SDK中。

初始化代码为：

```
id<OSSCredentialProvider> credential = [[OSSStsTokenCredentialProvider alloc]
initWithAccessKeyId:@"<StsToken.AccessKeyId>" secretKeyId:@"<StsToken.SecretKeyId>"
securityToken:@"<StsToken.SecurityToken>"];

client = [[OSSClient alloc] initWithEndpoint:endpoint credentialProvider:credential];
```

在您判断到Token即将过期时，您可以重新构造新的OSSClient，也可以通过如下方式更新CredentialProvider:

```
client.credentialProvider = [[OSSStsTokenCredentialProvider alloc]
initWithAccessKeyId:@"<StsToken.AccessKeyId>" secretKeyId:@"<StsToken.SecretKeyId>"
securityToken:@"<StsToken.SecurityToken>"];
```

实现获取STSToken回调

如果您期望SDK能自动帮您管理Token的更新，您需要告诉SDK如何获取Token。在SDK的应用中，您需要实现一个回调，这个回调通过您实现的方式去获取一个Federation Token(即STSToken)，然后返回。SDK会利用这个Token来进行加签处理，并在需要更新时主动调用这个回调获取Token，如图示：

```
id<OSSCredentialProvider> credential = [[OSSFederationCredentialProvider alloc]
initWithFederationTokenGetter:^OSSFederationToken * {
// 您需要在这里实现获取一个FederationToken，并构造成OSSFederationToken对象返回
// 如果因为某种原因获取失败，可直接返回nil
OSSFederationToken * token;
// 下面是一些获取token的代码，比如从您的server获取
...
return token;
}];
client = [[OSSClient alloc] initWithEndpoint:endpoint credentialProvider:credential];
```

此外，如果您已经通过别的方式拿到token所需的各个字段，也可以在这个回调中直接返回。如果这么做的话，您需要自己处理token的更新，更新后重新设置该OSSClient实例的OSSCredentialProvider。

使用示例：

假设您搭建的server地址为: http://localhost:8080/distribute-token.json，并假设访问这个地址，返回的数据如下：

```
{
"StatusCode": 200,
"AccessKeyId":"STS.iA645eTOXEqP3cg3VeHf",
"AccessKeySecret":"rV3VQrpFQ4BsyHSAvi5NVLpPIVffDJv4LojUBZCf",
"Expiration":"2015-11-03T09:52:59Z",
"SecurityToken":"CAES7QIIARKAAZPlqaN9ILiQZPS+JDkS/GSZN45RLx4YS/p3OgaUC+oJl3XSlbJ7StKpQ...."}
```

那么，您可以这么实现一个OSSFederationCredentialProvider实例：

```
id<OSSCredentialProvider> credential2 = [[OSSFederationCredentialProvider alloc]
```

```
initWithFederationTokenGetter:^OSSFederationToken * {
// 构造请求访问您的业务server
NSURL * url = [NSURL URLWithString:@"http://localhost:8080/distribute-token.json"];
NSURLRequest * request = [NSURLRequest requestWithURL:url];
OSSTaskCompletionSource * tcs = [OSSTaskCompletionSource taskCompletionSource];
NSURLSession * session = [NSURLSession sharedSession];

// 发送请求
NSURLSessionTask * sessionTask = [session dataTaskWithRequest:request
completionHandler:^(NSData *data, NSURLResponse *response, NSError *error) {
if (error) {
[tcs setError:error];
return;
}
[tcs setResult:data];
}];
[sessionTask resume];
// 需要阻塞等待请求返回
[tcs.task waitUntilFinished];
// 解析结果
if (tcs.task.error) {
NSLog(@"get token error: %@", tcs.task.error);
return nil;
} else {
// 返回数据是json格式，需要解析得到token的各个字段
NSDictionary * object = [NSJSONSerialization JSONObjectWithData:tcs.task.result
options:kNilOptions
error:nil];
OSSFederationToken * token = [OSSFederationToken new];
token.tAccessKey = [object objectForKey:@"AccessKeyId"];
token.tSecretKey = [object objectForKey:@"AccessKeySecret"];
token.tToken = [object objectForKey:@"SecurityToken"];
token.expirationTimeInGMTFormat = [object objectForKey:@"Expiration"];
NSLog(@"get token: %@", token);
return token;
}
}];
```

自签名模式

```
id<OSSCredentialProvider> credential = [[OSSCustomSignerCredentialProvider alloc]
initWithImplementedSigner:^NSString *(NSString *contentToSign, NSError *__autoreleasing *error) {
// 您需要在这里依照OSS规定的签名算法，实现加签一串字符内容，并把得到的签名传拼接上AccessKeyId后返回
// 一般实现是，将字符内容post到您的业务服务器，然后返回签名
// 如果因为某种原因加签失败，描述error信息后，返回nil
NSString *signature = [OSSUtil calBase64Sha1WithData:contentToSign withSecret:@"<your accessKeySecret>"]; //
这里是用SDK内的工具函数进行本地加签，建议您通过业务server实现远程加签
if (signature != nil) {
*error = nil;
} else {
*error = [NSError errorWithDomain:@"<your domain>" code:-1001 userInfo:@"<your error info>"];
return nil;
}
return [NSString stringWithFormat:@"OSS %@:%@", @"<your accessKeyId>", signature];
```

```
}];

client = [[OSSClient alloc] initWithEndpoint:endpoint credentialProvider:credential];
```

注意： 无论是STS鉴权模式，还是自签名模式，您实现的回调函数，都需要保证调用时返回结果。所以，如果您在其中实现了向业务server获取token、signature的网络请求，建议调用网络库的同步接口，或进行异步到同步的转换。回调都是在SDK具体请求的时候，在请求的子线程中执行，所以不用担心阻塞主线程。

上传文件

OSS 移动端SDK 上传文件的方式可以分为：简单上传，追加上传，分片上传和断点续传。

简单上传

上传Object可以直接上传OSSData，或者通过NSURL上传一个文件：

```
OSSPutObjectRequest * put = [OSSPutObjectRequest new];

// 必填字段
put.bucketName = @"<bucketName>";
put.objectKey = @"<objectKey>";

put.uploadingFileURL = [NSURL fileURLWithPath:@"<filepath>"];
// put.uploadingData = <NSData *>; // 直接上传NSData

// 可选字段，可不设置
put.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpectedToSend) {
// 当前上传段长度、当前已经上传总长度、一共需要上传的总长度
NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};

// 以下可选字段的含义参考： https://docs.aliyun.com/#/pub/oss/api-reference/object&PutObject
// put.contentType = @"";
// put.contentMd5 = @"";
// put.contentEncoding = @"";
// put.contentDisposition = @"";

// put.objectMeta = [NSMutableDictionary dictionaryWithObjectsAndKeys:@"value1", @"x-oss-meta-name1", nil];
// 可以在上传时设置元信息或者其他HTTP头部

OSSTask * putTask = [client putObject:put];

[putTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
```

```
NSLog(@"upload object success!");
} else {
NSLog(@"upload object failed, error: %@" , task.error);
}
return nil;
}];

// [putTask waitUntilFinished];

// [put cancel];
```

上传到文件目录

OSS服务是没有文件夹这个概念的，所有元素都是以文件来存储，但给用户提供了创建模拟文件夹的方式。创建模拟文件夹本质上来说是创建了一个名字以“/”结尾的文件，对于这个文件照样可以上传下载,只是控制台会对以“/”结尾的文件以文件夹的方式展示。

如，在上传文件是，如果把ObjectKey写为"folder/subfolder/file"，即是模拟了把文件上传到folder/subfolder/下的file文件。注意，路径默认是”根目录” ，不需要以’ / ‘开头。

上传时设置Content-Type和开启校验MD5

上传时可以显式指定ContentType，如果没有指定，SDK会根据文件名或者上传的ObjectKey自动判断。另外，上传Object时如果设置了Content-Md5，那么OSS会用之检查消息内容是否与发送时一致。SDK提供了方便的Base64和MD5计算方法。

```
OSSPutObjectRequest * put = [OSSPutObjectRequest new];

// 必填字段
put.bucketName = @"<bucketName>";
put.objectKey = @"<objectKey>";

put.uploadingFileURL = [NSURL fileURLWithPath:@"<filepath>"];
// put.uploadingData = <NSData *>; // 直接上传NSData

// 设置Content-Type，可选
put.contentType = @"application/octet-stream";

// 设置MD5校验，可选
put.contentMd5 = [OSSUtil base64Md5ForFilePath:@"<filePath>"]; // 如果是文件路径
// put.contentMd5 = [OSSUtil base64Md5ForData:<NSData *>]; // 如果是二进制数据

// 进度设置，可选
put.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpectedToSend) {
// 当前上传段长度、当前已经上传总长度、一共需要上传的总长度
NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};

OSSTask * putTask = [client putObject:put];
```

```
[putTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
NSLog(@"upload object success!");
} else {
NSLog(@"upload object failed, error: %@" , task.error);
}
return nil;
}];

// [putTask waitUntilFinished];

// [put cancel];
```

追加上传

Append Object以追加写的方式上传文件。通过Append Object操作创建的Object类型为Appendable Object，而通过Put Object上传的Object是Normal Object。

```
OSSAppendObjectRequest * append = [OSSAppendObjectRequest new];

// 必填字段
append.bucketName = @"<bucketName>";
append.objectKey = @"<objectKey>";
append.appendPosition = 0; // 指定从何处进行追加
NSString * docDir = [self getDocumentDirectory];
append.uploadingFileURL = [NSURL fileURLWithPath:@"<filepath>"];

// 可选字段
append.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpectedToSend) {
NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};

// 以下可选字段的含义参考：https://docs.aliyun.com/#/pub/oss/api-reference/object&AppendObject
// append.contentType = @"";
// append.contentMd5 = @"";
// append.contentEncoding = @"";
// append.contentDisposition = @"";

OSSTask * appendTask = [client appendObject:append];

[appendTask continueWithBlock:^id(OSSTask *task) {
NSLog(@"objectKey: %@", append.objectKey);
if (!task.error) {
NSLog(@"append object success!");
OSSAppendObjectResult * result = task.result;
NSString * etag = result.eTag;
long nextPosition = result.xOssNextAppendPosition;
} else {
NSLog(@"append object failed, error: %@" , task.error);
}
return nil;
}];
```

上传后回调通知

客户端在上传Object时可以指定OSS服务端在处理完上传请求后，通知您的业务服务器，在该服务器确认接收了该回调后将回调的结果返回给客户端。因为加入了回调请求和响应的过程，相比简单上传，使用回调通知机制一般会导致客户端花费更多的等待时间。

具体说明参考：Callback

代码示例：

```
OSSPutObjectRequest * request = [OSSPutObjectRequest new];
request.bucketName = @"<bucketName>";
request.objectKey = @"<objectKey>";
request.uploadingFileURL = [NSURL fileURLWithPath:@<filepath>"];

// 设置回调参数
request.callbackParam = @{
@"callbackUrl": @"<your server callback address>",
@"callbackBody": @"<your callback body>"
};
// 设置自定义变量
request.callbackVar = @{
@"<var1>": @"<value1>",
@"<var2>": @"<value2>"
};

request.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpectedToSend) {
NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};

OSSTask * task = [client putObject:request];

[task continueWithBlock:^id(OSSTask *task) {
if (task.error) {
OSSLogError(@"%@", task.error);
} else {
OSSPutObjectResult * result = task.result;
NSLog(@"Result - requestId: %@, headerFields: %@, servercallback: %@",
result.requestId,
result.httpResponseHeaderFields,
result.serverReturnJsonString);
}
return nil;
}];
```

分片上传

因为篇幅的原因，分片上传参考：分片上传

断点续传

特别注意：

> 断点续传暂时只支持上传本地文件。
>
> 对于移动端来说，如果不是比较大的文件，不建议使用这种方式上传，因为断点续传是通过分片上传实现的，上传单个文件需要进行多次网络请求，效率不高。**

在无线网络下，上传比较大的文件持续时间长，可能会遇到因为网络条件差、用户切换网络等原因导致上传中途失败，整个文件需要重新上传。为此，SDK提供了断点上传功能。

在上传前，可以指定断点记录的保存文件夹。若不进行此项设置，断点上传只在本次上传生效，某个分片因为网络原因等上传失败时会进行重试，避免整个大文件重新上传，节省重试时间和耗用流量。如果设置了断点记录的保存文件夹，如果任务失败，在下次重新启动任务，上传同一文件到同一Bucket、Object时，如果用户设置取消时不删除断点记录。再次上传将从断点记录处继续上传。详见随后的范例。

断点续传失败时，如果同一任务一直得不到续传，可能会在OSS上积累无用碎片。对这种情况，可以为Bucket设置lifeCycle规则，定时清理碎片。参考：生命周期管理。

出于碎片管理的原因，如果在断点续传时取消当前任务。默认会同步清理已经上传到服务器的分片。如果取消时需要保留断点上传记录，需要指定断点记录的保存文件夹并修改deleteUploadIdOnCancelling参数。需要注意，如果本地保留记录时间过长，且Bucket设置lifeCycle规则定时清理了服务端分片。会出现服务端和移动端记录不一致的问题。

说明：

> 断点续传的实现依赖InitMultipartUpload/UploadPart/ListParts/CompleteMultipartUpload/AbortMultipartUpload，如果采用STS鉴权模式，请注意加上这些API所需的权限。
>
> 断点续传也支持上传后回调通知，用法和上述普通上传回调通知一致。
>
> 断点续传已经默认开启每个分片上传时的Md5校验，请勿重复在request中设置Content-Md5头部。

在本地持久保存断点记录的调用方式(默认是不设置)：

```
OSSResumableUploadRequest * resumableUpload = [OSSResumableUploadRequest new];
resumableUpload.bucketName = OSS_BUCKET_PRIVATE;
//...
```

```
NSString *cachesDir = [NSSearchPathForDirectoriesInDomains(NSCachesDirectory, NSUserDomainMask, YES)
firstObject];
resumableUpload.recordDirectoryPath = cachesDir;
```

断点续传功能实现

```
// 获得UploadId进行上传，如果任务失败并且可以续传，利用同一个UploadId可以上传同一文件到同一个OSS上的存储对象
OSSResumableUploadRequest * resumableUpload = [OSSResumableUploadRequest new];
resumableUpload.bucketName = <bucketName>;
resumableUpload.objectKey = <objectKey>;
resumableUpload.partSize = 1024 * 1024;
resumableUpload.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpectedToSend) {
NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};
NSString *cachesDir = [NSSearchPathForDirectoriesInDomains(NSCachesDirectory, NSUserDomainMask, YES)
firstObject];
//设置断点记录文件
resumableUpload.recordDirectoryPath = cachesDir;
//设置NO,取消时，不删除断点记录文件，如果不进行设置，默认YES，是会删除断点记录文件，下次再进行上传时会重新上传
。
resumableUpload.deleteUploadIdOnCancelling = NO;

resumableUpload.uploadingFileURL = [NSURL fileURLWithPath:<your file path>];
OSSTask * resumeTask = [client resumableUpload:resumableUpload];
[resumeTask continueWithBlock:^id(OSSTask *task) {
if (task.error) {
NSLog(@"error: %@", task.error);
if ([task.error.domain isEqualToString:OSSClientErrorDomain] && task.error.code ==
OSSClientErrorCodeCannotResumeUpload) {
// 该任务无法续传，需要获取新的uploadId重新上传
}
} else {
NSLog(@"Upload file success");
}
return nil;
}];

// [resumeTask waitUntilFinished];

// [resumableUpload cancel];
```

数据完整性校验

因为移动端网络环境的复杂性，OSS SDK提供了基于MD5和CRC64的端到端的数据完整性验证功能。

MD5校验

需要在上传文件时提供文件的Content-MD5值，OSS服务器会帮助用户进行MD5校验，只有在OSS服务器计算接收到的文件得到的MD5值和上传提供的MD5一致时才可以上传成功，从而保证上传数据的完整性。

```
OSSPutObjectRequest * request = [OSSPutObjectRequest new];
```

```
request.bucketName = BUCKET_NAME;
...
request.contentMd5 = [OSSUtil fileMD5String:filepath];
```

CRC校验

与MD5相比，CRC64可以边上传边计算CRC值。

```
// 构造上传请求
OSSPutObjectRequest * request = [OSSPutObjectRequest new];
request.bucketName = OSS_BUCKET_PRIVATE;
///....
request.crcFlag = OSSRequestCRCOpen;
// 开启crc效验后。如果在传输中数据不一致，会提示OSSClientErrorCodeInvalidCRC 错误。

OSSTask * task = [_client putObject:request];
[[task continueWithBlock:^id(OSSTask *task) {
//如果crc效验失败，会有error
XCTAssertNil(task.error);
return nil;
}] waitUntilFinished];
```

下载文件

简单下载

下载文件，可以指定下载为本地文件，或者下载为NSData:

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];

// 必填字段
request.bucketName = @"<bucketName>";
request.objectKey = @"<objectKey>";

// 可选字段
request.downloadProgress = ^(int64_t bytesWritten, int64_t totalBytesWritten, int64_t totalBytesExpectedToWrite) {
// 当前下载段长度、当前已经下载总长度、一共需要下载的总长度
NSLog(@"%lld, %lld, %lld", bytesWritten, totalBytesWritten, totalBytesExpectedToWrite);
};
// request.range = [[OSSRange alloc] initWithStart:0 withEnd:99]; // bytes=0-99，指定范围下载
// request.downloadToFileURL = [NSURL fileURLWithPath:@"<filepath>"]; // 如果需要直接下载到文件，需要指明目标文件地址

OSSTask * getTask = [client getObject:request];

[getTask continueWithBlock:^id(OSSTask *task) {
```

```
if (!task.error) {
NSLog(@"download object success!");
OSSGetObjectResult * getResult = task.result;
NSLog(@"download result: %@", getResult.dowloadedData);
} else {
NSLog(@"download object failed, error: %@" ,task.error);
}
return nil;
}];

// [getTask waitUntilFinished];

// [request cancel];
```

图片处理

OSS图片处理，是OSS对外提供的海量、安全、低成本、高可靠的图片处理服务。用户将原始图片上传保存到OSS，通过简单的 RESTful 接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。图片处理提供图片处理接口，图片上传请使用上传接口。基于OSS图片处理，用户可以搭建自己的图片处理服务。

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放、裁剪、旋转
- 图片效果
- 图片添加图片、文字、图文混合水印
- 自定义图片处理样式，在控制台的 图片处理 -> 样式管理 中定义
- 通过级联处理调用多个图片处理功能

SDK中使用时，只需要在下载图片时，为request设置xOssProcess属性。示例：

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];

request.bucketName = @"<bucketName>";
request.objectKey = @"example.jpg";

// 图片处理
request.xOssProcess = @"image/resize,m_lfit,w_100,h_100";

OSSTask * getTask = [client getObject:request];

[getTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
NSLog(@"download image success!");
OSSGetObjectResult * getResult = task.result;
NSLog(@"download image data: %@", getResult.dowloadedData);
} else {
NSLog(@"download object failed, error: %@" ,task.error);
}
return nil;
```

```
}];

// [getTask waitUntilFinished];

// [request cancel];
```

需要对图片进行其它处理，只要替换request.xOssProcess的值就可以了。需要帮助，请参考：

- 可视化图片处理工具 ImageStyleViever ，可以直观的看到OSS图片处理的结果
- OSS图片处理的功能、使用演示 页面

流式下载

实际上，SDK没有提供stream类型的下载接口，但是提供了类似NSURLSession库的didRecieveData的分段回调功能，下载时，每次得到一段数据，会回调这个函数进行通知。注意，如果设置了这个回调，下载的结果将不再包含实际数据。

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
// required
request.bucketName = @"<bucketName>";
request.objectKey = @"<objectKey>";

// 分段回调函数
request.onRecieveData = ^(NSData * data) {
NSLog(@"Recieve data, length: %ld", [data length]);
};

OSSTask * getTask = [client getObject:request];

[getTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
NSLog(@"download object success!");
} else {
NSLog(@"download object failed, error: %@" ,task.error);
}
return nil;
}];

// [getTask waitUntilFinished];

// [request cancel];
```

指定范围下载

您可以在下载文件时指定一段范围，对于较大的Object适于使用此功能；如果在请求头中使用Range参数，则返回消息中会包含整个文件的长度和此次返回的范围。

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
request.bucketName = @"<bucketName>;
```

```
request.objectKey = @"<objectKey>";
request.range = [[OSSRange alloc] initWithStart:1 withEnd:99]; // bytes=1-99
// request.range = [[OSSRange alloc] initWithStart:-1 withEnd:99]; // bytes=-99
// request.range = [[OSSRange alloc] initWithStart:10 withEnd:-1]; // bytes=10-

request.downloadProgress = ^(int64_t bytesWritten, int64_t totalBytesWritten, int64_t totalBytesExpectedToWrite) {
NSLog(@"%lld, %lld, %lld", bytesWritten, totalBytesWritten, totalBytesExpectedToWrite);
};

OSSTask * getTask = [client getObject:request];

[getTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
NSLog(@"download object success!");
OSSGetObjectResult * getResult = task.result;
NSLog(@"download result: %@", getResult.dowloadedData);
} else {
NSLog(@"download object failed, error: %@" ,task.error);
}
return nil;
}];

// [getTask waitUntilFinished];

// [request cancel];
```

只获取文件元信息

通过headObject方法可以只获文件元信息而不获取文件的实体。代码如下：

```
OSSHeadObjectRequest * request = [OSSHeadObjectRequest new];
request.bucketName = @"<bucketName>;
request.objectKey = @"<objectKey>";

OSSTask * headTask = [client headObject:request];

[headTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
NSLog(@"head object success!");
OSSHeadObjectResult * result = task.result;
NSLog(@"header fields: %@", result.httpResponseHeaderFields);
for (NSString * key in result.objectMeta) {
NSLog(@"ObjectMeta: %@ - %@", key, [result.objectMeta objectForKey:key]);
}
} else {
NSLog(@"head object failed, error: %@" ,task.error);
}
return nil;
}];
```

数据完整性校验

因为移动端网络环境的复杂性，数据在客户端和服务器之间传输时有可能会出错。因此OSS SDK在下载过程中提供了基于CRC的端到端的数据完整性效验功能。

CRC校验在读取下载过程中时候，如果开启了crc效验，并有在传输中有数据错误的情况。会提示OSSClientErrorCodeInvalidCRC的错误。注：如果用户

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
request.bucketName = ...;
//开启crc效验
request.crcFlag = OSSRequestCRCOpen;

OSSTask * task = [testProxyClient getObject:request];

[[task continueWithBlock:^id(OSSTask *task) {
//如果开启了crc效验，并有在传输中有数据错误的情况。会提示OSSClientErrorCodeInvalidCRC的错误
XCTAssertNil(task.error);
return nil;
}] waitUntilFinished];

注：如果设置onReceiveData block。开启CRC效验后，需要自行比较CRC数值是否一致。范例如下
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
request.bucketName = ....
request.crcFlag = OSSRequestCRCOpen;
....

__block uint64_t localCrc64 = 0;
//如果设置onReceiveData block
NSMutableData *receivedData = [NSMutableData data];
request.onRecieveData = ^(NSData *data) {
if (data)
{
NSMutableData *mutableData = [data mutableCopy];
void *bytes = mutableData.mutableBytes;
localCrc64 = [OSSUtil crc64ecma:localCrc64 buffer:bytes length:data.length];
[receivedData appendData:data];
}
};

__block uint64_t remoteCrc64 = 0;
OSSTask * task = [_client getObject:request];
[[task continueWithBlock:^id(OSSTask *task) {
XCTAssertNil(task.error);
OSSGetObjectResult *result = task.result;
if (result.remoteCRC64ecma)
{
NSScanner *scanner = [NSScanner scannerWithString:result.remoteCRC64ecma];
[scanner scanUnsignedLongLong:&remoteCrc64];
if (remoteCrc64 == localCrc64)
{
NSLog(@"crc64校验成功!");
}
else
{
NSLog(@"crc64校验失败!");
}
```

```
}
return nil;
}] waitUntilFinished];
```

授权访问

SDK支持签名出特定有效时长或者公开的URL，用于转给第三方实现授权访问。

签名私有资源的指定有效时长的访问URL

如果Bucket或Object不是公共可读的，那么需要调用以下接口，获得签名后的URL：

```
NSString * constrainURL = nil;

// sign constrain url
OSSTask * task = [client presignConstrainURLWithBucketName:@"<bucket name>"
                                             withObjectKey:@"<object key>"
                                    withExpirationInterval: 30 * 60];
if (!task.error) {
    constrainURL = task.result;
} else {
    NSLog(@"error: %@", task.error);
}
```

签名公开的访问URL

如果Bucket或Object是公共可读的，那么调用一下接口，获得可公开访问Object的URL：

```
NSString * publicURL = nil;

// sign public url
task = [client presignPublicURLWithBucketName:@"<bucket name>"
                                withObjectKey:@"<object key>"];
if (!task.error) {
    publicURL = task.result;
} else {
    NSLog(@"sign url error: %@", task.error);
}
```

分片上传

下面演示通过分片上传文件的整个流程：

初始化分片上传

```
__block NSString * uploadId = nil;
__block NSMutableArray * partInfos = [NSMutableArray new];

NSString * uploadToBucket = @"<bucketName>";
NSString * uploadObjectkey = @"<objectKey>";

OSSInitMultipartUploadRequest * init = [OSSInitMultipartUploadRequest new];
init.bucketName = uploadToBucket;
init.objectKey = uploadObjectkey;

// init.contentType = @"application/octet-stream";

OSSTask * initTask = [client multipartUploadInit:init];

[initTask waitUntilFinished];

if (!initTask.error) {
OSSInitMultipartUploadResult * result = initTask.result;
uploadId = result.uploadId;
} else {
NSLog(@"multipart upload failed, error: %@", initTask.error);
return;
}
```

- OSSInitMultipartUploadRequest指定上传文件的所属存储空间Bucket和文件名字。
- multipartUploadInit返回的结果中包含UploadId，是区分分片上传的唯一标示，后面的操作中会用到。

上传分片

```
NSString * filePath = [docDir stringByAppendingPathComponent:@"***"];
//要上传的文件
int chuckCount = *;
//分片上传数量
uint64_t offset = fileSie/chuckCount;
//分片大小
for (int i = 1; i <= chuckCount; i++) {
OSSUploadPartRequest * uploadPart = [OSSUploadPartRequest new];
uploadPart.bucketName = uploadToBucket;
uploadPart.objectkey = uploadObjectkey;
uploadPart.uploadId = uploadId;
```

```
uploadPart.partNumber = i; // part number start from 1

NSFileHandle* readHandle = [NSFileHandle fileHandleForReadingAtPath:filePath];
[readHandle seekToFileOffset:offset * (i -1)];

NSData* data = [readHandle readDataOfLength:offset];
uploadPart.uploadPartData = data;

OSSTask * uploadPartTask = [client uploadPart:uploadPart];

[uploadPartTask waitUntilFinished];

if (!uploadPartTask.error) {
OSSUploadPartResult * result = uploadPartTask.result;
uint64_t fileSize = [[[NSFileManager defaultManager]
attributesOfItemAtPath:uploadPart.uploadPartFileURL.absoluteString error:nil] fileSize];
[partInfos addObject:[OSSPartInfo partInfoWithPartNum:i eTag:result.eTag size:fileSize]];
} else {
NSLog(@"upload part error: %@", uploadPartTask.error);
return;
}
}
```

上述代码调用uploadPart来上传每一个分片，注意：

- 每一个分片上传请求需指定UploadId和PartNum。
- uploadPart要求除最后一个Part外，其他的Part大小都要大于100KB。但是Upload Part接口并不会立即校验上传。Part的大小（因为不知道是否为最后一块）；只有当Complete Multipart Upload的时候才会校验。
- Part号码的范围是1~10000。如果超出这个范围，OSS将返回InvalidArgument的错误码。
- 每次上传part时都要把流定位到此次上传片开头所对应的位置。
- 每次上传part之后，OSS的返回结果会包含一个分片的ETag，值为part数据的MD5值，您需要将它和块编号组合成PartEtag保存，后续完成分片上传需要用到。

MD5校验设置

因为移动端网络环境的复杂性，OSS提供了基于端到端的文件验证功能。如果要校验分片上传到OSS的文件和本地文件是否一致，可以在上传文件时携带文件的Content-MD5值，OSS服务器会帮助用户进行MD5校验，只有在OSS服务器接收到的文件MD5值和Content-MD5一致时才可以上传成功，从而保证上传文件的一致性。

```
OSSUploadPartRequest * uploadPart = [OSSUploadPartRequest new];
uploadPart.bucketName = TEST_BUCKET;
uploadPart.uploadId = uploadId;
....
uploadPart.contentMd5 = [OSSUtil fileMD5String:filepath];
```

完成分片上传

下面代码中的 partInfos就是进行分片上传中保存的partETag的列表，OSS收到用户提交的Part列表后，会逐

一验证每个数据Part的有效性。当所有的数据Part验证通过后，OSS会将这些part组合成一个完整的文件。

```
OSSCompleteMultipartUploadRequest * complete = [OSSCompleteMultipartUploadRequest new];
complete.bucketName = uploadToBucket;
complete.objectKey = uploadObjectkey;
complete.uploadId = uploadId;
complete.partInfos = partInfos;

OSSTask * completeTask = [client completeMultipartUpload:complete];

[[completeTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
OSSCompleteMultipartUploadResult * result = task.result;
// ...
} else {
// ...
}
return nil;
}] waitUntilFinished];
```

完成分片上传（设置ServerCallback）

完成分片上传请求可以设置Server Callback参数，请求完成后会向指定的Server Adress发送回调请求；可通过查看返回结果的result.serverReturnJsonString，查看servercallback结果。

```
OSSCompleteMultipartUploadRequest * complete = [OSSCompleteMultipartUploadRequest new];
complete.bucketName = @"<bucketName>";
complete.objectKey = @"<objectKey>";
complete.uploadId = uploadId;
complete.partInfos = partInfos;
complete.callbackParam = @{
@"callbackUrl": @"<server address>",
@"callbackBody": @"<test>"
};
complete.callbackVar = @{
@"var1": @"value1",
@"var2": @"value2"
};
OSSTask * completeTask = [client completeMultipartUpload:complete];

[[completeTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
OSSCompleteMultipartUploadResult * result = task.result;
NSLog(@"server call back return : %@", result.serverReturnJsonString);
} else {
// ...
}
return nil;
}] waitUntilFinished];
```

删除分片上传事件

下面代码取消了对应UploadId的分片上传请求。

```
OSSAbortMultipartUploadRequest * abort = [OSSAbortMultipartUploadRequest new];
abort.bucketName = @"<bucketName>";
abort.objectKey = @"<objectKey>";
abort.uploadId = uploadId;

OSSTask * abortTask = [client abortMultipartUpload:abort];

[abortTask waitUntilFinished];

if (!abortTask.error) {
OSSAbortMultipartUploadResult * result = abortTask.result;
uploadId = result.uploadId;
} else {
NSLog(@"multipart upload failed, error: %@", abortTask.error);
return;
}
```

罗列分片

调用listParts方法获取某个上传事件所有已上传的分片。

```
OSSListPartsRequest * listParts = [OSSListPartsRequest new];
listParts.bucketName = @"<bucketName>";
listParts.objectKey = @"<objectkey>";
listParts.uploadId = @"<uploadid>";

OSSTask * listPartTask = [client listParts:listParts];

[listPartTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
NSLog(@"list part result success!");
OSSListPartsResult * listPartResult = task.result;
for (NSDictionary * partInfo in listPartResult.parts) {
NSLog(@"each part: %@", partInfo);
}
} else {
NSLog(@"list part result error: %@", task.error);
}
return nil;
}];
```

注：默认情况下，如果存储空间中的分片上传事件的数量大于1000，则只会返回1000个Multipart Upload信息，且返回结果中 IsTruncated 为false，并返回 NextPartNumberMarker作为下此读取的起点。

管理文件

罗列Bucket所有Object

```
OSSGetBucketRequest * getBucket = [OSSGetBucketRequest new];
getBucket.bucketName = @"<bucketName>";

// 可选参数，具体含义参考：https://docs.aliyun.com/#/pub/oss/api-reference/bucket&GetBucket
// getBucket.marker = @"";
// getBucket.prefix = @"";
// getBucket.delimiter = @"";

OSSTask * getBucketTask = [client getBucket:getBucket];

[getBucketTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
OSSGetBucketResult * result = task.result;
NSLog(@"get bucket success!");
for (NSDictionary * objectInfo in result.contents) {
NSLog(@"list object: %@", objectInfo);
}
} else {
NSLog(@"get bucket failed, error: %@", task.error);
}
return nil;
}];
```

罗列操作具体可设置的参数名称和作用如下：

名称	作用
delimiter	用于对Object名字进行分组的字符。所有名字包含指定的前缀且第一次出现delimiter字符之间的object作为一组元素: CommonPrefixes。
marker	设定结果从marker之后按字母排序的第一个开始返回。
maxkeys	限定此次返回object的最大数，如果不设定，默认为100，maxkeys取值不能大于1000。
prefix	限定返回的object key必须以prefix作为前缀。注意使用prefix查询时，返回的key中仍会包含prefix。

检查文件是否存在

SDK提供了同步接口检测某个指定Object是否在OSS上：

```
NSError * error = nil;
BOOL isExist = [client doesObjectExistInBucket:TEST_BUCKET withObjectKey:@"file1m" withError:&error];
if (!error) {
if(isExist) {
NSLog(@"File exists.");
} else {
NSLog(@"File not exists.");
}
} else {
NSLog(@"Error!");
}
```

复制Object

```
OSSCopyObjectRequest * copy = [OSSCopyObjectRequest new];
copy.bucketName = @<bucketName>;
copy.objectKey = @"<objectKey>";
copy.sourceCopyFrom = [NSString stringWithFormat:@"/%@/%@", @"<bucketName>,
@"<objectKey_copyFrom>"];

OSSTask * task = [client copyObject:copy];

[task continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
// ...
}
return nil;s
}];
```

上述代码实现了CopyObject，注意：

- 源Object和目标Object必须属于同一个数据中心。
- 如果拷贝操作的源Object地址和目标Object地址相同，可以修改已有Object的meta信息。
- 拷贝文件大小不能超过1G，超过1G需使用Multipart Upload操作。

删除Object

下面代码实现了DeleteObject，要求对所在的Bucket有写权限。

```
OSSDeleteObjectRequest * delete = [OSSDeleteObjectRequest new];
delete.bucketName = @"<bucketName>";
delete.objectKey = @"<objectKey>";

OSSTask * deleteTask = [client deleteObject:delete];

[deleteTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
// ...
}
```

```
return nil;
}];

// [deleteTask waitUntilFinished];
```

只获取Object的Meta信息

下面代码用于获取Object的元信息：

```
OSSHeadObjectRequest * head = [OSSHeadObjectRequest new];
head.bucketName = @"<bucketName>";
head.objectKey = @"<objectKey>";

OSSTask * headTask = [client headObject:head];

[headTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
OSSHeadObjectResult * headResult = task.result;
NSLog(@"all response header: %@", headResult.httpResponseHeaderFields);

// some object properties include the 'x-oss-meta-*'s
NSLog(@"head object result: %@", headResult.objectMeta);
} else {
NSLog(@"head object error: %@", task.error);
}
return nil;
}];
```

管理Bucket

创建bucket

```
OSSCreateBucketRequest * create = [OSSCreateBucketRequest new];
create.bucketName = @"<bucketName>";
create.xOssACL = @"public-read";
create.location = @"oss-cn-hangzhou";

OSSTask * createTask = [client createBucket:create];

[createTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
NSLog(@"create bucket success!");
} else {
NSLog(@"create bucket failed, error: %@", task.error);
```

```
}
return nil;
}];
```

上述代码在创建bucket时，指定了Bucket的ACL和所在的数据中心。

- 每个用户的Bucket数量不能超过30个。
- 每个Bucket的名字全局唯一，也就是说创建的Bucket不能和其他用户已经在使用的Bucket同名，否则会创建失败。
- 创建的时候可以选择Bucket ACL权限，如果不设置ACL，默认是private。
- 创建成功结果返回Bucket所在数据中心。

罗列所有bucket

```
OSSGetServiceRequest * getService = [OSSGetServiceRequest new];
OSSTask * getServiceTask = [client getService:getService];
[getServiceTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
OSSGetServiceResult * result = task.result;
NSLog(@"buckets: %@", result.buckets);
NSLog(@"owner: %@, %@", result.ownerId, result.ownerDispName);
[result.buckets enumerateObjectsUsingBlock:^(id _Nonnull obj, NSUInteger idx, BOOL * _Nonnull stop) {
NSDictionary * bucketInfo = obj;
NSLog(@"BucketName: %@", [bucketInfo objectForKey:@"Name"]);
NSLog(@"CreationDate: %@", [bucketInfo objectForKey:@"CreationDate"]);
NSLog(@"Location: %@", [bucketInfo objectForKey:@"Location"]);
}];
}
return nil;
}];
```

上处代码返回请求者拥有的所有Bucket。

- 匿名访问不支持该操作。

罗列bucket中的文件

```
OSSGetBucketRequest * getBucket = [OSSGetBucketRequest new];
getBucket.bucketName = @"<bucketName>";
// getBucket.marker = @"";
// getBucket.prefix = @"";
// getBucket.delimiter = @"";

OSSTask * getBucketTask = [client getBucket:getBucket];

[getBucketTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
OSSGetBucketResult * result = task.result;
NSLog(@"get bucket success!");
```

```
for (NSDictionary * objectInfo in result.contents) {
NSLog(@"list object: %@", objectInfo);
}
} else {
NSLog(@"get bucket failed, error: %@", task.error);
}
return nil;
}];
```

上述代码罗列了Bucket中的文件

- 罗列操作必须具备访问该Bukcet的权限。
- 罗列时，可以通过prefix，marker，delimiter和max-keys对list做限定，返回部分结果。

删除bucket

```
OSSDeleteBucketRequest * delete = [OSSDeleteBucketRequest new];
delete.bucketName = @"<bucketName>";

OSSTask * deleteTask = [client deleteBucket:delete];

[deleteTask continueWithBlock:^id(OSSTask *task) {
if (!task.error) {
NSLog(@"delete bucket success!");
} else {
NSLog(@"delete bucket failed, error: %@", task.error);
}
return nil;
}];
```

上述代码删除了一个Bucket。

- 只有Bucket的拥有者才能删除这个Bucket。
- 为了防止误删除的发生，OSS不允许用户删除一个非空的Bucket。

异常响应

SDK中发生的异常分为两类：ClientError和ServerError。其中前者指的是参数错误、网络错误等，后者指OSS Server返回的异常响应。

Error类型	Error Domain	Code	UserInfo
ClientError	com.aliyun.oss.clientError	OSSClientErrorCodeNetworkingFailWithResponseCode0	连接异常
		OSSClientErrorCode	签名失败

		SignFailed	
		OSSClientErrorCode FileCantWrite	文件无法写入
		OSSClientErrorCodeI nvalidArgument	参数非法
		OSSClientErrorCode NilUploadid	断点续传任务未获取到uploadId
		OSSClientErrorCode TaskCancelled	任务被取消
		OSSClientErrorCode NetworkError	网络异常
		OSSClientErrorCode CannotResumeUplo ad	断点上传失败，无法继续上传
		OSSClientErrorCode NotKnown	未知异常
ServerError	com.aliyun.oss.serve rError	(-1 * httpResponseCode)	解析响应XML得到的Dictionary

.NET-SDK

前言

简介

- OSS C# SDK适用的 .NET Framework 版本为2.0及其以上。
- 本文档主要介绍OSS C# SDK的安装和使用，针对于OSS C# SDK版本 2.8.0。
- 并且假设您已经开通了阿里云OSS服务，并创建了 AccessKeyId 和 AccessKeySecret。
- 如果您还没有开通或者还不了解阿里云OSS服务，请登录OSS产品主页了解。
- 如果还没有创建 AccessKeyId和 AccessKeySecret，请到阿里云控制台上创建。

SDK下载

- 当前最新版本：2.8.0

- SDK包：aliyun_oss_dotnet_sdk_2_8_0.zip
- API文档地址：API Doc
- GitHub地址：GitHub
- 版本迭代

兼容性

对于2.x.x 系列SDK：

- 接口：
  - 兼容
- 命名空间：
  - 兼容

对于1.0.x 系列SDK：

- 接口：
  - 兼容
- 命名空间：
  - 不兼容：Aliyun.OpenServices.OpenStorageService变更为Aliyun.OSS

安装

SDK安装

版本依赖

Windows

- 适用于.NET 2.0 及以上版本
- 适用于Visual Studio 2010及以上版本

Linux / Mac

- 适用于Mono 3.12 及以上版本

版本迭代

- 版本迭代详情，请参考Change Log。

Windows环境安装

NuGet安装

- 如果您的Visual Studio没有安装NuGet，请先安装 NuGet。
- 安装好NuGet后，先在Visual Studio中新建或者打开已有的项目，然后选择**工具** > **NuGet程序包管理器** > **管理解决方案的NuGet程序包。**
- 搜索 **aliyun.oss.sdk**，在结果中找到**Aliyun.OSS.SDK**，选择最新版本，单击**安装**，成功后添加到项目应用中。

GitHub安装

- 如果没有安装git，请先安装 git。
- git clone。请参见 https://github.com/aliyun/aliyun-oss-csharp-sdk.git 。
- 下载好源码后，按照项目引入方式安装即可。

DLL引用方式安装

- 下载SDK包 aliyun_oss_dotnet_sdk_2_8_0.zip，解压后bin目录包括了Aliyun.OSS.dll文件。
- 在Visual Studio的**解决方案资源管理器**中选择您的项目，然后右键**项目名称** > **引用**，在弹出的菜单中选择**添加引用**，在弹出**添加引用**对话框后，选择**浏览**，找到SDK包解压的目录，在bin目录下选中**Aliyun.OSS.dll**文件，单击**确定**即可。

项目引入方式安装

- 如果是下载了SDK包或者从GitHub上下载了源码，希望源码安装，可以右键**解决方案**，在弹出的菜单中单击**添加** > **现有项目。**
- 在弹出的对话框中选择**aliyun-oss-sdk.csproj**文件，单击**打开。**
- 接下来右键**您的项目** > **引用**，选择**添加引用**，在弹出的对话框选择**项目**选项卡后选中**aliyun-oss-sdk**项目，单击**确定**即可。

Unix/Mac环境安装

NuGet安装

- 先在Xamarin中新建或者打开已有的项目，然后选择**工具** > **Add NuGet Packages。**
- 搜索**aliyun.oss.sdk**，在结果中找到**Aliyun.OSS.SDK**，选择最新版本，单击**Add Package**，成功后添加到项目应用中。

GitHub安装

- 如果没有安装git，请先安装 git。
- git clone。请参见 https://github.com/aliyun/aliyun-oss-csharp-sdk.git 。
- 下载好源码后，使用Xamarin打开，在Release模式下编译aliyun-oss-sdk项目，生成

Aliyun.OSS.dll，然后通过DLL引用方式安装。

DLL引用方式安装

- 从下载SDK包 aliyun_oss_dotnet_sdk_2_8_0.zip，解压后bin目录包括了Aliyun.OSS.dll文件。
- 在Xamarin的**解决方案**中选择您的项目，然后右键**项目名称** > **引用**，在弹出的菜单中选择**Edit References**。在**Edit References**对话框中，选择**.Net Assembly** > **浏览**>，找到SDK包解压的目录，在bin目录下选中**Aliyun.OSS.dll**文件，单击**Open**即可。

示例程序

OSS C# SDK提供丰富的示例程序，方便用户参考或直接使用。您可以从GitHub获取示例程序。示例程序包括以下内容：

示例文件	示例内容
PutObjectSample.cs	展示了Object上传的用法
AppendObjectSample.cs	展示了Object追加上传的用法
DoesObjectExistSample.cs	展示了判断Object是否存在的用法
DeleteObjectsSample.cs	展示了Object删除的用法
CopyObjectSample.cs	展示了Object复制的用法
ModifyObjectMetaSample.cs	展示了修改Object Meta的用法
MultipartUploadSample.cs	展示了分片上传的用法
ResumableSample.cs	展示了断点续传上传的用法
GetObjectSample.cs	展示了Object下载的用法
GetObjectByRangeSample.cs	展示了Object范围下载的用法
GetObjectAclSample.cs	展示了获取Object访问权限的用法
SetObjectAclSample.cs	展示了设置Object访问权限的用法
ListObjectsSample.cs	展示了列举Objects的用法
UrlSignatureSample.cs	展示了授权访问的用法
UploadCallbackSample.cs	展示了上传回调的用法
ProgressSample.cs	展示了上传下载进度条的用法
CNameSample.cs	展示使用CName访问OSS的用法
PostPolicySample.cs	展示了PostObject的用法
CreateBucketSample.cs	展示了创建Bucket的用法
DeleteBucketSample.cs	展示了删除Bucket的用法
DoesBucketExistSample.cs	展示了判断Bucket是否存在的用法
ListBucketsSample.cs	展示了列举Bucket的用法
SetBucketAclSample.cs	展示了设置Bucket的访问权限的用法

SetBucketLifecycleSample.cs	展示了设置Bucket中Objects生命周期的用法
SetBucketLoggingSample.cs	展示了设置Bucket访问日志的用法
SetBucketRefererSample.cs	展示了设置Bucket防盗链的用法
SetBucketWetbsiteSample.cs	展示了设置Bucket静态网站托管的用法
SetBucketCorsSample.cs	展示了设置Bucket跨域访问的用法
ImageProcessSample.cs	展示了图片处理的用法

初始化

OssClient是OSS服务的C#客户端，它为调用者提供了一系列的方法，可以用来操作，管理存储空间（Bucket）和文件（Object）等。

确定Endpoint

Endpoint是阿里云OSS服务在各个区域的域名地址，目前支持两种形式

Endpoint类型	**解释**
OSS区域地址	使用OSS Bucket所在区域地址，各个区域Endpoint参考这里
用户自定义域名	用户自定义域名，且CNAME指向OSS域名

OSS区域地址

使用OSS Bucket所在区域地址，Endpoint查询可以通过以下两种方式：

- 查询Endpoint与区域对应关系详情，可以参考：点击查看。
- 您可以登录 阿里云OSS控制台，进入Bucket概览页，Bucket域名的后缀部分：如bucket-1.oss-cn-hangzhou.aliyuncs.com的oss-cn-hangzhou.aliyuncs.com部分为该Bucket的外网Endpoint。

CNAME

- 您可以将自己拥有的域名通过CNAME绑定到某个存储空间（Bucket）上，然后通过自己域名访问存储空间内的文件
- 比如您要将域名new-image.xxxxx.com绑定到深圳区域的名称为image的存储空间上：
- 您需要到您的域名xxxxx.com托管商那里设定一个新的域名解析，将http://new-image.xxxxx.com解析到 http://image.oss-cn-shenzhen.aliyuncs.com ，类型为CNAME

配置密钥

要接入阿里云OSS，您需要拥有一个有效的 Access Key(包括AccessKeyId和AccessKeySecret)用来进行签名认证。可以通过如下步骤获得：

- 注册阿里云账号
- 申请AccessKey

在获取到 AccessKeyId和 AccessKeySecret之后，您可以按照下面步骤进行初始化对接

新建Client

使用OSS域名新建Client

新建一个OssClient很简单，如下面代码所示：

```
using Aliyun.OSS;

const string accessKeyId = "<your AccessKeyId>";
const string accessKeySecret = "<your AccessKeySecret>";
const string endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

/// <summary>
/// 由用户指定的OSS访问地址、阿里云颁发的AccessKeyId/AccessKeySecret构造一个新的OssClient实例。
/// </summary>
/// <param name="endpoint">OSS的访问地址。</param>
/// <param name="accessKeyId">OSS的访问ID。</param>
/// <param name="accessKeySecret">OSS的访问密钥。</param>
var ossClient = new OssClient(endpoint, accessKeyId, accessKeySecret);
```

使用自定义域名（CNAME）新建Client

下面的代码让客户端使用CNAME访问OSS服务：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;

// 创建ClientConfiguration实例
var conf = new ClientConfiguration();

// 配置使用Cname
conf.IsCname = true;

/// <summary>
/// 由用户指定的OSS访问地址、、阿里云颁发的AccessKeyId/AccessKeySecret、客户端配置
/// 构造一个新的OssClient实例。
/// </summary>
/// <param name="endpoint">OSS的访问地址。</param>
```

```
/// <param name="accessKeyId">OSS的访问ID。</param>
/// <param name="accessKeySecret">OSS的访问密钥。</param>
/// <param name="conf">客户端配置。</param>
var client = new OssClient(endpoint, accessKeyId, accessKeySecret, conf);
```

注意：

- 使用CNAME时，无法使用ListBuckets接口。

配置Client

如果您想配置OssClient的一些细节的参数，可以在构造OssClient的时候传入ClientConfiguration对象。ClientConfiguration是OSS服务的配置类，可以为客户端配置代理，最大连接数等参数。

网络参数

您可以用ClientConfiguration设置一些网络参数：

参数	描述	默认值
ConnectionLimit	最大并发连接数	512
MaxErrorRetry	请求发生错误时最大重试次数	3
ConnectionTimeout	设置连接超时时间，单位毫秒，默认不超时	-1
SetCustomEpochTicks	设置自定义基准时间，即与本机时间差，单位秒	0
IsCname	Endpoint是否是Cname	false
ProgressUpdateInterval	进度条更新间隔，单位字节	8096

示例如下：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;

var conf = new ClientConfiguration();
conf.ConnectionLimit = 512;
conf.MaxErrorRetry = 3;
conf.ConnectionTimeout = 300;

var client = new OssClient(endpoint, accessKeyId, accessKeySecret, conf);
```

数据校验

您可以设置参数EnalbeMD5Check，对上传下载的数据自动进行MD5校验，默认关闭。请使用以下代码打开

MD5校验：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;

var conf = new ClientConfiguration();
conf.EnalbeMD5Check = true;

var client = new OssClient(endpoint, accessKeyId, accessKeySecret, conf);
```

提示：

- MD5校验是边上传数据边计算校验和，与服务器返回的校验和比较，两者不符则抛ClientException异常。
- MD5校验会有一定的计算和性能开销。

代理网络

如果您使用的网络是代理（Proxy）网络，可以配置以下参数访问OSS：

参数	描述	默认值
ProxyHost	代理服务器，如8.8.8.8 或 abc.def.com	空
ProxyPort	代理端口，如3128 或 8080	空
ProxyUserName	代码服务账号，可选参数	无
ProxyPassword	代码服务密码，可选参数	无

无账号密码的代理网络访问示例：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;

var conf = new ClientConfiguration();
conf.ProxyHost = "8.8.8.8";
conf.ProxyPort = 3128;

var client = new OssClient(endpoint, accessKeyId, accessKeySecret, conf);
```

带账号密码的代理网络访问示例：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;

var conf = new ClientConfiguration();
conf.ProxyHost = "8.8.8.8";
conf.ProxyPort = 3128;
```

```
conf.ProxyUserName = "user";
conf.ProxyPassword = "6666";

var client = new OssClient(endpoint, accessKeyId, accessKeySecret, conf);
```

快速入门

在这一章里，我们将学到如何用OSS .NET SDK完成一些基本的操作。

Step-1.初始化一个OssClient

SDK的OSS操作通过OssClient类完成的，下面代码创建一个OssClient对象:

```
using Aliyun.OSS;

/// <summary>
/// 由用户指定的OSS访问地址、阿里云颁发的AccessKeyId/AccessKeySecret构造一个新的OssClient实例。
/// </summary>
/// <param name="endpoint">OSS的访问地址。</param>
/// <param name="accessKeyId">OSS的访问ID。</param>
/// <param name="accessKeySecret">OSS的访问密钥。</param>
public void CreateClient(string endpoint, string accessKeyId, string accessKeySecret)
{
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
}
```

提示：

- 在上面代码中，变量 accessKeyId 与 accessKeySecret 是由系统分配给用户的，用于标识用户，可能属于您的阿里云账号或者RAM账号。
- 关于OssClient的详细介绍，参见 初始化。

Step-2. 新建存储空间

存储空间（Bucket）是OSS全局命名空间，相当于数据的容器，可以存储若干文件。您可以按照下面的代码新建一个存储空间：

```
using Aliyun.OSS;

/// <summary>
/// 在OSS中创建一个新的存储空间。
```

```
/// </summary>
/// <param name="bucketName">要创建的存储空间的名称</param>
public void CreateBucket(string bucketName)
{
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
var bucket = client.CreateBucket(bucketName);

Console.WriteLine("Create bucket succeeded.");

Console.WriteLine("Name:{0}", bucket.Name);
}
catch (Exception ex)
{
Console.WriteLine("Create bucket failed, {0}", ex.Message);
}
}
```

提示：

- 关于存储空间的命名规范，参见管理Bucket中的命名规范。

Step-3. 上传文件

文件（Object）是OSS中最基本的数据单元，用下面代码可以实现上传文件：

```
using Aliyun.OSS;

/// <summary>
/// 上传指定的文件到指定的OSS的存储空间
/// </summary>
/// <param name="bucketName">指定的存储空间名称</param>
/// <param name="key">文件的在OSS上保存的名称</param>
/// <param name="fileToUpload">指定上传文件的本地路径</param>
public void PutObject(string bucketName, string key, string fileToUpload)
{
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

try
{
var result = client.PutObject(bucketName, key, fileToUpload);

Console.WriteLine("Put object succeeded");
Console.WriteLine("ETag:{0}", result.ETag);
}
catch (Exception ex)
{
Console.WriteLine("Put object failed, {0}", ex.Message);
}
}
```

注意：

- 每个上传的文件，都可以指定和此文件关联的ObjectMeta。
- ObjectMeta是用户对该文件的描述，由一系列key-value对组成；
- 为了保证上传文件服务器端与本地一致，用户可以设置Content-MD5，OSS会计算上传数据的MD5值并与用户上传的MD5值比较，如果不一致返回InvalidDigest错误码。
- 计算出来的Content-MD5需要在上传时设置给ObjectMetadata的ETag。
- 关于文件的命名规范和其他更详细的信息，参见上传文件 。

Step-4. 列出所有文件

当您完成一系列上传后，可能需要查看某个存储空间中有哪些文件，可以通过下面的程序实现：

```
using Aliyun.OSS;

/// <summary>
/// 列出指定存储空间的文件列表
/// </summary>
/// <param name="bucketName">存储空间的名称</param>
public void ListObjects(string bucketName)
{
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

try
{
var listObjectsRequest = new ListObjectsRequest(bucketName);
var result = client.ListObjects(listObjectsRequest);

Console.WriteLine("List object succeeded");
foreach (var summary in result.ObjectSummaries)
{
Console.WriteLine(summary.Key);
}
}
catch (Exception ex)
{
Console.WriteLine("List object failed, {0}", ex.Message);
}
}
```

Step-5. 获取指定文件

您可以参考下面的代码简单地实现一个文件的获取：

```
using Aliyun.OSS;

var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
```

```
/// <summary>
/// 从指定的OSS存储空间中获取指定的文件
/// </summary>
/// <param name="bucketName">要获取的文件所在的存储空间名称</param>
/// <param name="key">要获取的文件在OSS上的名称</param>
/// <param name="fileToDownload">本地存储下载文件的目录<param>
public void GetObject(string bucketName, string key, string fileToDownload)
{
try
{
var object = client.GetObject(bucketName, key);

//将从OSS读取到的文件写到本地
using (var requestStream = object.Content)
{
byte[] buf = new byte[1024];
using (var fs = File.Open(fileToDownload, FileMode.OpenOrCreate);
{
var len = 0;
while ((len = requestStream.Read(buf, 0, 1024)) != 0)
{
fs.Write(buf, 0, len);
}
}
}
}
catch (Exception ex)
{
Console.WriteLine("Get object failed, {0}", ex.Message);
}
}
```

提示：

- 当调用OssClient的GetObject方法时，会返回一个OssObject的对象，此对象包含了文件的各种信息。
- 通过OssObject的GetObjectContent方法，可以获取返回的文件的输入流，通过读取此输入流获取此文件的内容，在用完之后关闭这个流。

Step-6. 删除指定文件

您可以参考下面的代码实现一个文件的删除：

```
using Aliyun.OSS;

/// <summary>
/// 删除指定的文件
/// </summary>
/// <param name="bucketName">文件所在存储空间的名称</param>
/// <param name="key">待删除的文件名称</param>
public void DeleteObject(string bucketName, string key)
```

```
{
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
client.DeleteObject(bucketName, key);
Console.WriteLine("Delete object succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Delete object failed, {0}", ex.Message);
}
}
```

示例程序

下面是一个完整的程序，演示了创建存储空间，设置存储空间ACL，查询存储空间的ACL，上传文件，下载文件，查询文件列表，删除文件，删除存储空间等操作。

```
using System;
using System.Collections.Generic;
using Aliyun.OSS;

namespace TaoYe
{
/// <summary>
/// 快速入门示例程序
/// </summary>
public static class SimpleSamples
{
private const string _accessKeyId = "<your AccessKeyId>";
private const string _accessKeySecret = "<your AccessKeySecret>";
private const string _endpoint = "<valid host name>";

private const string _bucketName = "<your bucket name>";
private const string _key = "<your key>";
private const string _fileToUpload = "<your local file path>";

private static OssClient _client = new OssClient(_endpoint, _accessKeyId, _accessKeySecret);

public static void Main(string[] args)
{
CreateBucket();
SetBucketAcl();
GetBucketAcl();

PutObject();
ListObjects();
GetObject();
DeleteObject();

// DeleteBucket();

Console.WriteLine("Press any key to continue . . . ");
```

```
Console.ReadKey(true);
}

/// <summary>
/// 创建一个新的存储空间
/// </summary>
private static void CreateBucket()
{
try
{
var result = _client.CreateBucket(_bucketName);
Console.WriteLine("创建存储空间{0}成功", result.Name);
}
catch (Exception ex)
{
Console.WriteLine("创建存储空间失败. 原因：{0}", ex.Message);
}
}

/// <summary>
/// 上传一个新文件
/// </summary>
private static void PutObject()
{
try
{
_client.PutObject(_bucketName, _key, _fileToUpload);
Console.WriteLine("上传文件成功");
}
catch (Exception ex)
{
Console.WriteLine("上传文件失败.原因: {0}", ex.Message);
}
}

/// <summary>
/// 列出存储空间内的所有文件
/// </summary>
private static void ListObjects()
{
try
{
var keys = new List<string>();
ObjectListing result = null;
string nextMarker = string.Empty;

/// 由于ListObjects每次最多返回100个结果，所以，这里需要循环去获取，直到返回结果中IsTruncated为false
do
{
var listObjectsRequest = new ListObjectsRequest(_bucketName)
{
Marker = nextMarker,
MaxKeys = 100
};
result = _client.ListObjects(listObjectsRequest);
```

```
foreach (var summary in result.ObjectSummaries)
{
keys.Add(summary.Key);
}

nextMarker = result.NextMarker;
} while (result.IsTruncated);

Console.WriteLine("列出存储空间中的文件");
foreach (var key in keys)
{
Console.WriteLine("文件名称：{0}", key);
}
}
catch (Exception ex)
{
Console.WriteLine("列出存储空间中的文件失败.原因： {0}", ex.Message);
}

}

/// <summary>
/// 下载文件
/// </summary>
private static void GetObject()
{
try
{
var result = _client.GetObject(_bucketName, _key);
Console.WriteLine("下载的文件成功，名称是：{0}，长度：{1}", result.Key, result.Metadata.ContentLength);
}
catch (Exception ex)
{
Console.WriteLine("下载文件失败.原因：{0}", ex.Message);
}
}

/// <summary>
/// 删除文件
/// </summary>
private static void DeleteObject()
{
try
{
_client.DeleteObject(_bucketName, _key);
Console.WriteLine("删除文件成功");
}
catch (Exception ex)
{
Console.WriteLine("删除文件失败.原因： {0}", ex.Message);
}
}

/// <summary>
/// 获取存储空间ACL的值
/// </summary>
```

```
private static void GetBucketAcl()
{
try
{
var result = _client.GetBucketAcl(_bucketName);

foreach (var grant in result.Grants)
{
Console.WriteLine("获取存储空间权限成功，当前权限:{0}", grant.Permission.ToString());
}
}
catch (Exception ex)
{
Console.WriteLine("获取存储空间权限失败.原因： {0}", ex.Message);
}
}

/// <summary>
/// 设置存储空间ACL的值
/// </summary>
private static void SetBucketAcl()
{
try
{
_client.SetBucketAcl(_bucketName, CannedAccessControlList.PublicRead);
Console.WriteLine("设置存储空间权限成功");
}
catch (Exception ex)
{
Console.WriteLine("设置存储空间权限失败. 原因：{0}", ex.Message);
}
}

/// <summary>
/// 删除存储空间
/// </summary>
private static void DeleteBucket()
{
try
{
_client.DeleteBucket(_bucketName);
Console.WriteLine("删除存储空间成功");
}
catch (Exception ex)
{
Console.WriteLine("删除存储空间失败", ex.Message);
}
}
}
}
```

管理Bucket

存储空间（Bucket）是OSS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体；

新建存储空间

如下代码可以新建一个存储空间：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 创建一个新的存储空间（Bucket）
/// </summary>
/// <param name="bucketName">存储空间的名称</param>
public void CreateBucket(string bucketName)
{
try
{
// 新建一个Bucket
client.CreateBucket(bucketName);
Console.WriteLine("Create bucket succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Create bucket failed. {0}", ex.Message);
}
}
```

提示：

- 完整代码参考：GitHub

重要：

- 由于存储空间的名字是全局唯一的，所以必须保证您的BucketName不与别人重复。

列出用户所有的存储空间

下面代码可以列出用户所有的存储空间：

```
using Aliyun.OSS;
```

```
// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 列出账户下所有的存储空间信息
/// </summary>
public void ListBuckets()
{
try
{
var buckets = client.ListBuckets();

Console.WriteLine("List bucket succeeded");
foreach (var bucket in buckets)
{
Console.WriteLine("Bucket name：{0}，Location：{1}，Owner：{2}", bucket.Name, bucket.Location, bucket.Owner);
}
}
catch (Exception ex)
{
Console.WriteLine("List bucket failed. {0}", ex.Message);
}
}
```

提示：

- 完整代码参考：GitHub

使用CNAME进行访问

当用户将自己的域名CNAME指向自己的一个存储空间的域名后，用户可以使用自己的域名来访问OSS：如果需要使用CNAME，需要将ClientConfigration中的IsCname设置为true

```
using Aliyun.OSS;
using Aliyun.OSS.Common;

/// <summary>
/// 通过CNAME上传文件
/// </summary>
public void PutObjectByCname()
{
try
{
// 比如你的域名"http://my-cname.com"CNAME指向你的存储空间域名"mybucket.oss-cn-hangzhou.aliyuncs.com"
// 创建ClientConfiguration实例
var conf = new ClientConfiguration();

// 配置使用Cname
conf.IsCname = true;

var client = new OssClient("http://my-cname.com/", accessKeyId, accessKeySecret, conf);
var result = client.putObject("mybucket", key, fileToUpload);
```

```
Console.WriteLine("Put object succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Put object failed. {0}", ex.Message);
}
}
```

提示：

- 完整代码参考：GitHub

提示：

- 用户只需要在创建OssClinet类实例时，将原本填入该存储空间的endpoint更换成CNAME后的域名，且将ClientConfiguration的参数IsCname设置为true。
- 同时需要注意的是，使用该OssClient实例的后续操作中，存储空间的名称只能填成被指向的存储空间名称。

判断存储空间是否存在

判断存储空间是否存在可以使用以下代码：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 判断存储空间是否存在
/// </summary>
/// <param name="bucketName">存储空间的名称</param>
public void DoesBucketExist(string bucketName)
{
try
{
var exist = client.DoesBucketExist(bucketName);

Console.WriteLine("Check object Exist succeeded");
Console.WriteLine("exist ? {0}", exist);
}
catch (Exception ex)
{
Console.WriteLine("Check object Exist failed. {0}", ex.Message);
}
}
```

提示：

- 完整代码参考：GitHub

设置存储空间访问权限

设置存储空间访问权限可以使用以下代码：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 设置存储空间的访问权限
/// </summary>
/// <param name="bucketName">存储空间的名称</param>
public void SetBucketAcl(string buckteName)
{
try
{
// 指定Bucket ACL为公共读
client.SetBucketAcl(bucketName, CannedAccessControlList.PublicRead);
Console.WriteLine("Set bucket ACL succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Set bucket ACL failed. {0}", ex.Message);
}
}
```

提示：

- 完整代码参考：GitHub

获取存储空间访问权限

获取存储空间访问权限可以使用以下代码：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 获取存储空间的访问权限
/// </summary>
/// <param name="bucketName">存储空间的名称</param>
public void GetBucketAcl(string bucketName)
{
try
```

```
{
string bucketName = "your-bucket";
var acl = client.GetBucketAcl(bucketName);

Console.WriteLine("Get bucket ACL success");
foreach (var grant in acl.Grants)
{
Console.WriteLine("获取存储空间权限成功，当前权限:{0}", grant.Permission.ToString());
}
}
catch (Exception ex)
{
Console.WriteLine("Get bucket ACL failed. {0}", ex.Message);
}
}
```

提示：

- 完整代码参考：GitHub

删除存储空间

下面代码删除了一个存储空间

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 删除存储空间
/// </summary>
/// <param name="bucketName">存储空间的名称</param>
public void DeleteBucket(string bucketName)
{
try
{
client.DeleteBucket(bucketName);

Console.WriteLine("Delete bucket succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Delete bucket failed. {0}", ex.Message);
}
}
```

提示：

- 完整代码参考：GitHub

重要：

- 如果存储空间不为空（存储空间中有文件或者分片上传碎片），则存储空间无法删除
- 必须先删除存储空间中的所有文件后，存储空间才能成功删除。

上传文件

在OSS中，用户操作的基本数据单元是文件（Object）。单个文件最大允许大小根据上传数据方式不同而不同，Put Object方式最大不能超过5GB, 使用multipart上传方式文件大小不能超过48.8TB。

简单的上传

上传指定字符串：

```
using System.Text;
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret)
try
{
string str = "a line of simple text";
byte[] binaryData = Encoding.ASCII.GetBytes(str);
MemoryStream requestContent = new MemoryStream(binaryData);
client.PutObject(bucketName, key, requestContent);
Console.WriteLine("Put object succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

说明： 完整代码参考：GitHub。

上传指定的本地文件

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
```

```
{
string fileToUpload = "your local file path";
client.PutObject(bucketName, key, fileToUpload);
Console.WriteLine("Put object succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

说明： 完整代码参考GitHub。

上传文件并且带MD5校验

```
using Aliyun.OSS;
using Aliyun.OSS.util;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
string fileToUpload = "your local file path";
string md5;
using (var fs = File.Open(fileToUpload, FileMode.Open));
{
md5 = OssUtils.ComputeContentMd5(fs, fs.Length);
}

var objectMeta = new ObjectMetadata
{
ContentMd5 = md5
};
client.PutObject(bucketName, key, fileToUpload, objectMeta);
Console.WriteLine("Put object succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

说明：

- 完整代码参考GitHub。
- 为了保证SDK发送的数据和OSS服务端接收到的数据一致，可以在ObjectMeta中增加Content-Md5值，这样服务端就会使用这个MD5值做校验。
- 使用Md5时，性能会有所损失。

上传文件带Header

带标准Header

OSS服务允许用户自定义文件的Http Header。下面代码为文件设置了过期时间：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

try
{
using (var fs = File.Open(fileToUpload, FileMode.Open))
{
var metadata = new ObjectMetadata();
metadata.ContentLength = fs.Length;
metadata.ExpirationTime = DateTime.Parse("2015-10-12T00:00:00.000Z");
client.PutObject(bucketName, key, fs, metadata);
Console.WriteLine("Put object succeeded");
}
}
catch (Exception ex)
{
Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

说明：

- 完整代码参考GitHub。
- .NET SDK支持的Http Header：Cache-Control 、 Content-Disposition 、Content-Encoding 、 Expires、Content-Type等 。
- 它们的相关介绍见 RFC2616 。

带自定义Header

OSS支持用户自定义元信息来对文件进行描述。比如：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

try
{
using (var fs = File.Open(fileToUpload, FileMode.Open))
{
var metadata = new ObjectMetadata();
metadata.UserMetadata.Add("name", "my-data");
metadata.ContentLength = fs.Length;
client.PutObject(bucketName, key, fs, metadata);
Console.WriteLine("Put object succeeded");
```

```
}
}
catch (Exception ex)
{
Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

说明：

- 完整代码参考GitHub。
- 在上面代码中，用户自定义了一个名字为“name”，值为“my-data”的元信息。
- 当用户下载此文件的时候，此元信息也可以一并得到。
- 一个文件可以有多个类似的参数，但所有的user meta总大小不能超过2KB。
- 使用上述方法上传最大文件不能超过5G。如果超过可以使用MutipartUpload上传。
- user meta的名称大小写不敏感，比如用户上传文件时，定义名字为“Name”的meta,在表头中存储的参数为：“x-oss-meta-name”,所以读取时读取名字为“name”的参数即可。
- 但如果存入参数为“name”，读取时使用“Name”读取不到对应信息，会返回“null”。

创建模拟文件夹

OSS服务是没有文件夹这个概念的，所有元素都是以文件来存储。但给用户提供了创建模拟文件夹的方式，如下代码：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

try
{
// 重要: 这时候作为目录的key必须以斜线（ / ）结尾
const string key = "yourfolder/";
// 此时的目录是一个内容为空的文件
using (var stream = new MemoryStream())
{
client.PutObject(bucketName, key, stream);
Console.WriteLine("Create dir {0} succeeded", key);
}
}
catch (Exception ex)
{
Console.WriteLine("Create dir failed, {0}", ex.Message);
}
```

说明：

- 完整代码参考GitHub。
- 创建模拟文件夹本质上来说是创建了一个空文件。

- 对于这个文件照样可以上传下载,只是控制台会对以”/“结尾的文件以文件夹的方式展示。
- 所以用户可以使用上述方式来实现创建模拟文件夹。
- 而对文件夹的访问可以参看文件夹功能模拟

异步上传

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

static AutoResetEvent _event = new AutoResetEvent(false);

public static void AsyncPutObject()
{
try
{
using (var fs = File.Open(fileToUpload, FileMode.Open))
{
var metadata = new ObjectMetadata();
metadata.CacheControl = "No-Cache";
metadata.ContentType = "text/html";
client.BeginPutObject(bucketName, key, fs, metadata, PutObjectCallback, new string('a', 8));

_event.WaitOne();
}
}
catch (Exception ex)
{
Console.WriteLine("Put object failed, {0}", ex.Message);
}
}

private static void PutObjectCallback(IAsyncResult ar)
{
try
{
var result = client.EndPutObject(ar);
Console.WriteLine("ETag:{0}", result.ETag);
Console.WriteLine("User Parameter:{0}", ar.AsyncState as string);
Console.WriteLine("Put object succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Put object failed, {0}", ex.Message);
}
finally
{
_event.Set();
}
}
```

说明：

- 完整代码参考GitHub。
- 使用异步上传时您需要实现自己的回调处理函数。

追加上传

追加写的方式上传文件。详细介绍请参考API。

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 追加内容到指定的OSS文件中。
/// </summary>
/// <param name="bucketName">指定的存储空间名称。</param>
/// <param name="key">OSS上会被追加内容的文件名称。</param>
/// <param name="fileToUpload">指定被追加的文件的路径。</param>
public static void AppendObject(string bucketName, string key, string fileToUpload)
{
//第一次追加文件的时候，文件可能已经存在，先获取文件的当前长度，如果不存在，position为0
long position = 0;
try
{
var metadata = client.GetObjectMetadata(bucketName, key);
position = metadata.ContentLength;
}
catch(Exception) {}

try
{
using (var fs = File.Open(fileToUpload, FileMode.Open))
{
var request = new AppendObjectRequest(bucketName, key)
{
ObjectMetadata = new ObjectMetadata(),
Content = fs,
Position = position
};

var result = client.AppendObject(request);
// 设置下次追加文件时的position位置
position = result.NextAppendPosition;

Console.WriteLine("Append object succeeded, next append position:{0}", position);
}

// 再次追加文件，这时候的position值可以从上次的结果中获取到
using (var fs = File.Open(fileToUpload, FileMode.Open))
{
```

```
var request = new AppendObjectRequest(bucketName, key)
{
ObjectMetadata = new ObjectMetadata(),
Content = fs,
Position = position
};

var result = client.AppendObject(request);
position = result.NextAppendPosition;

Console.WriteLine("Append object succeeded, next append position:{0}", position);
}
}
catch (Exception ex)
{
Console.WriteLine("Append object failed, {0}", ex.Message);
}
}
```

说明：

- 完整代码参考GitHub。
- 注意事项请参考OSS AppendObject API。

分片上传

除了通过PutObject接口上传文件到OSS以外，OSS还提供了另外一种上传模式 —— Multipart Upload。

用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload上传模式，如：

- 需要支持断点上传。
- 上传超过100MB大小的文件。
- 网络条件较差，和OSS的服务器之间的链接经常断开。
- 上传文件之前，无法确定上传文件的大小。

下面我们将一步步学习怎样实现Multipart Upload。

分步完成Multipart Upload

初始化Multipart Upload

我们使用 initiateMultipartUpload 方法来初始化一个分片上传事件：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

try
```

```
{
string bucketName = "your-bucket-name";
string key = "your-key";

// 开始Multipart Upload
var request = new InitiateMultipartUploadRequest(bucketName, key);
var result = client.InitiateMultipartUpload(request);

// 打印UploadId
Console.WriteLine("Init multi part upload succeeded");
Console.WriteLine("Upload Id:{0}", result.UploadId);
}
catch (Exception ex)
{
Console.WriteLine("Init multi part upload failed, {0}", ex.Message);
}
```

说明：

- 完整代码参考GitHub。
- 我们用InitiateMultipartUploadRequest来指定上传文件的名字和所属存储空间（Bucket）。
- 在InitiateMultipartUploadRequest中，您也可以设置ObjectMeta，但是不必指定其中的ContentLength。
- initiateMultipartUpload 的返回结果中含有UploadId ，它是区分分片上传事件的唯一标识，在后面的操作中，我们将用到它。

Upload Part本地上传

我们把本地文件分片上传。假设有一个文件，本地路径为 /path/to/file.zip 由于文件比较大，我们将其分片传输到OSS中。

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

// 计算片个数
var fi = new FileInfo(fileToUpload);
var fileSize = fi.Length;
var partCount = fileSize / partSize;
if (fileSize % partSize != 0)
{
partCount++;
}

// 开始分片上传
try
{
var partETags = new List<PartETag>();
using (var fs = File.Open(fileToUpload, FileMode.Open))
{
```

```
for (var i = 0; i < partCount; i++)
{
var skipBytes = (long)partSize * i;

//定位到本次上传片应该开始的位置
fs.Seek(skipBytes, 0);

//计算本次上传的片大小，最后一片为剩余的数据大小，其余片都是part size大小。
var size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
var request = new UploadPartRequest(bucketName, objectName, uploadId)
{
InputStream = fs,
PartSize = size,
PartNumber = i + 1
};

//调用UploadPart接口执行上传功能，返回结果中包含了这个数据片的ETag值
var result = _ossClient.UploadPart(request);
partETags.Add(result.PartETag);
}
Console.WriteLine("Put multi part upload succeeded");
}
}
catch (Exception ex)
{
Console.WriteLine("Put multi part upload failed, {0}", ex.Message);
}
```

说明： 完整代码参考GitHub。

上面程序的核心是调用UploadPart方法来上传每一个分片，但是要注意以下几点：

- UploadPart 方法要求除最后一个Part以外，其他的Part大小都要大于100KB。但是Upload Part接口并不会立即校验上传Part的大小（因为不知道是否为最后一片）；只有当Complete Multipart Upload的时候才会校验。
- OSS会将服务器端收到Part数据的MD5值放在ETag头内返回给用户。
- 为了保证数据在网络传输过程中不出现错误，SDK会自动设置Content-MD5，OSS会计算上传数据的MD5值与SDK计算的MD5值比较，如果不一致返回InvalidDigest错误码。
- Part号码的范围是1~10000。如果超出这个范围，OSS将返回InvalidArgument的错误码。
- 每次上传part时都要把流定位到此次上传片开头所对应的位置。
- 每次上传part之后，OSS的返回结果会包含一个 PartETag 对象，他是上传片的ETag与片编号（PartNumber）的组合，在后续完成分片上传的步骤中会用到它，因此我们需要将其保存起来。一般来讲我们将这些 PartETag 对象保存到List中。

完成分片上传

完成分片上传代码如下：

```
using Aliyun.OSS;
```

```
// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

try
{
var completeMultipartUploadRequest = new CompleteMultipartUploadRequest(bucketName, key, uploadId);
foreach (var partETag in partETags)
{
completeMultipartUploadRequest.PartETags.Add(partETag);
}
var result = client.CompleteMultipartUpload(completeMultipartUploadRequest);
Console.WriteLine("complete multi part succeeded");
}
catch (Exception ex)
{
Console.WriteLine("complete multi part failed, {0}", ex.Message);
}
```

说明：

- 完整代码参考GitHub。
- 上面代码中的 partETags 就是进行分片上传中保存的partETag的列表，OSS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。
- 当所有的数据Part验证通过后，OSS会将这些part组合成一个完整的文件。

取消分片上传事件

我们可以用 AbortMultipartUpload 方法取消分片上传。

```
 using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

try
{
var request = new AbortMultipartUpload(bucketName, key, uploadId);
client.AbortMultipartUpload(request);
Console.WriteLine("Abort multi part succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Abort multi part failed, {0}", ex.Message);
}
```

说明： 完整代码参考请GitHub。

获取存储空间内所有分片上传事件

我们可以用 ListMultipartUploads 方法获取存储空间内所有上传事件。

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

try
{
// 获取Bucket内所有上传事件
var request = new ListMultipartUploadsRequest(bucketName);
var multipartUploadListing = client.ListMultipartUploads(request);
Console.WriteLine("List multi part succeeded");

// 获取各事件信息
var multipartUploads = multipartUploadListing.MultipartUploads;
foreach (var mu : multipartUploads)
{
Console.WriteLine("Key:{0}, UploadId:{1}", mu.Key , mu.UploadId);
}

var commonPrefixes = multipartUploadListing.CommonPrefixes;
foreach (var prefix : commonPrefixes)
{
Console.WriteLine("Prefix:{0}", prefix);
}
}
catch (Exception ex)
{
Console.WriteLine("List multi part uploads failed, {0}", ex.Message);
}
```

说明：

- 完整代码请参考GitHub。
- 默认情况下，如果存储空间中的分片上传事件的数量大于1000，则只会返回1000个文件， 且返回结果中 IsTruncated 为 false，返回 NextKeyMarker 和 NextUploadIdMarker 作为下此读取的起点。
- 若想增大返回分片上传事件数目，可以修改 MaxUploads 参数，或者使用 KeyMarker 以及 UploadIdMarker 参数分次读取。

获取所有已上传的片信息

我们可以用 ListParts 方法获取某个上传事件所有已上传的片。

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

try
{
```

```
var listPartsRequest = new ListPartsRequest(bucketName, key, uploadId);
var listPartsResult = client.ListParts(listPartsRequest);

Console.WriteLine("List parts succeeded");

// 遍历所有Part
var parts = listPartsResult.Parts;
foreach (var part : parts)
{
Console.WriteLine("partNumber:{0}, ETag:{1}, Size:{2}", part.PartNumber, part.ETag, part.Size);
}
}
catch (Exception ex)
{
Console.WriteLine("List parts failed, {0}", ex.Message);
}
```

说明：

- 完整代码参考GitHub
- 默认情况下，如果存储空间中的分片上传事件的数量大于1000，则只会返回1000个Multipart Upload信息，且返回结果中 IsTruncated 为false，并返回 NextPartNumberMarker作为下此读取的起点。
- 若想增大返回分片上传事件数目，可以修改 MaxParts 参数，或者使用 PartNumberMarker 参数分次读取。

通过断点续传上传

除了支持分片上传外，还提供了断点续传功能，如果某次上传中断，下次可以从上次失败的位置开始上传，以便加快速度。

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 断点续传
/// </summary>
/// <param name="bucketName">指定的存储空间名称。</param>
/// <param name="key">保存到OSS上的名称。</param>
/// <param name="fileToUpload">指定上传文件的路径。</param>
/// <param name="checkpointDir">保存断点续传中间状态文件的目录，如果指定了，则会具有断点续传功能，否则会重新上传</param>
public static void ResumableUploadObject(string bucketName, string key, string fileToUpload, string checkpointDir)
{
try
{
client.ResumableUploadObject(bucketName, key, fileToUpload, null, checkpointDir);
```

```
Console.WriteLine("Resumable upload object:{0} succeeded", key);
}
catch (Exception ex)
{
Console.WriteLine("Resumable upload object failed, {0}", ex.Message);
}
}
```

说明：

- 完整代码参考GitHub。
- checkpointDir目录中会保存断点续传的中间状态，用于失败后，下次继续上传时使用。
- 如果checkpointDir为null，断点续传功能不会生效，每次都会重新上传。

上传回调

OSS在上传文件完成的时，可以提供回调（Callback）给应用服务器。用户只需要在发送给OSS的请求中携带相应的回调参数，即能实现回调。支持上传回调接口有：PutObject、PostObject、CompleteMultipartUpload。您想了解上传回调的更详细信息，请参考上传回调。下面以 PutObject 为例说明上传回调的用法。

```
using Aliyun.OSS;

var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

public static void PutObjectCallback(string bucketName)
{
const string key = "PutObjectProgress";
const string callbackUrl = "http://callback.oss-demo.com:23450";
const string callbackBody =
"bucket=${bucket}&object=${object}&etag=${etag}&size=${size}&mimeType=${mimeType}&" +
"my_var1=${x:var1}&my_var2=${x:var2}";

try
{
string responseContent = "";
var metadata = BuildCallbackMetadata(callbackUrl, callbackBody);
using (var fs = File.Open(fileToUpload, FileMode.Open))
{
var putObjectRequest = new PutObjectRequest(bucketName, key, fs, metadata);
var result = client.PutObject(putObjectRequest);
responseContent = GetCallbackResponse(result);
}
Console.WriteLine("Put object:{0} succeeded, callback response content:{1}", key, responseContent);
}
catch (OssException ex)
{
Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
```

```
catch (Exception ex)
{
Console.WriteLine("Failed with error info: {0}", ex.Message);
}
}

private static string GetCallbackResponse(PutObjectResult putObjectResult)
{
string callbackResponse = null;
using (var stream = putObjectResult.ResponseStream)
{
var buffer = new byte[4 * 1024];
var bytesRead = stream.Read(buffer, 0, buffer.Length);
callbackResponse = Encoding.Default.GetString(buffer, 0, bytesRead);
}
return callbackResponse;
}
```

说明：

- PutObject、CompleteMultipartUpload、ResumableUploadObject支持上传回调，完整代码请参考GitHub。
- 上传回调的调试方法和错误排除，请参考上传回调错误及排除。

进度条

OSS .Net SDK支持进度条功能，指示上传、下载的进度。下面的代码以 PutObject 为例，说明进度条功能的使用方法。

```
using Aliyun.OSS;

var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

public static void PutObjectProgress(string bucketName)
{
const string key = "PutObjectProgress";
try
{
using (var fs = File.Open(fileToUpload, FileMode.Open))
{
var putObjectRequest = new PutObjectRequest(bucketName, key, fs);
putObjectRequest.StreamTransferProgress += streamProgressCallback;
client.PutObject(putObjectRequest);
}
Console.WriteLine("Put object:{0} succeeded", key);
}
catch (OssException ex)
{
Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
```

```
catch (Exception ex)
{
Console.WriteLine("Failed with error info: {0}", ex.Message);
}
}

private static void streamProgressCallback(object sender, StreamTransferProgressArgs args)
{
System.Console.WriteLine("ProgressCallback - TotalBytes:{0}, TransferredBytes:{1}",
args.TotalBytes, args.TransferredBytes);
}
```

说明： PutObject、AppendObject、UploadPart、ResumableUploadObject、GetObject都支持进度功能，完整代码请参考GitHub。

下载文件

简单的下载文件

我们可以通过以下代码将文件读取到一个流中：

```
// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 从指定的OSS存储空间中获取指定的文件
/// </summary>
/// <param name="bucketName">要获取的文件所在的存储空间的名称</param>
/// <param name="key">要获取的文件的名称</param>
/// <param name="fileToDownload">文件保存的本地路径</param>
public void GetObject(string bucketName, string key, string fileToDownload)
{
try
{
var object = client.GetObject(bucketName, key);
using (var requestStream = object.Content)
{
byte[] buf = new byte[1024];
var fs = File.Open(fileToDownload, FileMode.OpenOrCreate);
var len = 0;
while ((len = requestStream.Read(buf, 0, 1024)) != 0)
{
fs.Write(buf, 0, len);
}
fs.Close();
}
```

```
Console.WriteLine("Get object succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Get object failed. {0}", ex.Message);
}
}
```

提示：

- 完整代码参考：GitHub
- OssObject 包含了文件的各种信息，包含文件所在的存储空间（Bucket）、文件的名称、Metadata以及一个输入流。
- 我们可以通过操作输入流将文件的内容读取到文件或者内存中。而ObjectMeta包含了文件上传时定义的，ETag，Http Header以及自定义的元信息。

分段读取文件

为了实现更多的功能，我们可以通过使用 GetObjectRequest 来读取文件，比如分段读取：

```
// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

public void GetObject(string bucketName, string key, string fileToDownload)
{
try
{
var getObjectRequest = new GetObjectRequest(bucketName, key);

//读取文件中的20到100个字符，包括第20和第100个字符
getObjectRequest.SetRange(20, 100);

var object = client.GetObject(getObjectRequest);

// 将读到的数据写到fielToDownload文件中去
using (var requestStream = object.Content)
{
byte[] buf = new byte[1024];
var fs = File.Open(fileToDownload, FileMode.OpenOrCreate);
var len = 0;
while ((len = requestStream.Read(buf, 0, 1024)) != 0)
{
fs.Write(buf, 0, len);
}
fs.Close();
}
Console.WriteLine("Get object succeeded");
}
catch (Exception ex)
{
```

```
Console.WriteLine("Get object failed. {0}", ex.Message);
}
}
```

提示：

- 完整代码参考：GitHub
- 我们通过GetObjectRequest的setRange方法设置了返回的文件的范围。
- 我们可以用此功能实现文件的分段下载和断点续传。
- GetObjectRequest可以设置以下参数：

参数	说明
Range	指定文件传输的范围。
ModifiedSinceConstraint	如果指定的时间早于实际修改时间，则正常传送文件。否则抛出304 Not Modified异常。
UnmodifiedSinceConstraint	如果传入参数中的时间等于或者晚于文件实际修改时间，则正常传输文件。否则抛出412 precondition failed异常
MatchingETagConstraints	传入一组ETag，如果传入期望的ETag和文件的ETag匹配，则正常传输文件。否则抛出412 precondition failed异常
NonmatchingEtagConstraints	传入一组ETag，如果传入的ETag值和文件的ETag不匹配，则正常传输文件。否则抛出304 Not Modified异常。
ResponseHeaderOverrides	自定义OSS返回请求中的一些Header。

只获取文件元信息

通过GetObjectMetadata方法可以只获取ObjectMeta而不获取文件的实体。代码如下：

```
// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 获取文件的元信息。
/// </summary>
/// <param name="bucketName">存储空间的名称</param>
/// <param name="key">文件在OSS上的名称</param>
public void GetObjectMetadata(string bucketName, string key)
{
try
{
var metadata = client.GetObjectMetadata(bucketName, key);

Console.WriteLine("Get object meta succeeded");
Console.WriteLine("Content-Type:{0}", metadata.ContentType);
```

```
Console.WriteLine("Cache-Control:{0}", metadata.CacheControl);
}
catch (Exception ex)
{
Console.WriteLine("Get object meta failed. {0}", ex.Message);
}
}
```

进度条

OSS .Net SDK支持进度条功能，指示上传、下载的进度。下面的代码以 GetObject 为例，说明进度条功能的使用方法。

```
using Aliyun.OSS;

var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

public static void GetObjectProgress(string bucketName)
{
const string key = "GetObjectProgress";
try
{
client.PutObject(bucketName, key, fileToUpload);

var getObjectRequest = new GetObjectRequest(bucketName, key);
getObjectRequest.StreamTransferProgress += streamProgressCallback;
var ossObject = client.GetObject(getObjectRequest);
using (var stream = ossObject.Content)
{
var buffer = new byte[1024 * 1024];
var bytesRead = 0;
while ((bytesRead = stream.Read(buffer, 0, buffer.Length)) > 0)
{
// 处理读出的数据
}
}

Console.WriteLine("Get object:{0} succeeded", key);
}
catch (OssException ex)
{
Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
Console.WriteLine("Failed with error info: {0}", ex.Message);
}
}

private static void streamProgressCallback(object sender, StreamTransferProgressArgs args)
{
System.Console.WriteLine("ProgressCallback - TotalBytes:{0}, TransferredBytes:{1}",
```

```
args.TotalBytes, args.TransferredBytes);
}
```

提示：

- PutObject、AppendObject、UploadPart、ResumableUploadObject、GetObject都支持进度功能，完整代码请参考：GitHub。

管理文件

在OSS中，用户可以通过一系列的接口管理存储空间（Bucket）中的文件（Object），比如ListObjects、DeleteObject、CopyObject、DoesObjectExist等。

列出存储空间中的文件

简单列出文件

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 列出指定存储空间下的文件的摘要信息OssObjectSummary列表
/// </summary>
/// <param name="bucketName">存储空间的名称</param>
public void ListObjects(string bucketName)
{
try
{
var listObjectsRequest = new ListObjectsRequest(bucketName);
var result = client.ListObjects(listObjectsRequest);

Console.WriteLine("List objects succeeded");
foreach (var summary in result.ObjectSummaries)
{
Console.WriteLine("File name:{0}", summary.Key);
}
}
catch (Exception ex)
{
Console.WriteLine("List objects failed. {0}", ex.Message);
}
}
```

说明：

- 完整代码参考GitHub。
- 默认情况下，如果存储空间中的文件数量大于100，则只会返回100个文件， 且返回结果中 IsTruncated 为 true，并返回 NextMarker 作为下此读取的起点。
- 若想增大返回文件数目，可以修改MaxKeys参数，或者使用Marker参数分次读取。

带前缀过滤的列出文件

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 列出指定存储空间下其Key以prefix为前缀的文件的摘要信息OssObjectSummary
/// </summary>
/// <param name="bucketName">存储空间的名称</param>
/// <param name="prefix">限定返回的文件必须以此作为前缀</param>
public void ListObjects(string bucketName, string prefix)
{
try
{
var listObjectsRequest = new ListObjectsRequest(bucketName)
{
Prefix = prefix
};

var result = client.ListObjects(listObjectsRequest);

Console.WriteLine("List objects succeeded");
foreach (var summary in result.ObjectSummaries)
{
Console.WriteLine("File Name:{0}", summary.Key);
}
}
catch (Exception ex)
{
Console.WriteLine("List objects failed. {0}", ex.Message);
}
}
```

通过异步方式列出文件

```
using Aliyun.OSS;

// 初始化OssClient
static OssClient ossClient = new OssClient(endpoint, accessKeyId, accessKeySecret);
static AutoResetEvent _event = new AutoResetEvent(false);
```

```
public static void AsyncListObjects()
{
try
{
var listObjectsRequest = new ListObjectsRequest(bucketName);
ossClient.BeginListObjects(listObjectsRequest, ListObjectCallback, null);

_event.WaitOne();
}
catch (Exception ex)
{
Console.WriteLine("List objects failed. {0}", ex.Message);
}
}

private static void ListObjectCallback(IAsyncResult ar)
{
try
{
var result = ossClient.EndListObjects(ar);

foreach (var summary in result.ObjectSummaries)
{
Console.WriteLine("文件名称:{0}", summary.Key);
}

_event.Set();
Console.WriteLine("List objects succeeded");
}
catch (Exception ex)
{
Console.WriteLine("List objects failed. {0}", ex.Message);
}
}
```

说明：

- 完整代码参考GitHub。
- 上面的ListObjectCallback方法就是异步调用结束后执行的回调方法，如果使用异步类型的接口，都需要实现类似接口。

通过ListObjectsRequest列出文件

我们可以通过设置ListObjectsRequest的参数来完成更强大的功能。比如：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 列出指定存储空间下的文件的摘要信息OssObjectSummary列表
```

```
/// </summary>
/// <param name="bucketName">存储空间的名称</param>
public void ListObjects(string bucketName)
{
try
{
var listObjectsRequest = new ListObjectsRequest(bucketName)
{
Delimiter = "/",
Marker = "abc"
};

result = client.ListObjects(listObjectsRequest);

Console.WriteLine("List objects succeeded");
foreach (var summary in result.ObjectSummaries)
{
Console.WriteLine("文件名称:{0}", summary.Key);
}
}
catch (Exception ex)
{
Console.WriteLine("List objects failed. {0}", ex.Message);
}
}
```

可以设置的参数名称和作用如下：

名称	作用
Delimiter	用于对文件名字进行分组的字符。所有名字包含指定的前缀且第一次出现Delimiter字符之间的文件作为一组元素: CommonPrefixes。
Marker	设定结果从Marker之后按字母排序的第一个开始返回。
MaxKeys	限定此次返回文件的最大数，如果不设定，默认为100，MaxKeys取值不能大于1000。
Prefix	限定返回的文件名称必须以Prefix作为前缀。注意使用Prefix查询时，返回的文件名中仍会包含Prefix。

说明：

- 完整代码参考GitHub。
- 如果需要遍历所有的文件，而文件数量大于100，则需要进行多次迭代。每次迭代时，将上次迭代列取最后一个文件的key作为本次迭代中的Marker即可。

文件夹功能模拟

我们还可以通过 Delimiter 和 Prefix 参数的配合模拟出文件夹功能。Delimiter 和 Prefix 的组合效果是这样的

：

- 如果把 Prefix 设为某个文件夹名，就可以罗列以此 Prefix 开头的文件，即该文件夹下递归的所有的文件和子文件夹。
- 如果再把 Delimiter 设置为“/”时，返回值就只罗列该文件夹下的文件，该文件夹下的子文件夹返回在 CommonPrefixes 部分，子文件夹下递归的文件和文件夹不被显示。假设Bucket中有4个文件：oss.jpg、fun/test.jpg、fun/movie/001.avi、fun/movie/007.avi ，我们把“/”符号作为文件夹的分隔符。

列出存储空间内所有文件

当我们需要获取存储空间下的所有文件时，可以这样写：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

public void ListObject(string bucketName)
{
try
{
ObjectListing result = null;
string nextMarker = string.Empty;
do
{
var listObjectsRequest = new ListObjectsRequest(bucketName)
{
Marker = nextMarker,
MaxKeys = 100
};
result = client.ListObjects(listObjectsRequest);

Console.WriteLine("File:");
foreach (var summary in result.ObjectSummaries)
{
Console.WriteLine("Name:{0}", summary.Key);
}
nextMarker = result.NextMarker;
} while (result.IsTruncated);
}
catch (Exception ex)
{
Console.WriteLine("List object failed. {0}", ex.Message);
}
}
```

输出：

```
File:
Name:fun/movie/001.avi
Name:fun/movie/007.avi
```

```
Name:fun/test.jpg
Name:oss.jpg
```

递归列出目录下所有文件

我们可以通过设置 Prefix 参数来获取某个目录下所有的文件：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

public void ListObject(string bucketName)
{
try
{
var listObjectsRequest = new ListObjectsRequest(bucketName)
{
Prefix = "fun/"
};
result = client.ListObjects(listObjectsRequest);

Console.WriteLine("List object succeeded");
Console.WriteLine("File:");
foreach (var summary in result.ObjectSummaries)
{
Console.WriteLine("Name:{0}", summary.Key);
}

Console.WriteLine("Dir:");
foreach (var prefix in result.CommonPrefixes)
{
Console.WriteLine("Name:{0}", prefix);
}
}
catch (Exception ex)
{
Console.WriteLine("List object failed. {0}", ex.Message);
}
}
```

输出:

```
List object succeeded
File:
Name:fun/movie/001.avi
Name:fun/movie/007.avi
Name:fun/test.jpg

目录:
```

列出目录下的文件和子目录

在 Prefix 和 Delimiter 结合的情况下，可以列出目录下的文件和子目录：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

public void ListObjects(string bucketName)
{
try
{
var listObjectsRequest = new ListObjectsRequest(bucketName)
{
Prefix = "fun/",
Delimiter = "/"
};

result = client.ListObjects(listObjectsRequest);

Console.WriteLine("List object succeeded");
Console.WriteLine("File:");
foreach (var summary in result.ObjectSummaries)
{
Console.WriteLine("Name:{0}", summary.Key);
}

Console.WriteLine("Dir:");
foreach (var prefix in result.CommonPrefixes)
{
Console.WriteLine("Name:{0}", prefix);
}
}
catch (Exception ex)
{
Console.WriteLine("List object failed. {0}", ex.Message);
}
}
```

输出：

```
List object success
File:
Name:fun/test.jpg

Dir:
Name:fun/movie/
```

说明：

- 返回的结果中， ObjectSummaries 的列表中给出的是fun目录下的文件。
- 而 CommonPrefixs 的列表中给出的是fun目录下的所有子文件夹。可以看出 **fun/movie/001.avi** 和 **fun/movie/007.avi** 两个文件并没有被列出来，因为它们属于fun文件夹

下的movie目录。

删除文件

删除一个文件

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 列出指定存储空间下的特定文件
/// </summary>
/// <param name="bucketName">存储空间的名称</param>
/// <param name="key">文件的名称</param>
public void DeleteObject(string bucketName, string key)
{
try
{
client.DeleteObject(bucketName, key);
Console.WriteLine("Delete object succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Delete object failed. {0}", ex.Message);
}
}
```

说明： 完整代码参考GitHub。

删除多个文件

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 删除指定存储空间中的所有文件
/// </summary>
/// <param name="bucketName">存储空间的名称</param>
public void DeleteObjects(string bucketName)
{
try
{
var keys = new List<string>();
var listResult = client.ListObjects(bucketName);
```

```
foreach (var summary in listResult.ObjectSummaries)
{
keys.Add(summary.Key);
}

var request = new DeleteObjectsRequest(bucketName, keys, false);
client.DeleteObjects(request);
Console.WriteLine("Delete objects succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Delete objects failed. {0}", ex.Message);
}
}
```

说明： 完整代码参考GitHub。

拷贝文件

在同一个区域（杭州，深圳，青岛等）中，用户可以对有操作权限的文件进行复制操作。

拷贝一个文件

通过 copyObject 方法我们可以拷贝一个文件，代码如下：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 拷贝文件
/// </summary>
/// <param name="sourceBucket">原文件所在存储空间的名称</param>
/// <param name="sourceKey">原文件的名称</param>
/// <param name="targetBucket">目标文件所在存储空间的名称</param>
/// <param name="targetKey">目标文件的名称</param>
public void CopyObect(string sourceBucket, string sourceKey, string targetBucket, string targetKey)
{
try
{
var metadata = new ObjectMetadata();
metadata.AddHeader(Util.HttpHeaders.ContentType, "text/html");
var req = new CopyObjectRequest(sourceBucket, sourceKey, targetBucket, targetKey)
{
NewObjectMetadata = metadata
};
var ret = client.CopyObject(req);

Console.WriteLine("Copy object succeeded");
Console.WriteLine("文件的ETag:{0}", ret.ETag);
```

```
}
catch (Exception ex)
{
Console.WriteLine("Copy object failed. {0}", ex.Message);
}
}
```

说明：

- 完整代码参考GitHub。
- 使用该方法拷贝的文件必须小于1G，否则会报错。若文件大于1G，使用下面的Upload Part Copy。

拷贝大文件

初始化Multipart Upload

我们使用 initiateMultipartUpload 方法来初始化一个分片上传事件：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

public void InitiateMultipartUpload(string bucketName, string key)
{
try
{
// 开始Multipart Upload
var request = new InitiateMultipartUploadRequest(bucketName, key);
var result = client.InitiateMultipartUpload(request);

// 打印UploadId
Console.WriteLine("Init multipart upload succeeded");
Console.WriteLine("Upload Id:{0}", result.UploadId);
}
catch (Exception ex)
{
Console.WriteLine("Init multipart upload failed. {0}", ex.Message);
}
}
```

说明： 完整代码参考GitHub。

Upload Part Copy拷贝上传

Upload Part Copy 通过从一个已经存在文件的中拷贝数据来上传一个新文件。当拷贝一个大于500MB的文件，建议使用Upload Part Copy的方式来进行拷贝。

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

public void UploadPartCopy(string sourceBucket, string sourceKey, string targetBucket, string targetKey, string
uploadId)
{
try
{
// 计算总共的分片个数
var metadata = client.GetObjectMetadata(sourceBucket, sourceKey);
var fileSize = metadata.ContentLength;
var partCount = (int)fileSize / partSize;
if (fileSize % partSize != 0)
{
partCount++;
}
// 开始分片拷贝
var partETags = new List<PartETag>();
for (var i = 0; i < partCount; i++)
{
var skipBytes = (long)partSize * i;
var size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
var request = new UploadPartCopyRequest(targetBucket, targetKey, sourceBucket, sourceKey, uploadId)
{
PartSize = size,
PartNumber = i + 1,
BeginIndex = skipBytes
};

var result = client.UploadPartCopy(request);
partETags.Add(result.PartETag);
}

Console.WriteLine("Upload part copy succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Upload part copy failed. {0}", ex.Message);
}
```

说明：

- 完整代码参考GitHub。
- 以上程序调用uploadPartCopy方法来拷贝每一个分片。
- 与UploadPart要求基本一致，需要通过BeginIndex来定位到此次上传片开头所对应的位置，同时需要通过SourceKey来指定拷贝的文件。

完成分片上传

完成分片上传代码如下：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

public void CompleteMultipartUpload(string bucketName, string key, string uploadId)
{
try
{
var completeMultipartUploadRequest = new CompleteMultipartUploadRequest(bucketName, key, uploadId);
foreach (var partETag in partETags)
{
completeMultipartUploadRequest.PartETags.Add(partETag);
}
var result = client.CompleteMultipartUpload(completeMultipartUploadRequest);

Console.WriteLine("Complete multipart upload succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Complete multipart upload failed. {0}", ex.Message);
}
}
```

说明：

- 完整代码参考GitHub。
- 上面代码中的 partETags 就是进行分片上传中保存的partETag的列表，OSS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。
- 当所有的数据Part验证通过后，OSS会将这些part组合成一个完整的文件。

通过断点续传拷贝

除了支持分片拷贝外，还提供了断点续传功能，如果某次拷贝中断，下次可以从上次失败的位置开始拷贝，以便加快速度。

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

public static void ResumableCopyObject(string sourceBucketName, string sourceKey,
string destBucketName, string destKey)
{
string checkpointDir = @"<your checkpoint dir>";
try
{
var request = new CopyObjectRequest(sourceBucketName, sourceKey, destBucketName, destKey);
client.ResumableCopyObject(request, checkpointDir);

Console.WriteLine("Resumable copy new object:{0} succeeded", request.DestinationKey);
```

```
}
catch (Exception ex)
{
Console.WriteLine("Resumable copy new object failed, {0}", ex.Message);
}
}
```

说明：

- 完整代码参考GitHub。
- checkpointDir目录中会保存断点续传的中间状态，用于失败后，下次继续拷贝时使用。
- 如果checkpointDir为null，断点续传功能不会生效，每次都会重新拷贝。

修改文件Meta

可以通过ModifyObjectMeta操作来实现修改已有文件的 meta 信息

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 修改文件的meta值
/// </summary>
/// <param name="bucketName">文件所在存储空间的名称</param>
/// <param name="key">文件的名称</param>
public void ModifyObjectMeta(string bucketName, string key)
{
try
{
var meta = new ObjectMetadata();
meta.ContentType = "application/octet-stream";

client.ModifyObjectMeta(bucketName, key, meta);

Console.WriteLine("Modify object meta succeeded");
}
catch (Exception ex)
{
Console.WriteLine("Modify object meta failed. {0}", ex.Message);
}
}
```

说明： 完整代码参考GitHub。

设置文件权限

文件权限的详细说明请参看权限控制，您可以通过SetBucketAcl设置文件权限。

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 设置文件权限
/// </summary>
/// <param name="bucketName">文件所在存储空间的名称</param>
/// <param name="key">文件的名称</param>
public void SetObjectAclSample(string bucketName, string key)
{
try
{
client.SetObjectAcl(bucketName, key, CannedAccessControlList.PublicRead);
Console.WriteLine("Set Object:{0} Acl succeeded ", key);
}
catch (Exception ex)
{
Console.WriteLine("Failed with error info: {0}", ex.Message);
}
}
```

说明：

- 文件的四种访问权限：private、public-read、public-read-write、default分别对应于CannedAccessControlList.Private、CannedAccessControlList.PublicRead、CannedAccessControlList.PublicReadWrite、CannedAccessControlList.Default。
- 完整代码参考GitHub。

读取文件权限

您可以通过GetBucketAcl读取文件权限。

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

/// <summary>
/// 读取文件权限
/// </summary>
/// <param name="bucketName">文件所在存储空间的名称</param>
/// <param name="key">文件的名称</param>
public void SetObjectAclSample(string bucketName, string key)
{
try
{
var result = client.GetObjectAcl(bucketName, key);
Console.WriteLine("Get Object Acl succeeded,Id:{0} Acl:{1} succeeded",
```

```
result.Owner.Id, result.ACL.ToString());
}
catch (Exception ex)
{
Console.WriteLine("Failed with error info: {0}", ex.Message);
}
}
```

说明： 完整代码参考GitHub。

授权访问

使用URL签名授权访问

生成签名URL

通过生成签名URL的形式提供给用户一个临时的访问URL。在生成URL时，您可以指定URL过期的时间，从而限制用户长时间访问。

生成一个签名的URL

代码如下：

```
var req = new GeneratePresignedUriRequest(bucketName, key, SignHttpMethod.Get)
{
Expiration = DateTime.Now.AddHours(1)
};
var uri = client.GeneratePresignedUri(req);
```

生成的URL默认以GET方式访问，这样，用户可以直接通过浏览器访问相关内容。

生成其他Http方法的URL

如果您想允许用户临时进行其他操作（比如上传，删除文件），可能需要签名其他方法的URL，如下：

```
// 生成PUT方法的URL
var req = new GeneratePresignedUriRequest(bucketName, key, SignHttpMethod.Put)
{
Expiration = DateTime.Now.AddHours(1),
ContentType = "text/html"
};
```

```
var uri = client.GeneratePresignedUri(req);
```

使用签名URL发送请求

现在.Net SDK支持Put Object和Get Object两种方式的URL签名请求。

使用URL签名的方式Put Object

```
var generatePresignedUriRequest = new GeneratePresignedUriRequest(bucketName, key, SignHttpMethod.Put);
var signedUrl = client.GeneratePresignedUri(generatePresignedUriRequest);
var result = client.PutObject(signedUrl, fileToUpload);
```

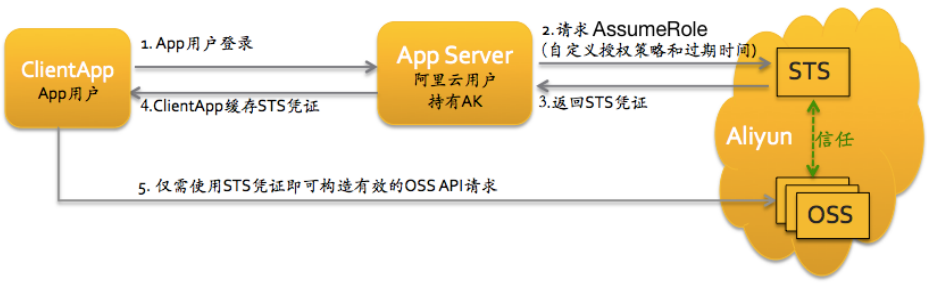
使用STS服务临时授权

介绍

OSS可以通过阿里云STS服务，临时进行授权访问。阿里云STS (Security Token Service) 是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或联邦用户（用户身份由您自己管理）颁发一个自定义时效和权限的访问凭证。第三方应用或联邦用户可以使用该访问凭证直接调用阿里云产品API，或者使用阿里云产品提供的SDK来访问云产品API。

- 您不需要透露您的长期密钥(AccessKey)给第三方应用，只需要生成一个访问令牌并将令牌交给第三方应用即可。这个令牌的访问权限及有效期限都可以由您自定义。
- 您不需要关心权限撤销问题，访问令牌过期后就自动失效。

以APP应用为例，交互流程如下图：



方案的详细描述如下：

1. App用户登录。App用户身份是客户自己管理。客户可以自定义身份管理系统，也可以使用外部Web账号或OpenID。对于每个有效的App用户来说，AppServer是可以确切地定义出每个App用户的最小访问权限。
2. AppServer请求STS服务获取一个安全令牌(SecurityToken)。在调用STS之前，AppServer需要确定App用户的最小访问权限（用Policy语法描述）以及授权的过期时间。然后通过调用STS的AssumeRole(扮演角色)接口来获取安全令牌。角色管理与使用相关内容请参考RAM使用指南中的角色管理。

3. STS返回给AppServer一个有效的访问凭证，包括一个安全令牌(SecurityToken)、临时访问密钥(AccessKeyId, AccessKeySecret)以及过期时间。
4. AppServer将访问凭证返回给ClientApp。ClientApp可以缓存这个凭证。当凭证失效时，ClientApp需要向AppServer申请新的有效访问凭证。比如，访问凭证有效期为1小时，那么ClientApp可以每30分钟向AppServer请求更新访问凭证。
5. ClientApp使用本地缓存的访问凭证去请求Aliyun Service API。云服务会感知STS访问凭证，并会依赖STS服务来验证访问凭证，并正确响应用户请求。

STS安全令牌详情，请参考《RAM使用指南》中的角色管理。关键是调用STS服务接口AssumeRole来获取有效访问凭证即可。也可以直接使用STS SDK来调用该方法。

使用STS凭证构造签名请求

用户的client端拿到STS临时凭证后，通过其中安全令牌(SecurityToken)以及临时访问密钥(AccessKeyId, AccessKeySecret)生成OssClient。以上传文件为例：

```
string accessKeyId = "<accessKeyId>";
string accessKeySecret = "<accessKeySecret>";
string securityToken = "<securityToken>"

// 以杭州为例
string endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

var ossClient = new OssClient(endpoint, accessKeyId, accessKeySecret, securityToken);
```

生命周期管理

OSS提供文件生命周期管理来为用户管理对象。用户可以为某个存储空间定义生命周期配置，来为该存储空间的文件定义各种规则。 目前，用户可以通过规则来删除相匹配的文件。每条规则都由如下几个部分组成：

- 文件名称前缀，只有匹配该前缀的文件才适用这个规则
- 操作，用户希望对匹配的文件所执行的操作。
- 日期或天数，用户期望在特定日期或者是在文件最后修改时间后多少天执行指定的操作。

设置生命周期

生命周期的配置规则由一段xml表示。

```
<LifecycleConfiguration>
  <Rule>
    <ID>delete obsoleted files</ID>
```

```
    <Prefix>obsoleted/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>3</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>delete temporary files</ID>
    <Prefix>temporary/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Date>2022-10-12T00:00:00.000Z</Date>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

一个生命周期的Config里面可以包含多个Rule（最多1000个）。

各字段解释：

- ID字段是用来唯一表示本条规则。
- Prefix指定对存储空间下的符合特定前缀的文件使用规则，不能重叠。
- Status指定本条规则的状态，只有Enabled和Disabled，分别表示启用规则和禁用规则。
- Expiration节点里面的Days表示大于文件最后修改时间指定的天数就删除文件，Date则表示到指定的绝对时间之后就删除文件(绝对时间服从ISO8601的格式)。

可以通过下面的代码，设置上述生命周期规则。

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

var setBucketLifecycleRequest = new SetBucketLifecycleRequest(bucketName);

// 创建第一条规则
LifecycleRule lcr1 = new LifecycleRule()
{
    ID = "delete obsoleted files",
    Prefix = "obsoleted/",
    Status = RuleStatus.Enabled,
    ExpriationDays = 3
};

//创建第二条规则
//当ExpirationTime指定使用Date时，请注意此时的含义是：从这个时刻开始一直生效
LifecycleRule lcr2 = new LifecycleRule()
{
     ID = "delete temporary files",
     Prefix = "temporary/",
     Status = RuleStatus.Enabled,
     ExpirationTime = DateTime.Parse("2022-10-12T00:00:00.000Z")
};
setBucketLifecycleRequest.AddLifecycleRule(lcr1);
```

```
setBucketLifecycleRequest.AddLifecycleRule(lcr2);

client.SetBucketLifecycle(setBucketLifecycleRequest);
```

提示：

- 完整代码参考：GitHub

注意

- 上面的规则一中ExpriationDays使用了Days，表示3天之后一直生效
- 上面的规则二中ExpriationDays使用了Date，表示2022-10-12T00:00:00.000Z之后一直生效。除非明确清楚使用Date时的含义，否则请慎重使用。

获取生命周期

可以通过下面的代码获取上述生命周期规则。

```
using Aliyun.OSS;

var rules = client.GetBucketLifecycle(bucketName);
foreach (var rule in rules)
{
    Console.WriteLine("ID: {0}", rule.ID);
    Console.WriteLine("Prefix: {0}", rule.Prefix);
    Console.WriteLine("Status: {0}", rule.Status);

    if (rule.ExpriationDays.HasValue)
        Console.WriteLine("ExpirationDays: {0}", rule.ExpriationDays);
    if (rule.ExpirationTime.HasValue)
        Console.WriteLine("ExpirationTime: {0}", FormatIso8601Date(rule.ExpirationTime.Value));
}
```

提示：

- 完整代码参考：GitHub

清空生命周期

可以通过下面的代码清空存储空间中生命周期规则。

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

var LifecycleRequest = new SetBucketLifecycleRequest(bucketName);
```

```
client.SetBucketLifecycle(LifecycleRequest);
```

设置访问日志

访问日志简介

用户可以通过设置Bucket的访问日志配置，把对该Bucket的访问日志保存在指定的Bucket中，以供后续的分析。访问日志以文件的形式存在于指定的Bucket中，每小时会生成一个文本文件。文件名的格式为：

```
<TargetPrefix><SourceBucket>-YYYY-mm-DD-HH-MM-SS-UniqueString
```

其中TargetPrefix由用户在配置中指定。

日志配置由如下部分组成：

- TargetBucket：目标Bucket名，生成的日志文件会保存到这个Bucket中。
- TargetPrefix：日志文件名前缀，可以为空。

更多关于访问日志文件名格式，日志格式请参考 Bucket访问日志。

开启日志功能

下面的代码开启日志功能，且把日志保存在 TargetBucket 中，日志文件名前缀为 logging- ：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

// 开启访问日志
var request = new SetBucketLoggingRequest(bucketName, targetBucketName, "logging-");
client.SetBucketLogging(request);
```

提示：

- bucketName 和 targetBucketName 可以为相同Bucket
- 完整代码请参考：GitHub

查看日志设置

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

var result = client.GetBucketLogging(bucketName);

Console.WriteLine("Get bucket:{0} Logging, prefix:{1}, target bucket:{2}",
bucketName, result.TargetPrefix, result.TargetBucket);
```

提示：

- 完整代码请参考：GitHub

关闭日志功能

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

client.DeleteBucketLogging(bucketName);
```

提示：

- 日志功能关闭后，已经生成的日志文件不会删除
- 完整代码请参考：GitHub

跨域资源共享

跨域资源共享(CORS)允许web端的应用程序访问不属于本域的资源。OSS提供接口方便开发者控制跨域访问的权限。

设定CORS规则

通过setBucketCors 方法将指定的存储空间上设定一个跨域资源共享CORS的规则，如果原规则存在则覆盖原规则。具体的规则主要通过CORSRule类来进行参数设置。代码如下：

```
using Aliyun.OSS;
```

```
// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

var req = new SetBucketCorsRequest(bucketName);
var r1 = new CORSRule();

//指定允许跨域请求的来源
r1.AddAllowedOrigin("http://www.a.com");

//指定允许的跨域请求方法(GET/PUT/DELETE/POST/HEAD)
r1.AddAllowedMethod("POST");

//控制在OPTIONS预取指令中Access-Control-Request-Headers头中指定的header是否允许。
r1.AddAllowedHeader("*");

//指定允许用户从应用程序中访问的响应头
r1.AddExposeHeader("x-oss-test");
req.AddCORSRule(r1);
client.SetBucketCors(req);
```

提示：

- 完整代码参考：GitHub

注意：

- 每个存储空间最多只能使用10条规则。
- AllowedOrigins和AllowedMethods都能够最多支持一个”*”通配符。”*”表示对于所有的域来源或者操作都满足。
- 而AllowedHeaders和ExposeHeaders不支持通配符。

获取CORS规则

我们可以参考存储空间的CORS规则，通过GetBucketCors方法。代码如下：

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

var rules = client.GetBucketCors(bucketName);

foreach (var rule in rules)
{
Console.WriteLine("AllowedOrigins:{0}", rule.AllowedOrigins);
Console.WriteLine("AllowedMethods:{0}", rule.AllowedMethods);
Console.WriteLine("AllowedHeaders:{0}", rule.AllowedHeaders);
Console.WriteLine("ExposeHeaders:{0}", rule.ExposeHeaders);
Console.WriteLine("MaxAgeSeconds:{0}", rule.MaxAgeSeconds);
}
```

提示：

- 完整代码参考：GitHub

删除CORS规则

用于关闭指定存储空间对应的CORS并清空所有规则。

```
using Aliyun.OSS;

// 初始化OssClient
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

// 清空存储空间的CORS规则
client.DeleteBucketCors(bucketName);
```

提示：

- 完整代码参考：GitHub

防盗链

OSS是按使用收费的服务，为了防止用户在OSS上的数据被其他人盗链，OSS支持基于HTTP header中表头字段referer的防盗链方法。

设置Referer白名单

通过下面代码设置Referer白名单：

```
using Aliyun.OSS;

var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

var refererList = new List<string>();
// 添加referer项
refererList.Add("http://www.aliyun.com");
refererList.Add("http://www.*.com");
refererList.Add("http://www.?.aliyuncs.com");

// 允许referer字段为空，并设置存储空间Referer列表
var request = new SetBucketRefererRequest(bucketName, refererList);
```

```
request. AllowEmptyReferer = true;

client.setBucketReferer(bucketName, br);

Console.WriteLine("设置存储空间{0}的referer白名单成功", bucketName);
```

提示：

- 完整代码参考：GitHub

注意：

- Referer参数支持通配符“*”和“？”，更多详细的规则配置可以参考开发人员指南OSS防盗链

获取Referer白名单

```
using Aliyun.OSS;

var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

var rc = client.GetBucketReferer(bucketName);
Console.WriteLine("allow ? " + (rc.AllowEmptyReferer ? "yes" : "no"));

if (rc.RefererList.Referers != null)
{
for (var i = 0; i < rc.RefererList.Referers.Length; i++)
Console.WriteLine(rc.RefererList.Referers[i]);
}
else
{
Console.WriteLine("Empty Referer List");
}
```

提示：

- 完整代码参考：GitHub

清空Referer白名单

Referer白名单不能直接清空，只能通过重新设置来覆盖之前的规则。

```
using Aliyun.OSS;

var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

// 默认允许referer字段为空，且referer白名单为空。
var request = new SetBucketRefererRequest(bucketName);
```

```
client.SetBucketReferer(request);
Console.WriteLine("清空存储空间{0}的referer白名单成功", bucketName);
```

图片处理

OSS图片处理，是OSS对外提供的海量、安全、低成本、高可靠的图片处理服务。用户将原始图片上传保存到OSS，通过简单的 RESTful 接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

图片处理提供图片处理接口，图片上传请使用上传接口。基于OSS图片处理，用户可以搭建自己的图片处理服务。

图片处理基础功能

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放、裁剪、旋转
- 图片效果
- 图片添加图片、文字、图文混合水印
- 自定义图片处理样式，在控制台的 图片处理 -> 样式管理 中定义
- 通过级联处理调用多个图片处理功能

图片处理使用

图片处理使用标准的 HTTP GET 请求来访问，所有的处理参数是编码在 URL 中的QueyString。

匿名访问

如果图片文件（Object）的访问权限是 公共读 ，如下表所示的权限，则可以匿名访问图片服务。

Bucket权限	Object权限
公共读私有写（public-read）或 公共读写（public-read-write）	默认（default）
任意权限	公共读私有写（public-read）或 公共读写（public-read-write）

通过如下格式的三级域名匿名访问图片处理：

```
http://bucket.<endpoint>/object?x-oss-process=image/action,parame_value
```

- bucket：用户的存储空间（bucket）名称
- endpoint：用户存储空间所在数据中心的访问域名
- object：用户上传在OSS上的图片文件
- image：图片处理保留标志符
- action：用户对图片做的操作，如缩放、裁剪、旋转等
- parame：用户对图片做的操作所对应的参数

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

自定义样式，使用如下格式的三级域名匿名访问图片处理：

```
http://bucket.<endpoint>/object?x-oss-process=x-oss-process=style/name
```

- style：用户自定义样式系统保留标志符
- name：自定义样式名称，即控制台定义样式的 规则名

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=style/oss-pic-style-w-100
```

通过级联处理，可以对一张图片顺序实施多个操作，格式如下：

```
http://bucket.<endpoint>/object?x-oss-process=image/action,parame_value/action,parame_value/...
```

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100/rotate,90
```

图片服务也支持HTTPS访问，例如：

```
https://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

授权访问

对私有权限的文件（Object），如下表所示的权限，必须通过授权才能访问图片服务。

Bucket权限	Object权限
私有读写（private）	默认权限（default）
任意权限	私有读写（private）

生成带签名的图片处理的URL代码如下：

```
public static void GenerateIamgeUri(string bucketName)
{
try
{

var process = "image/resize,m_fixed,w_100,h_100";
var req = new GeneratePresignedUriRequest(bucketName, key, SignHttpMethod.Get)
{
Expiration = DateTime.Now.AddSeconds(30 * 60),
Process = process
};

// 产生带有签名的URI
var uri = client.GeneratePresignedUri(req);

Console.WriteLine("Generate Presigned Uri:{0} with process:{1} succeeded ", uri, process);
}
catch (Exception ex)
{
Console.WriteLine("Failed with error info: {0}", ex.Message);
}
}
```

提示：

- 授权访问支持 自定义样式、 HTTPS、 级联处理
- GeneratePresignedUri 过期时间单位是**秒**
- 完整的代码请参看 GitHub

SDK访问

对于任意权限的图片文件，都可以直接使用 SDK 访问图片、进行处理。

提示：

- 图片处理的完整代码请参考：GitHub
- SDK处理图片文件支持 自定义样式、 HTTPS、 级联处理

基础操作

图片处理的基础操作包括，获取图片信息、格式转换、缩放、裁剪、旋转、效果、水印等。

```
public static void ImageProcess(string bucketName)
{
try
{
```

```
client.PutObject(bucketName, key, imageFile);

// 图片缩放
var process = "image/resize,m_fixed,w_100,h_100";
var ossObject = client.GetObject(new GetObjectRequest(bucketName, key, process));

WriteToFile(dirToDownload + "/sample-resize.jpg", ossObject.Content);

// 图片裁剪
process = "image/crop,w_100,h_100,x_100,y_100,r_1";
ossObject = client.GetObject(new GetObjectRequest(bucketName, key, process));

WriteToFile(dirToDownload + "/sample-crop.jpg", ossObject.Content);

// 图片旋转
process = "image/rotate,90";
ossObject = client.GetObject(new GetObjectRequest(bucketName, key, process));

WriteToFile(dirToDownload + "/sample-rotate.jpg", ossObject.Content);

// 图片锐化
process = "image/sharpen,100";
ossObject = client.GetObject(new GetObjectRequest(bucketName, key, process));

WriteToFile(dirToDownload + "/sample-sharpen.jpg", ossObject.Content);

// 图片加水印
process = "image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ";
ossObject = client.GetObject(new GetObjectRequest(bucketName, key, process));

WriteToFile(dirToDownload + "/sample-watermark.jpg", ossObject.Content);

// 图片格式转换
process = "image/format,png";
ossObject = client.GetObject(new GetObjectRequest(bucketName, key, process));

WriteToFile(dirToDownload + "/sample-format.png", ossObject.Content);

// 图片信息
process = "image/info";
ossObject = client.GetObject(new GetObjectRequest(bucketName, key, process));

WriteToFile(dirToDownload + "/sample-info.txt", ossObject.Content);

Console.WriteLine("Get Object:{0} with process:{1} succeeded ", key, process);
}
catch (OssException ex)
{
Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
Console.WriteLine("Failed with error info: {0}", ex.Message);
}
}
```

自定义样式

```
public static void ImageProcess(string bucketName)
{
try
{
client.PutObject(bucketName, key, imageFile);

// 自定义样式
var process = "style/oss-pic-style-w-100";

var ossObject = client.GetObject(new GetObjectRequest(bucketName, key, process));

WriteToFile(dirToDownload + "/sample-style.jpg", ossObject.Content);
}
catch (OssException ex)
{
Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
Console.WriteLine("Failed with error info: {0}", ex.Message);
}
}
```

级联处理

```
public static void ImageProcess(string bucketName)
{
try
{
client.PutObject(bucketName, key, imageFile);

// 级联处理
var process = "image/resize,m_fixed,w_100,h_100/rotate,90";

var ossObject = client.GetObject(new GetObjectRequest(bucketName, key, process));

WriteToFile(dirToDownload + "/sample-style.jpg", ossObject.Content);
}
catch (OssException ex)
{
Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
Console.WriteLine("Failed with error info: {0}", ex.Message);
}
}
```

图片处理工具

- 可视化图片处理工具 ImageStyleViever ，可以直观的看到OSS图片处理的结果
- OSS图片处理的功能、使用演示 页面

异常

OSS .NET SDK 中有两种异常 ClientException 以及 OSSException ， 他们都继承自或者间接继承自 RuntimeException。

ClientException

ClientException指SDK内部出现的异常，比如未设置BucketName，网络无法到达等等。

OSSException

OSSException指服务器端错误，它来自于对服务器错误信息的解析。OSSException一般有以下几个成员：

- Code： OSS返回给用户的错误码。
- Message： OSS给出的详细错误信息。
- RequestId： 用于唯一标识该次请求的UUID；当您无法解决问题时，可以凭这个RequestId来请求OSS开发工程师的帮助。
- HostId： 用于标识访问的OSS集群（目前统一为oss.aliyuncs.com）

下面是OSS中常见的异常：

错误码	描述
AccessDenied	拒绝访问
BucketAlreadyExists	Bucket已经存在
BucketNotEmpty	Bucket不为空
EntityTooLarge	实体过大
EntityTooSmall	实体过小
FileGroupTooLarge	文件组过大
FilePartNotExist	文件Part不存在
FilePartStale	文件Part过时
InvalidArgument	参数格式错误

InvalidAccessKeyId	AccessKeyId不存在
InvalidBucketName	无效的Bucket名字
InvalidDigest	无效的摘要
InvalidObjectName	无效的Object名字
InvalidPart	无效的Part
InvalidPartOrder	无效的part顺序
InvalidTargetBucketForLogging	Logging操作中有无效的目标bucket
InternalError	OSS内部发生错误
MalformedXML	XML格式非法
MethodNotAllowed	不支持的方法
MissingArgument	缺少参数
MissingContentLength	缺少内容长度
NoSuchBucket	Bucket不存在
NoSuchKey	文件不存在
NoSuchUpload	Multipart Upload ID不存在
NotImplemented	无法处理的方法
PreconditionFailed	预处理错误
RequestTimeTooSkewed	发起请求的时间和服务器时间超出15分钟
RequestTimeout	请求超时
SignatureDoesNotMatch	签名错误
TooManyBuckets	用户的Bucket数目超过限制

Node-JS-SDK

安装

SDK安装

目前官网文档中的demo都是基于SDK 6.X，版本低于6.X的可参考 5.X开发文档, 升级6.X请移步升级文档

- Github地址
- API文档
- ChangeLog

要求

开通阿里云OSS服务，如果您还没不了解阿里云OSS服务，请登录OSS产品主页了解。

创建AccessKeyId和AccessKeySecret，由于云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践。如果部署在服务端可以使用RAM子账号或STS来进行API访问或日常运维管控操作，如果部署在客户端请使用STS方式来进行API访问。详情请参见访问控制。

环境要求

OSS NodeJS SDK基于Node.js环境构建。

使用方式

OSS NodeJS SDK同时支持同步和异步的使用方式，参考这篇文章：

- 同步方式：基于async和await方式, 异步编程同步化
- 异步方式：类似callback的方式，API接口返回Promise，使用.then()处理返回结果，使用.catch()处理错误

无论同步方式还是异步方式中，均使用new OSS()创建client。

下面分别举例，先上传一个文件，然后立即下载这个文件：

同步方式

```
let client = new OSS(...);

async function put () {
try {
let r1 = await.put('object','tmp/file');
console.log('put success: %j', r1);
let r2 = await client.get('object');
console.log('get success: %j', r2);
} catch(e) {
console.error('error: %j', err);
}
}

put();
```

异步方式

```
let client = new OSS(...);

client.put('object', '/tmp/file').then(function (r1) {
console.log('put success: %j', r1);
return client.get('object');
}).then(function (r2) {
console.log('get success: %j', r2);
}).catch(function (err) {
console.error('error: %j', err);
});
```

安装

使用Node.js

支持的Node.js版本：

Node.js >= 8.0.0
如果需要在 Node.js < 8 的环境中使用，请使用 ali-oss 4.x版本。

首先使用npm安装SDK的开发包：

```
npm install ali-oss
```

然后在你的程序中使用：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<oss region>',
//云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践，部署在服务端使用RAM子账号或STS，部署在客户端使用STS。
accessKeyId: '<Your accessKeyId>',
accessKeySecret: '<Your accessKeySecret>',
bucket: '<Your bucket name>'
});
```

如果使用npm遇到网络问题，可以使用淘宝提供的npm镜像：cnpm

配置项

OSS(options)介绍

- [accessKeyId] {String} 通过阿里云控制台创建的access key。
- [accessKeySecret] {String} 通过阿里云控制台创建的access secret。
- [bucket] {String} 通过控制台创建的bucket, 或通过putBucket创建。
- [endpoint] {String} oss 域名。
- [region] {String} bucket 所在的区域, 默认 oss-cn-hangzhou。
- [internal] {Boolean} 是否使用阿里云内部网访问，比如采用ecs访问oss，设置true, 采用internal的 endpoint 会节约费用, 默认false。
- [secure] {Boolean} (secure: true) 使用 HTTPS , (secure: false) 则使用 HTTP, 细节请看。
- [timeout] {String|Number} 超时时间, 默认 60s。

example:

```
var oss = require('ali-oss');

var store = oss({
accessKeyId: 'your access key',
accessKeySecret: 'your access secret',
bucket: 'your bucket name',
region: 'oss-cn-hangzhou'
});
```

快速开始-NodeJS

本文介绍如何在Node.js环境中快速使用OSS服务，包括查看Bucket列表、查看文件列表、上传/下载文件和删除文件。为了方便修改，本文会新建一个app.js，以下功能演示代码都写在这个文件中。

安装SDK

在工作目录安装ali-oss：

```
npm install ali-oss
```

使用同步方式

由于SDK基于ES6开发，采用async/await能够异步编程同步化。

使用异步方式

为了支持callback的使用方式，SDK同时也提供了异步的基于Promise的接口，使用上类似callback，具体可参考这篇博客。

下面的文档将以**同步的方式**为例。

初始化Client

创建一个文件：app.js并写入下面的内容：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>'
});
```

其中region参数是指您申请OSS服务时的区域，例如oss-cn-hangzhou。完整的区域列表可以在OSS服务节点查看。

如果所使用的endpoint不在上述列表中，可以通过以下参数指定endpoint：

- internal: 配合region使用，如果指定internal为true，则访问内网节点
- secure: 配合region使用，如果指定了secure为true，则使用HTTPS访问
- endpoint: 例如http://oss-cn-hangzhou.aliyuncs.com，如果指定了endpoint，则region会被忽略，endpoint可以指定HTTPS，也可以是IP形式
- cname: 配合endpoint使用，如果指定了cname为true，则将endpoint视为用户绑定的自定义域名
- bucket: 如果未指定bucket，则进行Object相关的操作时需要先调用useBucket接口（只需要调用一次）
- timeout: 默认为60秒，指定访问OSS的API的超时时间

查看Bucket列表

在app.js末尾添加如下内容，使用listBuckets接口查看Bucket列表：

```
async function listBuckets () {
try {
let result = await client.listBuckets();
} catch(err) {
console.log(err)
}
}

listBucket();
```

运行并查看结果：node app.js。

查看文件列表

修改app.js，使用list接口查看文件列表：

```
client.useBucket('Your bucket name');

async function list () {
try {
let result = client.list({
'max-keys': 5
})
console.log(result)
} catch (err) {
consol.log (err)
}
}

list();
```

使用node app.js运行并查看结果。

上传一个文件

修改app.js，使用put接口上传一个文件：

```
client.useBucket('Your bucket name');

async function put () {
try {
let result = await client.put('object-key', 'local file');
console.log(result);
} catch (err) {
consol.log (err);
}
}

put();
```

下载一个文件

修改app.js，使用get接口下载一个文件：

```
async function get () {
try {
let result = await client.get('object-key', 'local file');
```

```
console.log(result);
} catch (err) {
consol.log (err);
}
}

get();
```

删除一个文件

修改app.js，使用delete接口删除一个文件：

```
async function delete () {
try {
let result = await client.delete('object-key');
console.log(result);
} catch (err) {
console.log (err);
}
}

delete();
```

了解更多

- 管理Bucket
- 上传文件
- 下载文件
- 管理文件
- 自定义域名绑定
- 使用STS访问
- 设置访问权限
- 管理生命周期
- 设置访问日志
- 静态网站托管
- 设置防盗链
- 异常

管理Bucket

存储空间（Bucket）是OSS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

查看所有Bucket

使用listBuckets接口列出当前用户下的所有Bucket，用户还可以指定prefix参数，列出Bucket名字为特定前缀的所有Bucket：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>'
});

async function listBuckets() {
try {
const result = await client.listBuckets();
console.log(result);
const result2 = await client.listBuckets({
prefix: 'prefix',
});
console.log(result);
} catch (err) {
console.log(err);
}
}

listBuckets();
```

创建Bucket

使用putBucket接口创建一个Bucket，用户需要指定Bucket的名字：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>'
});

// putBucket
async function putBucket() {
try {
const result = await client.putBucket('your bucket name');
console.log(result);
} catch (err) {
console.log(err);
}
}

putBucket();
```

注意：

- Bucket的命名规范请查看OSS 基本概念。
- 由于存储空间的名字是全局唯一的，所以必须保证您的Bucket名字不与别人的重复。

删除Bucket

使用deleteBucket接口删除一个Bucket，用户需要指定Bucket的名字：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>'
});

async function deleteBucket() {
try {
const result = await client.deleteBucket('your bucket name');
console.log(result);
} catch (err) {
console.log(err);
}
}

deleteBucket();
```

注意：

- 如果该Bucket下还有文件存在，则需要先删除所有文件才能删除Bucket。
- 如果该Bucket下还有未完成的上传请求，则需要通过listUploads和 abortMultipartUpload先取消请求才能删除Bucket。

Bucket访问权限

用户可以设置Bucket的访问权限，允许或者禁止匿名用户对其内容进行读写。更多关于访问权限的内容请参考访问权限。

获取Bucket的访问权限（ACL）

通过getBucketACL查看Bucket的ACL：

```
let OSS = require('ali-oss');
```

```
let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>'
});

async function getBucketACL() {
try {
const result = await client.getBucketACL('luozhang002');
console.log(result);
} catch (err) {
console.log(err);
}
}

getBucketACL();
```

设置Bucket的访问权限（ACL）

通过putBucketACL设置Bucket的ACL：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>'
});

async function putBucketACL() {
try {
const result = await client.putBucketACL('bucket name', 'public-read');
console.log(result);
} catch (err) {
console.log(err);
}
}

putBucketACL();
```

上传文件

用户可以通过以下方式向OSS中上传文件：

> 以下示例代码中的catch语法，请自行学习下es6 promise, 和 三方co库。仔细阅读下sdk的使用方式，传送门。

- 上传本地文件
- 流式上传
- 上传Buffer内容
- 分片上传
- 断点上传

上传本地文件

通过put接口来上传一个本地文件到OSS：

```
var co = require('co');
var OSS = require('ali-oss')

var client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

co(function* () {
var result = yield client.put('object-key', 'local-file');
console.log(result);
}).catch(function (err) {
console.log(err);
});
```

流式上传

通过putStream接口来上传一个Stream中的内容，stream参数可以是任何实现了Readable Stream的对象，包含文件流，网络流等。当使用putStream接口时，SDK默认会发起一个chunked encoding的HTTP PUT请求。如果在options指定了contentLength参数，则不会使用chunked encoding。

```
var co = require('co');
var OSS = require('ali-oss');
var fs = require('fs');

var client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

co(function* () {
// use 'chunked encoding'
var stream = fs.createReadStream('local-file');
var result = yield client.putStream('object-key', stream);
```

```
console.log(result);

// don't use 'chunked encoding'
var stream = fs.createReadStream('local-file');
var size = fs.statSync('local-file').size;
var result = yield client.putStream(
'object-key', stream, {contentLength: size});
console.log(result);
}).catch(function (err) {
console.log(err);
});
```

上传Buffer内容

用户也可以通过put接口简单地将Buffer中的内容上传到OSS：

```
var co = require('co');
var OSS = require('ali-oss');

var client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

co(function* () {
var result = yield client.put('object-key', new Buffer('hello world'));
console.log(result);
}).catch(function (err) {
console.log(err);
});
```

分片上传

在需要上传的文件较大时，可以通过multipartUpload接口进行分片上传。分片上传的好处是将一个大请求分成多个小请求来执行，这样当其中一些请求失败后，不需要重新上传整个文件，而只需要上传失败的分片就可以了。一般对于大于100MB的文件，建议采用分片上传的方法。

相关参数：

- name {String} object 名称
- file {String|File} file path or HTML5 Web File
- [options] {Object} 额外参数
    - [checkpoint] {Object} 断点记录点，可以进行断点续传, 如果设置这个参数，上传会从断点开始，如果没有设置，就会重新上传.
    - [partSize] {Number} 分片大小
    - [progress] {Function} thunk 或 generator形式, 回调函数包含三个参数

        - (percentage {Number} 进度百分比(0-1之间小数)
        - checkpoint {Object} 断点记录点
        - res {Object}) 单次part成功返回的response
    - [meta] {Object} 用户自定义header meta信息, header前缀 x-oss-meta-
    - [headers] {Object} extra headers, detail see RFC 2616
        - ‘Cache-Control’ 通用消息头被用于在http 请求和响应中通过指定指令来实现缓存机制, e.g.: Cache-Control: public, no-cache
        - ‘Content-Disposition’ 指示回复的内容该以何种形式展示，是以内联的形式（即网页或者页面的一部分），还是以附件的形式下载并保存到本地, e.g.: Content-Disposition: somename
        - ‘Content-Encoding’ 用于对特定媒体类型的数据进行压缩, e.g.: Content-Encoding: gzip
        - ‘Expires’ 过期时间, e.g.: Expires: 3600000

```
var co = require('co');
var OSS = require('ali-oss')

var client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

co(function* () {
var result = yield client.multipartUpload('object-key', 'local-file', {
progress: function* (p) {
console.log('Progress: ' + p);
}
meta: {
year: 2017,
people: 'test'
}
});
console.log(result);
var head = yield client.head('object-key');
console.log(head);
}).catch(function (err) {
console.log(err);
});
```

上面的progress参数是一个进度回调函数，用于获取上传进度。progress可以是一个generator function(function*)，也可以是一个”thunk” ：

```
var progress = function (p) {
return function (done) {
console.log(p);
done();
};
};
```

上面的meta参数是一个用户自定义的元数据，通过head接口可以获取到object的meta数据。

断点上传

分片上传提供progress参数允许用户传递一个进度回调，在回调中SDK将当前已经上传成功的比例和断点信息作为参数。为了实现断点上传，可以在上传过程中保存断点信息（checkpoint），发生错误后，再将已保存的checkpoint作为参数传递给multipartUpload，此时将从上次失败的地方继续上传。

```
var co = require('co');
var OSS = require('ali-oss')

var client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

co(function* () {
var checkpoint;
// retry 5 times
for (var i = 0; i < 5; i++) {
var result = yield client.multipartUpload('object-key', 'local-file', {
checkpoint: checkpoint,
progress: function* (percentage, cpt) {
checkpoint = cpt;
}
});
console.log(result);
break; // break if success
} catch (err) {
console.log(err);
}
}
}).catch(function (err) {
console.log(err);
});
```

上面的代码只是将checkpoint保存在变量中，如果程序崩溃的话就丢失了，用户也可以将它保存在文件中，然后在程序重启后将checkpoint信息从文件中读取出来。

下载文件

用户可以通过以下方式从OSS中下载文件：

- 下载到本地文件

- 流式下载
- 下载到Buffer
- HTTP下载（浏览器下载）

下载到本地文件

通过get接口来下载Object到一个本地文件：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function get () {
try {
let result = await client.get('object-key', 'local-file');
console.log(result);
} catch (e) {
console.log(e);
}
}

get();
```

流式下载

使用getStream来下载文件时，返回一个Readable Stream，用户可以流式地处理文件内容。

```
let OSS = require('ali-oss');
let fs = require('fs');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function getStream () {
try {
let result = await client.getStream('object-key');
console.log(result);
let writeStream = fs.createWriteStream('local-file');
result.stream.pipe(writeStream);
} catch (e) {
console.log(e);
```

```
}
}

getStream()
```

下载Buffer

用户也可以通过get接口简单地将文件内容下载到Buffer中：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function getBuffer () {
try {
let result = awati client.get('object-key');
console.log(result.content);
} catch (e) {
console.log(e);
}
}

getBuffer();
```

HTTP下载

对于存放在OSS中的文件，在不用SDK的情况下用户也可以直接使用HTTP下载，这包括直接使用浏览器下载，或者使用wget, curl等命令行工具下载。这时文件的URL需要由SDK生成。使用signatureUrl方法生成可下载的HTTP地址，URL的有效时间默认为半个小时：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

let url = client.signatureUrl('object-key');
console.log(url);

let url = client.signatureUrl('object-key', {expires: 3600});
console.log(url);
```

```
// signed URL for PUT
let url = client.signatureUrl('object-key', {method: 'PUT'});
console.log(url);
```

管理文件

一个Bucket下可能有非常多的文件，SDK提供一系列的接口方便用户管理文件。

查看所有文件

通过list来列出当前Bucket下的所有文件。主要的参数如下：

- prefix 指定只列出符合特定前缀的文件
- marker 指定只列出文件名大于marker之后的文件
- delimiter 用于获取文件的公共前缀
- max-keys 用于指定最多返回的文件个数

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function list () {
try {
// 不带任何参数，默认最多返回1000个文件
let result = await client.list();
console.log(result);

// 根据nextMarker继续列出文件
if (result.isTruncated) {
let result = await client.list({
marker: result.nextMarker
});
}

// 列出前缀为'my-'的文件
let result = await client.list({
prefix: 'my-'
});
console.log(result);
```

```
// 列出前缀为'my-'且在'my-object'之后的文件
let result = await client.list({
prefix: 'my-',
marker: 'my-object'
});
console.log(result);
} catch (e) {
console.log(e);
}
}

list();
```

模拟目录结构

OSS是基于对象的存储服务，没有目录的概念。存储在一个Bucket中所有文件都是通过文件的key唯一标识，并没有层级的结构。这种结构可以让OSS的存储非常高效，但是用户管理文件时希望能够像传统的文件系统一样把文件分门别类放到不同的“目录”下面。通过OSS提供的“公共前缀”的功能，也可以很方便地模拟目录结构。公共前缀的概念请参考 列出Object

假设Bucket中已有如下文件：

```
foo/x
foo/y
foo/bar/a
foo/bar/b
foo/hello/C/1
foo/hello/C/2
...
foo/hello/C/9999
```

接下来我们实现一个函数叫listDir，列出指定目录下的文件和子目录：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function listDir(dir)
let result = await client.list({
prefix: dir,
delimiter: '/'
});

result.prefixes.forEach(function (subDir) {
console.log('SubDir: %s', subDir);
});
result.objects.forEach(function (obj) {
```

```
console.log(Object: %s', obj.name);
});
end
```

运行结果如下：

```
> await listDir('foo/')
=> SubDir: foo/bar/
SubDir: foo/hello/
Object: foo/x
Object: foo/y

> await listDir('foo/bar/')
=> Object: foo/bar/a
Object: foo/bar/b

> await listDir('foo/hello/C/')
=> Object: foo/hello/C/1
Object: foo/hello/C/2
...
Object: foo/hello/C/9999
```

文件元信息

向OSS上传文件时，除了文件内容，还可以指定文件的一些属性信息，称为“元信息”。这些信息在上传时与文件一起存储，在下载时与文件一起返回。

因为文件元信息在上传/下载时是附在HTTP Headers中， HTTP协议规定不能包含复杂字符。因此元信息**只能是简单的ASCII可见字符，不能包含换行。** 所有元信息的总大小不能超过8KB。

使用put，putStream和multipartUpload时都可以通过指定meta参数来指定文件的元信息：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function put () {
try {
let result = await client.put('object-key', 'local-file', {
meta: {
year: 2016,
people: 'mary'
}
});
console.log(result);
} catch (e) {
```

```
console.log(e);
}
}

put();
```

通过putMeta接口来更新文件元信息：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function putMeta () {
try {
let result = await client.putMeta('object-key', {
meta: {
year: 2015,
people: 'mary'
}
});
console.log(result);
} catch (e) {
console.log(e);
}
}

putMeta();
```

拷贝文件

使用copy拷贝一个文件。拷贝可以发生在下面两种情况：

- 同一个Bucket
- 两个不同Bucket，但是它们在同一个region，此时的源Object名字应为’/bucket/object’的形式

另外，拷贝时对文件元信息的处理有两种选择：

- 如果没有指定meta参数，则与源文件相同，即拷贝源文件的元信息
- 如果指定了meta参数，则使用新的元信息覆盖源文件的信息

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
```

```
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function copy () {
try {
// 两个Bucket之间拷贝
let result = await client.copy('to', '/from-bucket/from');
console.log(result);

// 拷贝元信息
let result = await client.copy('to', 'from');
console.log(result);

// 覆盖元信息
let result = await client.copy('to', 'from', {
meta: {
year: 2015,
people: 'mary'
}
});
console.log(result);
} catch (e) {
console.log(e);
}
}
```

删除文件

通过delete来删除某个文件：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function delete () {
try {
let result = yield client.delete('object-key');
console.log(result);
} catch (e) {
console.log(e);
}
}

delete();
```

批量删除文件

通过deleteMulti来删除一批文件，用户可以通过quiet参数来指定是否返回删除的结果：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function deleteMulti () {
try {
let result = await client.deleteMulti(['obj-1', 'obj-2', 'obj-3']);
console.log(result);
let result = await client.deleteMulti(['obj-1', 'obj-2', 'obj-3'], {
quiet: true
});
console.log(result);
} catch (e) {
console.log(e);
}
}

deleteMulti();
```

自定义域名绑定

OSS支持用户将自定义的域名绑定到OSS服务上，这样能够支持用户无缝地将存储迁移到OSS上。例如用户的域名是my-domain.com，之前用户的所有图片资源都是形如http://img.my-domain.com/x.jpg的格式，用户将图片存储迁移到OSS之后，通过绑定自定义域名，仍可以使用原来的地址访问到图片：

- 开通OSS服务并创建Bucket
- 修改域名的DNS配置，增加一个CNAME记录，将img.my-domain.com指向OSS服务的endpoint（如my-bucket.oss-cn-hangzhou.aliyuncs.com）
- 在官网控制台将img.my-domain.com与创建的Bucket绑定
- 将图片上传到OSS的这个Bucket中

这样就可以通过原地址http://img.my-domain.com/x.jpg访问到存储在OSS上的图片。绑定自定义域名请参考自定义域名绑定

在使用SDK时，也可以使用自定义域名作为endpoint，这时需要将cname参数设置为true，如下面的例子：

```
let OSS = require('ali-oss')

let client = new OSS({
endpoint: '<Your endpoint>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
cname: true
});

client.useBucket('my-bucket')
```

注意：

- 使用CNAME时，无法使用list_buckets接口。（因为自定义域名已经绑定到某个特定的Bucket）

使用STS访问

OSS可以通过阿里云STS服务，临时进行授权访问。使用STS时请按以下步骤进行：

1. 在官网控制台创建子账号，参考OSS STS。
2. 在官网控制台创建STS角色并赋予子账号扮演角色的权限，参考OSS STS。
3. 使用子账号的AccessKeyId/AccessKeySecret向STS申请临时token。
4. 使用临时token中的认证信息创建OSS的Client。
5. 使用OSS的Client访问OSS服务。

在使用STS访问OSS时，需要设置stsToken参数，如下面的例子所示：

```
let OSS = require('ali-oss');
let STS = OSS.STS;

let sts = new STS({
accessKeyId: '<子账号的AccessKeyId>',
accessKeySecret: '<子账号的AccessKeySecret>'
});

async function assumeRole () {
try {
let token = yield sts.assumeRole(
'<role-arn>', '<policy>', '<expiration>', '<session-name>');

let client = new OSS({
region: '<region>',
accessKeyId: token.credentials.AccessKeyId,
accessKeySecret: token.credentials.AccessKeySecret,
stsToken: token.credentials.SecurityToken,
bucket: '<bucket-name>'
```

```
});
} catch (e) {
console.log(e);
}
}

assumeRole();
```

在向STS申请临时token时，还可以指定自定义的STS Policy。这样申请的临时权限是**所扮演角色的权限与Policy指定的权限的交集。**下面的例子将通过指定STS Policy申请对my-bucket的只读权限，并指定临时token的过期时间为15分钟：

```
let OSS = require('ali-oss');
let STS = OSS.STS;

let sts = new STS({
accessKeyId: '<子账号的AccessKeyId>',
accessKeySecret: '<子账号的AccessKeySecret>'
});

let policy = {
"Statement": [
{
"Action": [
"oss:Get*"
],
"Effect": "Allow",
"Resource": ["acs:oss:*:*:my-bucket/*"]
}
],
"Version": "1"
};

async function assumeRole () {
try {
let token = await sts.assumeRole(
'<role-arn>', policy, 15 * 60, '<session-name>');
let client = new OSS({
region: '<region>',
accessKeyId: token.credentials.AccessKeyId,
accessKeySecret: token.credentials.AccessKeySecret,
stsToken: token.credentials.SecurityToken,
bucket: '<bucket-name>'
});
} catch (e) {
console.log(e);
}
}

assumeRole();
```

设置访问权限

OSS允许用户对Bucket和Object分别设置访问权限，方便用户控制自己的资源可以被如何访问。对于Bucket，有三种访问权限：

- public-read-write 允许匿名用户向该Bucket中创建/获取/删除Object
- public-read 允许匿名用户获取该Bucket中的Object
- private 不允许匿名访问，所有的访问都要经过签名

创建Bucket时，默认是private权限。之后用户可以通过putBucketACL来设置Bucket的权限，通过getBucketACL来获取Bucket的权限。

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function bucketACL () {
try {
let result = await client.getBucketACL('bucket-name');
console.log(result);

let result = await client.putBucketACL('bucket-name', 'region', 'acl');
console.log(result);
} catch (e) {
console.log(e);
}
}

bucketACL();
```

对于Object，有四种访问权限：

- default 继承所属的Bucket的访问权限，即与所属Bucket的权限值一样
- public-read-write 允许匿名用户读写该Object
- public-read 允许匿名用户读该Object
- private 不允许匿名访问，所有的访问都要经过签名

创建Object时，默认为default权限。之后用户可以通过putACL来设置Object的权限。

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
```

```
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function bucketACL () {
try {
let result = await client.getACL('my-object');
console.log(result.acl); // default

await client.putACL('my-object', 'public-read');
result = yield client.getACL('my-object');
console.log(result.acl); // public-read
} catch (e) {
console.log(e);
}
}
```

需要注意的是：

1. 如果设置了Object的权限（非default），则访问该Object时进行权限认证时会优先判断Object的权限，而Bucket的权限设置会被忽略。

   允许匿名访问时（设置了public-read或者public-read-write权限），用户可以直接通过浏览器访问，例如：

   ```
   http://bucket-name.oss-cn-hangzhou.aliyuncs.com/object.jpg
   ```

更多关于访问权限控制的内容请参考 访问控制

管理生命周期

OSS允许用户对Bucket设置生命周期规则，以自动淘汰过期掉的文件，节省存储空间。用户可以同时设置多条规则，一条规则包含：

- 规则ID，用于标识一条规则，不能重复
- 受影响的文件前缀，此规则只作用于符合前缀的文件
- 过期时间，有两种指定方式：
  - i. 指定距文件最后修改时间N天过期
  - ii. 指定在具体的某一天过期，即在那天之后符合前缀的文件将会过期，**而不论文件的最后修改时间**。不推荐使用。
- 是否生效

更多关于生命周期的内容请参考 文件生命周期

设置生命周期规则

通过putBucketLifecycle来设置生命周期规则：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function putBucketLifecycle () {
try {
let result = yield client.putBucketLifecycle('bucket-name', 'region', [
{
id: 'rule1',
status: 'Enabled',
prefix: 'foo/',
days: 3
},
{
id: 'rule2',
status: 'Disabled',
date: '2022-10-11T00:00:00.000Z'
}
]);
console.log(result);
} catch (e) {
console.log(e);
}
}

putBucketLifecycle();
```

查看生命周期规则

通过getBucketLifecycle来查看生命周期规则：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function getBucketLifecycle () {
try {
```

```
let result = await client.getBucketLifecycle('bucket-name', 'region');
console.log(result);
} catch (e) {
console.log(e);
}
}

getBucketLifecycle();
```

清空生命周期规则

通过deleteBucketLifecycle设置一个空的Rule数组来清空生命周期规则：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function deleteBucketLifecycle () {
try {
let result = await client.deleteBucketLifecycle('bucket-name', 'region');
console.log(result);
} catch (e) {
console.log(e);
}
}

deleteBucketLifecycle();
```

设置访问日志

OSS允许用户对Bucket设置访问日志记录，设置之后对于Bucket的访问会被记录成日志，日志存储在OSS上由用户指定的Bucket中，文件的格式为：

```
<TargetPrefix><SourceBucket>-YYYY-mm-DD-HH-MM-SS-UniqueString
```

其中TargetPrefix由用户指定。更多关于访问日志的内容请参考 Bucket访问日志

开启Bucket日志

通过putBucketLogging来开启日志功能：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function putBucketLogging () {
try {
let result = await client.putBucketLogging('bucket-name', 'region', 'logs/');
console.log(result)
} catch (e) {
console.log(e)
}
}

putBucketLogging();
```

查看Bucket日志设置

通过getBucketLogging来查看日志设置：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function getBucketLogging() {
try {
let result = await client.getBucketLogging('bucket-name', 'region');
console.log(result);
} catch (e) {
console.log(e);
}
})

getBucketLogging();
```

关闭Bucket日志

通过deleteBucketLogging来关闭日志功能：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function deleteBucketLogging () {
try {
let result = await client.deleteBucketLogging('bucket-name', 'region');
console.log(result);
} catch (e) {
console.log(e);
}

deleteBucketLogging();
```

静态网站托管

在自定义域名绑定中提到，OSS允许用户将自己的域名指向OSS服务的地址。这样用户访问他的网站的时候，实际上是在访问OSS的Bucket。对于网站，需要指定首页(index)和出错页(error)分别对应的Bucket中的文件名。

更多关于静态网站托管的内容请参考 OSS静态网站托管

设置托管页面

通过putBucketWebsite来设置托管页面：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function putBucketWebsite () {
try {
let result = await client.putBucketWebsite('bucket-name', 'region', {
index: 'index.html',
error: 'error.html'
```

```
});
console.log(result);
} catch (e) {
console.log(e);
}
}

putBucketWebsite();
```

查看托管页面

通过getBucketWebsite来查看托管页面：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function getBucketWebsite () {
try {
let result = await client.getBucketWebsite('bucket-name', 'region');
console.log(result);
} catch (e) {
console.log(e);
}
}

getBucketWebsite();
```

清除托管页面

通过deleteBucketWebsite来清除托管页面：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function deleteBucketWebsite() {
try {
let result = await client.deleteBucketWebsite('bucket-name', 'region');
console.log(result);
```

```
} catch (e) {
console.log(e);
}
}

deleteBucketWebsite();
```

设置防盗链

OSS是按使用收费的服务，为了防止用户在OSS上的数据被其他人盗链，OSS支持基于HTTP header中表头字段referer的防盗链方法。更多OSS防盗链请参考：OSS防盗链

设置Referer白名单

通过putBucketReferer设置Referer白名单：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function putBucketReferer () {
try {
let result = await client.putBucketReferer('bucket-name', 'region', true, [
'my-domain.com',
'*.example.com'
]);
console.log(result);
} catch (e) {
console.log(e);
}
}

putBucketReferer();
```

查看Referer白名单

通过getBucketReferer查看Referer白名单：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function getBucketReferer () {
try {
let result = await client.getBucketReferer('bucket-name', 'region');
console.log(result);
} catch (e) {
console.log(e);
}
}

getBucketReferer();
```

清空Referer白名单

通过deleteBucketReferer设置清空Referer白名单：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function deleteBucketReferer () {
try {
let result = await client.deleteBucketReferer('bucket-name', 'region');
console.log(result);
} catch (e) {
console.log(e);
}
}

deleteBucketReferer();
```

图片处理

OSS图片处理，是OSS对外提供的海量、安全、低成本、高可靠的图片处理。用户将原始图片上传保存到

OSS，通过简单的 RESTful 接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。图片处理提供图片处理接口，图片上传请使用上传接口。基于OSS图片处理，用户可以搭建自己的图片处理服务。

图片处理基础功能

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放、裁剪、旋转
- 图片效果
- 图片添加图片、文字、图文混合水印
- 自定义图片处理样式，在控制台的 图片处理 -> 样式管理 中定义
- 通过级联处理调用多个图片处理功能

图片处理使用

图片处理使用标准的 HTTP GET 请求来访问，所有的处理参数是编码在 URL 中的QueyString。

匿名访问

如果图片文件（Object）的访问权限是 公共读 ，如下表所示的权限，则可以匿名访问图片服务。

Bucket权限	Object权限
公共读私有写（public-read）或 公共读写（public-read-write）	默认（default）
任意权限	公共读私有写（public-read）或 公共读写（public-read-write）

通过如下格式的三级域名匿名访问图片处理：

```
http://bucket.<endpoint>/object?x-oss-process=image/action,param_value
```

- bucket：用户的存储空间（bucket）名称
- endpoint：用户存储空间所在数据中心的访问域名
- object：用户上传在OSS上的图片文件
- image：图片处理保留标志符
- action：用户对图片做的操作，如缩放、裁剪、旋转等
- param：用户对图片做的操作所对应的参数

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

自定义样式，使用如下格式的三级域名匿名访问图片处理：

```
http://bucket.<endpoint>/object?x-oss-process=x-oss-process=style/name
```

- style：用户自定义样式系统保留标志符
- name：自定义样式名称，即控制台定义样式的 规则名

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=style/oss-pic-style-w-100
```

通过级联处理，可以对一张图片顺序实施多个操作，格式如下：

```
http://bucket.<endpoint>/object?x-oss-process=image/action,parame_value/action,param_value/...
```

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100/rotate,90
```

图片服务也支持HTTPS访问，例如：

```
https://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

授权访问

对私有权限的文件（Object），如下表所示的权限，必须通过授权才能访问图片服务。

Bucket权限	Object权限
私有读写（private）	默认权限（default）
任意权限	私有读写（private）

生成带签名的图片处理的URL代码如下：

```
let OSS = require('ali-oss');

let client = new OSS({
accessKeyId: '<accessKeyId>',
accessKeySecret: '<accessKeySecret>',
bucket: '<bucketName>',
endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});

// 过期时间10分钟, 图片处理式样"image/resize,w_300"
let signUrl = client.signatureUrl('example.jpg', {expires: 600, 'process' : 'image/resize,w_300'});
```

```
console.log("signUrl="+signUrl);
```

提示：

- 授权访问支持 自定义样式、 HTTPS、 级联处理
- 指定过期时间expires的单位是秒

SDK访问

对于任意权限的图片文件，都可以直接使用 SDK 访问图片、进行处理。

提示：

- SDK处理图片文件支持 自定义样式、 HTTPS、 级联处理

基础操作

图片处理的基础操作包括，获取图片信息、格式转换、缩放、裁剪、旋转、效果、水印等。

```
let OSS = require('ali-oss');

let client = new OSS({
accessKeyId: '<accessKeyId>',
accessKeySecret: '<accessKeySecret>',
bucket: '<bucketName>',
endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});

// 缩放
async function scale () {
try {
let result = await client.get('example.jpg', './example-resize.jpg', { process: 'image/resize,m_fixed,w_100,h_100'});
} catch (e) {
console.log(e);
}
}

// 裁剪
async function cut () {
try {
let result = await client.get('example.jpg', './example-crop.jpg', { process:
'image/crop,w_100,h_100,x_100,y_100,r_1'});
} catch (e) {
console.log(e)
}
}

// 旋转
async function rotate () {
try {
```

```
let result = await client.get('example.jpg', './example-rotate.jpg', { process: 'image/rotate,90'});
} catch (e) {
console.log(e);
}
}

// 锐化
async function sharpen () {
try {
let result = yield client.get('example.jpg', './example-sharpen.jpg', { process: 'image/sharpen,100'});
} catch (e) {
console.log(e);
}
}

// 水印
async function watermark () {
try {
let result = yield client.get('example.jpg', './example-watermark.jpg', { process:
'image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ'});
} catch (e) {
console.log(e);
}
}

// 格式转换
async function format () {
try {
let result = await client.get('example.jpg', './example-format.jpg', { process: 'image/format,png'});
} catch (e) {
console.log(e);
}
}

// 图片信息
async function info () {
try {
let result = await client.get('example.jpg', './example-info.txt', {process: 'image/info'});
} catch (e) {
console.log(e);
}
}
```

自定义样式

提示：需要事先到oss控制台添加自定义式样example-sytle

```
let OSS = require('ali-oss');

let client = new OSS({
accessKeyId: '<accessKeyId>',
accessKeySecret: '<accessKeySecret>',
bucket: '<bucketName>',
```

```
endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});

// 自定义样式
async function style () {
try {
let result = await client.get('example.jpg', './example-custom-style.jpg', {process: 'style/example-sytle"'});
} catch (e) {
console.log(e);
}
}

style();
```

级联处理

```
let OSS = require('ali-oss');

let client = new OSS({
accessKeyId: '<accessKeyId>',
accessKeySecret: '<accessKeySecret>',
bucket: '<bucketName>',
endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});

// 级联处理
async function cascade () {
try {
let result = await client.get('example.jpg', './example-cascade.jpg', {process:
'image/resize,m_fixed,w_100,h_100/rotate,90'});
} catch (e) {
console.log(e);
}
}

cascade();
```

图片处理工具

- 可视化图片处理工具 ImageStyleViever ，可以直观的看到OSS图片处理的结果
- OSS图片处理的功能、使用演示 页面

异常

使用SDK时如果请求出错，会有相应的异常抛出。服务器端异常通常会包含以下信息：

- status: 出错请求的HTTP状态码
- code: OSS的错误码
- message: OSS的错误信息
- requestId: 标识该次请求的UUID；当您无法解决问题时，可以凭这个RequestId来请求OSS开发工程师的帮助

OSS中常见的错误信息请参考 OSS错误响应

调试

在Node.js中您可以通过设置DEBUG环境变量来开启调试：

```
DEBUG=ali-oss node app.js
```

在浏览器环境中您可以通过在console中设置localStorage.debug变量来开启调试：

```
localStorage.debug = 'ali-oss'
```

常见问题

如何 HTTPS 访问

初始化 SDK 时，可传入以下几个参数：

- region: 参数是指您申请 OSS 服务时的区域，例如oss-cn-hangzhou。完整的区域列表可以在 OSS 服务节点查看。
- internal: 配合region使用，如果指定 internal 为 true，则访问内网节点。
- secure: 配合region使用，如果指定了secure为true，则使用 HTTPS 访问。
- endpoint: 例如http://oss-cn-hangzhou.aliyuncs.com，如果指定了endpoint，则region会被忽略，endpoint可以指定HTTPS，也可以是IP形式。

如何获取上传进度

使用分片上传时，可获取上传进度。相关文档。

如何获取下载进度

Node 中可根据下载流的大小来计算进度。

如何上传base64编码的图片

将 Base64 内容转换成 File 对象，在调用接口上传至 OSS 服务器

```
 function dataURLtoFile(dataurl, filename) {
let arr = dataurl.split(','), mime = arr[0].match(/:(.*?);/)[1],
bstr = atob(arr[1]), n = bstr.length, u8arr = new Uint8Array(n);
while(n--){
u8arr[n] = bstr.charCodeAt(n);
}
return new File([u8arr], filename, {type:mime});
}

let file = dataURLtoFile('<base64 content>', '');

client.multipartUpload('<oss file name>', file).then( (res)=> {
console.log(res)
}).catch((err) => {
console.log(err)
});
```

如何上传文件到指定目录

给要上传的 object 名称前加指定目录前缀即可，可参考: OSS 和文件系统对比。

```
let OSS = require('ali-oss')
let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

client.multipartUpload('base-dir/' +'object-key', 'local-file', {
progress: async function (p) {
console.log('Progress: ' + p);
}
});
console.log(result);
}).then((res) => {
console.log(res)
}). catch((err) => {
console.log(err);
});
```

如何获取object的签名URL

可调用 signatureUrl 方法，获取下载地址，可查看相关文档。

常见错误参考

- SDK 开启异常日志
- OSS 常见错误

Browser-JS-SDK

安装

SDK安装

目前官网文档中的demo都是基于SDK 6.X，版本低于6.X的可参考 5.X开发文档, 升级6.X请移步升级文档

- Github地址
- API文档
- ChangeLog

要求

开通阿里云OSS服务，如果您还没不了解阿里云OSS服务，请登录OSS产品主页了解。详情请参见产品介绍。

创建AccessKeyId和AccessKeySecret，由于云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践。如果部署在服务端可以使用RAM子账号或STS来进行API访问或日常运维管控操作，如果部署在客户端请使用STS方式来进行API访问。详情请参见访问控制。

环境要求

OSS BrowserJS-SDK基于Node.js环境构建，通过Browserify和Babel产生适用于浏览器的代码。

但是由于浏览器环境的特殊性，有一些功能无法使用：

- 流式上传：浏览器中无法设置chunked编码，用分片上传替代
- 操作本地文件：浏览器中不能直接操作本地文件系统，用签名URL的方式下载

- Bucket相关的操作（listBuckets/BucketACL/BucketLogging等），由于OSS暂时不支持Bucket相关的跨域请求，Bucket相关的操作可以在控制台进行

使用方式

OSS BrowserJS-SDK同时支持同步和异步的使用方式

同步方式：基于ES6规范的async/await,使得异步方式同步化

异步方式：类似callback的方式，API接口返回Promise，使用.then()处理返回结果，使用.catch()处理错误

无论同步方式还是异步方式，均使用new OSS()创建client。

下面分别举例，先上传一个文件，然后立即下载这个文件：

同步方式

```
let client = new OSS(...);

async function put () {
try {
let r1 = await client.put('object', '/tmp/file');
console.log('put success: %j', r1);
let r2 = await client.get('object');
console.log('get success: %j', r2);
} catch (e) {
console.error('error: %j', e);
}
}

put();
```

异步方式

```
let client = new OSS(...);

client.put('object', '/tmp/file').then(function (r1) {
console.log('put success: %j', r1);
return client.get('object');
}).then(function (r2) {
console.log('get success: %j', r2);
}).catch(function (err) {
console.error('error: %j', err);
});
```

安装

在浏览器中使用

支持的浏览器：

- IE(>=10)和Edge
- 主流版本的Chrome/Firefox/Safari
- 主流版本的Android/iOS/WindowsPhone

浏览器引入：

注意：因为部分浏览器不支持promise，需要引入promise兼容库。
例如：IE10和IE11 需要引入promise-polyfill。

```
<!-- 引入在线资源 -->
<script src="http://gosspublic.alicdn.com/aliyun-oss-sdk-x.x.x.min.js"></script>


<!-- 引入本地资源 -->
<script src="./aliyun-oss-sdk-x.x.x.min.js"></script>
```

注意：

- x.x.x代表版本号，具体版本信息可在这访问查看
- 引入在线资源方式依赖于CDN服务器的稳定性，推荐用户使用离线方式引入，即通过本地资源或自行构建的方式。

就可以在代码中使用OSS对象：

```
<script type="text/javascript">
let client = new OSS({
region: '<oss region>',
//云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践，创建并使用STS方式来进行API访问
accessKeyId: '<Your accessKeyId(STS)>',
accessKeySecret: '<Your accessKeySecret(STS)>',
stsToken: '<Your securityToken(STS)>',
bucket: '<Your bucket name>'
});
</script>
```

使用npm安装SDK的开发包：

```
npm install ali-oss
```

成功安装完成之后，即可使用 import 或 require 进行引用。

```
let OSS = require('ali-oss');
let client = new OSS({
region: '<oss region>',
//云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践，部署在服务端使用RAM子账号或STS，部署在客户端使用STS。
accessKeyId: '<Your accessKeyId>',
accessKeySecret: '<Your accessKeySecret>',
bucket: '<Your bucket name>'
});
```

使用Bower:

对于Web项目，你也可以通过Bower安装SDK：

```
bower install ali-oss
```

然后在网页中添加<script>标签：

```
<script src="bower_components/ali-oss/dist/aliyun-oss-sdk.min.js"></script>
```

快速开始

下面介绍如何在BrowserJS-SDK中快速使用访问OSS服务，包括上传/下载文件和查看文件列表。

> **注意：**云账号AccessKey拥有所有API访问权限，建议遵循阿里云安全最佳实践，在客户端不要使用会泄露配置的ak，所以请创建并使用STS方式（临时账号权限）给客户端授权。

STS场景说明：临时账号权限（STS）
更多有关浏览器的使用请参考：浏览器应用
示例工程请参考：example

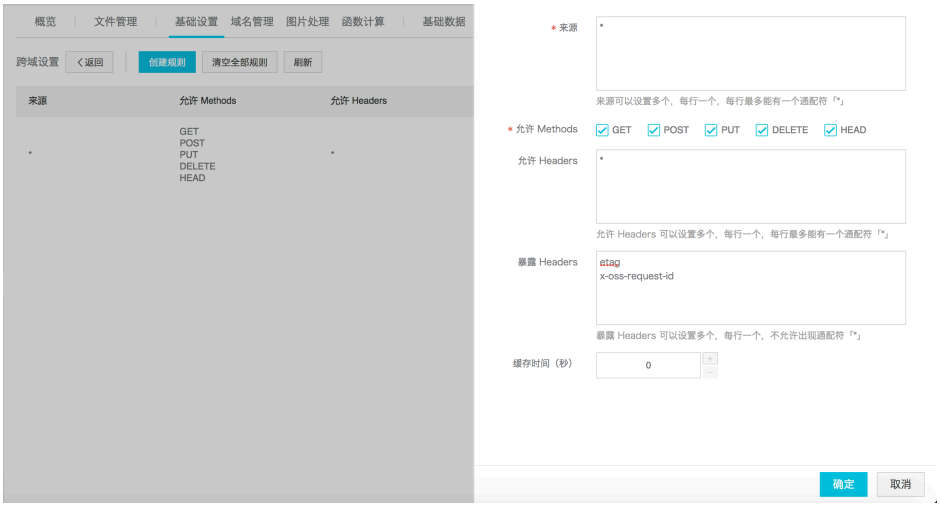
Bucket设置

从浏览器中直接访问OSS需要开通Bucket的CORS设置：

- 将allowed origins设置成 *
- 将allowed methods设置成 PUT, GET, POST, DELETE, HEAD

- 将allowed headers设置成 *
- 将expose headers设置成 etag x-oss-request-id

注意：请将该条CORS规则设置成所有CORS规则的第一条。



使用SDK

目前浏览器中不能直接进行Bucket相关的操作（例如list buckets, get/set bucket logging/referer/website/cors）。但是可以进行Object相关的操作（例如上传/下载文件，查看文件列表等）。

包含SDK

首先在html文件的<head>中包含如下标签：

```
<script src="http://gosspublic.alicdn.com/aliyun-oss-sdk-x.x.x.min.js"></script>
```

想了解其他引入方式可参考安装介绍。

通过new OSS来创建client，创建后返回的是Promise, 类似于callback的方式，在.then()中处理返回的结果，在.catch()中处理错误。

搭建STS Server 并从客户端获取临时授权信息

为账号开通STS服务，如果已开通可以跳过，开通STS请参见STS开通说明, 具体OSS授权 Policy配置参见RAM和STS授权策略（Policy）配置

借助STS的SDK或API应用专属的STS签名服务。为了方便大家快速理解，提供了多个语言的版本

Demo程序供参考：

- NodeJS: 下载地址
- PHP: 下载地址
- Java: 下载地址
- Ruby: 下载地址

> **说明：**Demo程序仅供STS功能实现参考，生产环境使用请自行开发鉴权等业务所需代码。

在前端（浏览器）中向搭建的STS服务发起请求，获取STS临时授权信息。

```
<script type="text/javascript">
OSS.urllib.request("http://your_sts_server/",
{method: 'GET'},
function (err, response) {
if (err) {
return alert(err);
}
try {
result = JSON.parse(response);
} catch (e) {
errmsg = 'parse sts response info error: ' + e.message;
return alert(errmsg);
}
// console.log(result)
})
</script>
```

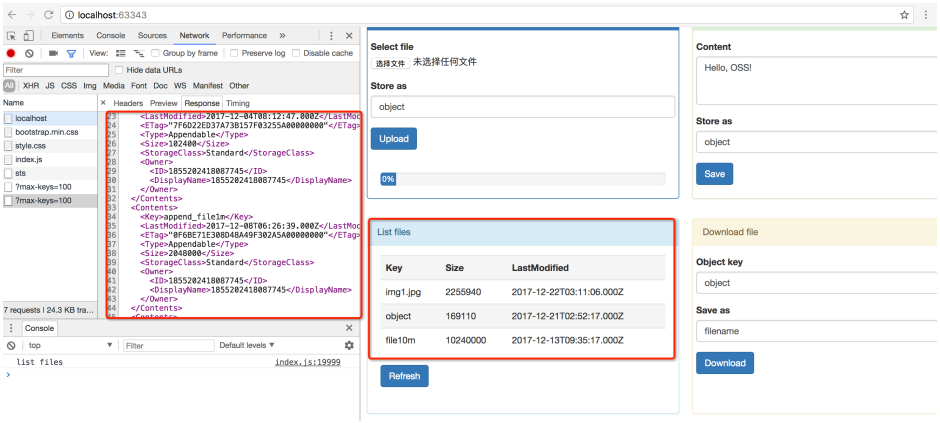
查看文件列表

```
<script type="text/javascript">
OSS.urllib.request("http://your_sts_server/",
{method: 'GET'},
function (err, response) {
if (err) {
return alert(err);
}
try {
result = JSON.parse(response);
} catch (e) {
return alert('parse sts response info error: ' + e.message);
}
let client = new OSS({
accessKeyId: result.AccessKeyId,
accessKeySecret: result.AccessKeySecret,
stsToken: result.SecurityToken,
endpoint: '<oss endpoint>',
bucket: '<Your bucket name>'
```

```
});

client.list({
'max-keys': 10
}).then(function (result) {
console.log(result);
}).catch(function (err) {
console.log(err);
});
});
</script>
```

其中region参数是指您申请OSS服务时的区域，例如oss-cn-hangzhou。完整的区域列表可以在OSS服务节点查看。

在浏览器中打开html文件，然后打开Chrome的开发者控制台，就可以看到list文件的结果了。



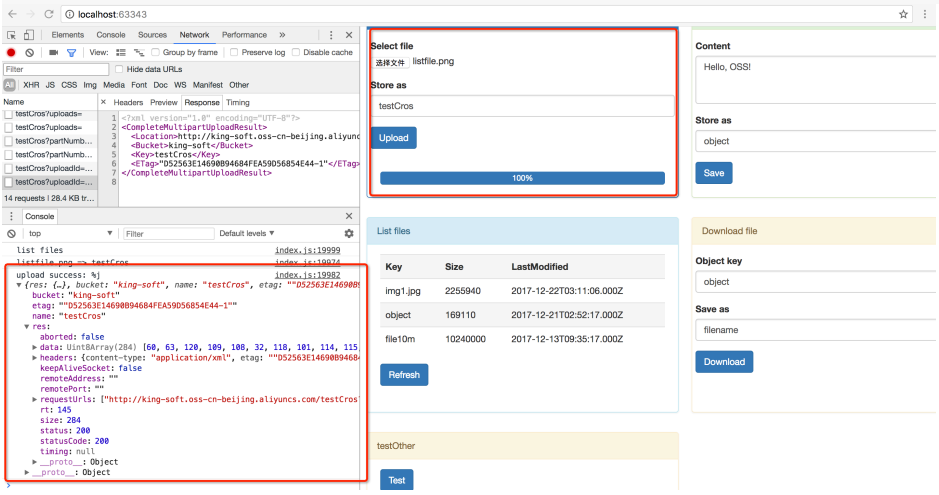
注意：浏览器是不受信任的环境，如果把AccessKeyId和AccessKeySecret直接保存在浏览器端，**存在极高的风险**。建议只在测试时使用明文设置，业务应用中请使用 STS鉴权模式 和 自签名模式 ,详细说明请参考访问控制、移动端直传。

上传文件

下面使用multipartUpload接口来上传文件, 具体multipartUpload使用请参考multipartUpload指南

```
<body>
<input type="file" id="file" />
<script type="text/javascript">
document.getElementById('file').addEventListener('change', function (e) {
let file = e.target.files[0];
let storeAs = 'upload-file';
console.log(file.name + ' => ' + storeAs);
OSS.urllib.request("http://your_sts_server/",
{method: 'GET'},
function (err, response) {
if (err) {
return alert(err);
}
```

```
try {
result = JSON.parse(response);
} catch (e) {
return alert('parse sts response info error: ' + e.message);
}
let client = new OSS({
accessKeyId: result.AccessKeyId,
accessKeySecret: result.AccessKeySecret,
stsToken: result.SecurityToken,
endpoint: '<oss endpoint>',
bucket: '<Your bucket name>'
});
//storeAs表示上传的object name , file表示上传的文件
client.multipartUpload(storeAs, file).then(function (result) {
console.log(result);
}).catch(function (err) {
console.log(err);
});
});
});
</script>
</body>
```

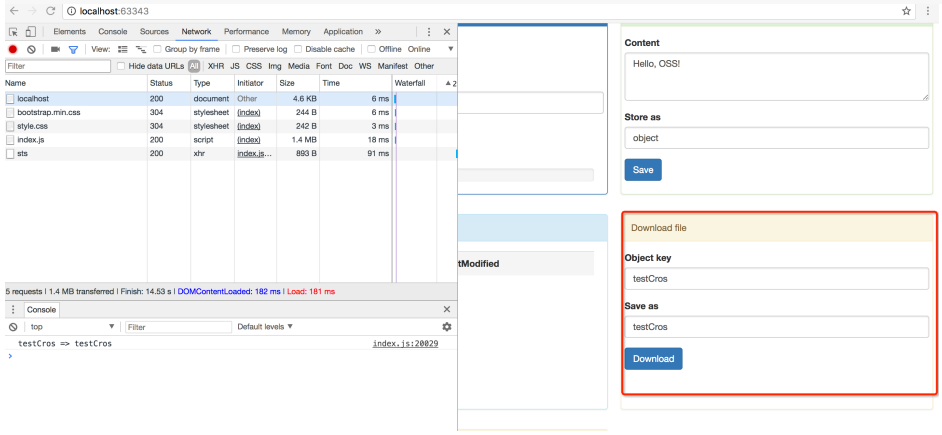


下载文件

在浏览器中不能直接操作文件，因此采用签名URL的方式来生成文件的下载链接，用户只需要单击链接就可以下载文件。

```
<body>
<input type="button" id="download" value="Download" />
<script type="text/javascript">
document.getElementById('download').addEventListener('click', function (e) {
let objectKey = 'test/download_file';
let saveAs = 'download_file';
console.log(objectKey + ' => ' + saveAs);
OSS.urllib.request("http://your_sts_server/",
{method: 'GET'},
```

```
function (err, response) {
if (err) {
return alert(err);
}
try {
result = JSON.parse(response);
} catch (e) {
return alert('parse sts response info error: ' + e.message);
}
let client = new OSS.Wrapper({
accessKeyId: result.AccessKeyId,
accessKeySecret: result.AccessKeySecret,
stsToken: result.SecurityToken,
endpoint: '<oss endpoint>',
bucket: '<Your bucket name>'
});
let result = client.signatureUrl(objectKey, {
expires: 3600,
response: {
'content-disposition': 'attachment; filename="' + saveAs + '"'
}
});
console.log(result);
window.location = result;
});
});
</script>
</body>
```



配置项

OSS(options)介绍

- [accessKeyId] {String} 通过阿里云控制台创建的access key。

- [accessKeySecret] {String} 通过阿里云控制台创建的access secret。
- [stsToken] {String} 使用临时授权方式，详情请参见使用STS访问。
- [bucket] {String} 通过控制台创建的bucket, 或通过putBucket创建。
- [endpoint] {String} oss 域名。
- [region] {String} bucket 所在的区域, 默认 oss-cn-hangzhou。
- [internal] {Boolean} 是否使用阿里云内部网访问，比如采用ecs访问oss，设置true, 采用internal的 endpoint 会节约费用, 默认false。
- [secure] {Boolean} (secure: true) 使用 HTTPS，(secure: false) 则使用 HTTP，具体可查看。
- [timeout] {String|Number} 超时时间, 默认 60s。

example:

```
let oss = require('ali-oss');

let store = oss({
accessKeyId: 'your access key',
accessKeySecret: 'your access secret',
bucket: 'your bucket name',
region: 'oss-cn-hangzhou'
});
```

上传文件

用户可以通过以下方式向OSS中上传文件：

- 上传本地文件
- 上传Buffer内容
- 上传blob数据
- 分片上传
- 暂停分片上传

上传本地文件

请直接参考分片上传

上传Buffer内容

用户也可以通过put接口简单地将Buffer中的内容上传到OSS：

```
let OSS = require('ali-oss');
```

```
let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function put () {
try {
let result = await client.put('object-key', new Buffer('hello world'));
console.log(result);
} catch (e) {
console.log(e);
}
}

put();
```

上传Blob数据

用户也可以通过put接口简单地将Blob中的内容上传到OSS：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function putBlob () {
try {
let result = await client.put('object-key', new Blob(['content'],{ type: 'text/plain' }));
console.log(result);
} catch (e) {
conosle.log(e);
}
}
putBlob();
```

分片上传

在需要上传的文件较大时，可以通过multipartUpload接口进行分片上传。分片上传的好处是将一个大请求分成多个小请求来执行，这样当其中一些请求失败后，不需要重新上传整个文件，而只需要上传失败的分片就可以了。一般对于大于100MB的文件，建议采用分片上传的方法，每次进行分片上传都建议重新new一个新的OSS实例。

阿里云分片上传流程主要会调用3个api，包含

1. InitiateMultipartUpload, 分片任务初始化接口。
2. UploadPart,单独的分片上传接口。
3. CompleteMultipartUpload,分片上传完成后任务完成接口。各个api的使用方式会在后续介绍，具体内容参见MultipartUpload。

相关参数：

- name {String} object 名称
- file {File|Blob} HTML5 Web File or Blob的数据格式
- [options] {Object} 额外参数
    - [checkpoint] {Object} 断点记录点，可以进行断点续传, 如果设置这个参数，上传会从断点开始，如果没有设置，就会重新上传.
        - file {File} 用户选取的文件对象，如果浏览器重启后这个需要用户手动触发进行设置
        - name {String} 上传的object key
        - fileSize {Number} 文件大小
        - partSize {Number} 分片大小
        - uploadId {String} 分片上传的ID
        - doneParts {Array} 已完成的分片的数组, 包含的对象结构如下
            - number {Number} 分片的number
            - etag {String} 分片的etag
    - [partSize] {Number} 分片大小
    - [progress] {Function} thunk 或 async 形式, 回调函数包含三个参数
        - (percentage {Number} 进度百分比(0-1之间小数)
        - checkpoint {Object} 断点记录点
        - res {Object}) 单次part成功返回的response
    - [meta] {Object} 用户自定义header meta信息, header前缀 x-oss-meta-
    - [headers] {Object} http 额外的头字段, 详情请看 RFC 2616
        - ‘Cache-Control’ 通用消息头被用于在http 请求和响应中通过指定指令来实现缓存机制, e.g.: Cache-Control: public, no-cache
        - ‘Content-Disposition’ 指示回复的内容该以何种形式展示，是以内联的形式（即网页或者页面的一部分），还是以附件的形式下载并保存到本地, e.g.: Content-Disposition: somename
        - ‘Content-Encoding’ 用于对特定媒体类型的数据进行压缩, e.g.: Content-Encoding: gzip
        - ‘Expires’ 过期时间, e.g.: Expires: 3600000
    - [callback] {Object} callback回调设置。具体内容参见Callback：
        - url {String} 和oss server交互的回调服务器地址。对应于CallBack参数中的callbackUrl。必须有。
        - body {String} 发起回调时请求body的值。jason格式。对应于CallBack参数中的callbackBody。必须有。
        - [host] {String} 发起回调请求时Host头的值。对应于CallBack参数中的callbackHost。
        - [contentType] {String} 发起回调请求的Content-Type。对应于CallBack参数中

的callbackBodyType 。
- [customValue] {Object} 发起回调请求自定义参数。对应于CallBack参数中的callback-var。

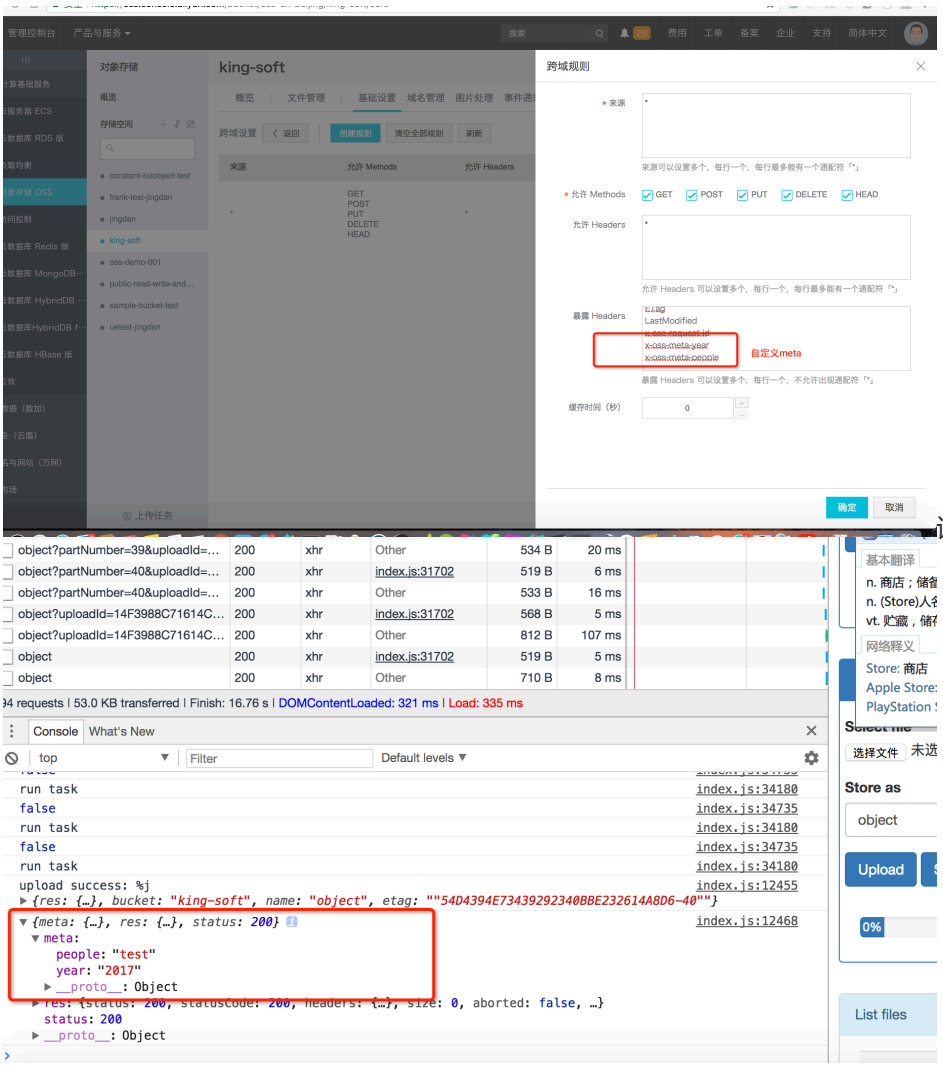
范例：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function multipartUpload () {
try {
let result = await client.multipartUpload('object-key', 'local-file', {
progress: async function (p) {
console.log('Progress: ' + p);
},
meta: {
year: 2017,
people: 'test'
},
mime: 'image/jpeg'
});
console.log(result);
let head = await client.head('object-key');
console.log(head);
} catch (e) {
console.log(e);
}
}
```

上面的progress参数是一个进度回调函数，用于获取上传进度

```
const progress = async function progress(p, checkpoint) {
console.log(p)
};
```

上面的meta参数是一个用户自定义的元数据，通过head接口可以获取到object的meta数据，但是这个meta header 需要在控制台跨域设置里边的暴露header中设置好，如图：



请求成功后的返回结果展示:

.initMultipartUpload(name[, options])

initiateMultipartUpload 的返回结果中含有UploadId ，它是区分分片上传事件的唯一标识，在分片上传后续的的操作中，我们都会用到它。

相关参数:

- name {String} object 名称
- [options] {Object} 额外参数
  - [timeout] {Number} 超时设置
  - [mime] 上传文件类型。例如: application/octet-stream
  - [meta] {Object} 用户自定义header meta信息, header前缀 ‘x-oss-meta- ’
  - [headers] {Object} http 额外的头字段, 详情请看 RFC 2616
    - ‘Cache-Control’ 通用消息头被用于在http 请求和响应中通过指定指令来实现缓存机制, e.g.: Cache-Control: public, no-cache
    - ‘Content-Disposition’ 指示回复的内容该以何种形式展示，是以内联的形式（即网页或者页面的一部分），还是以附件的形式下载并保存到本地, e.g.: Content-Disposition: somename

- ‘Content-Encoding’ 用于对特定媒体类型的数据进行压缩, e.g.: Content-Encoding: gzip
- ‘Expires’ 过期时间, e.g.: Expires: 3600000;
- [x-oss-server-side-encryption] 只需要请求中携带x-oss-server-side-encryption的HTTP Header，并指定其值为AES256，即可以实现该Object的服务器端加密存储

请求成功返回内容:

- res {Object} 返回内容，包括
  - status {Number} 返回状态码
  - headers {Object} 返回的头信息
  - size {Number} 返回内容的长度
  - rt {Number} 请求耗时 (ms)
- bucket {String} bucket 名称
- name {String} 储存在oss的object名称
- uploadId {String} upload id 分片上传任务id

范例

```
let result = await store.initMultipartUpload('object');
console.log(result);
```

.uploadPart(name, uploadId, partNo, file, start, end[, options])

uploadPart是分片上传的核心接口,有以下要点需要注意：

- UploadPart 方法要求除最后一个Part以外，其他的Part大小都要大于100KB。但是Upload Part接口并不会立即校验上传Part的大小（因为不知道是否为最后一块）；只有当Complete Multipart Upload的时候才会校验。
- OSS会将服务器端收到Part数据的MD5值放在ETag头内返回给用户。
- Part号码的范围是1~10000。如果超出这个范围，OSS将返回InvalidArgument的错误码。
- 每次上传part时都要把流定位到此次上传片开头所对应的位置。
- 每次上传part之后，OSS的返回结果会包含一个分片的ETag，您需要将它和块编号组合成PartEtag，保存在list中，后续完成分片上传需要用到。

相关参数:

- name {String} object 名称
- uploadId {String} 分片上传任务id
- partNo {Number} 分片序列号
- file {File} 整个上传文件对象
- start {Number} 分片在文件的启始位置，单位字节 e.g: 102400
- end {Number} 分片在文件的结束位置，单位字节 e.g: 204800
- [options] {Object} 额外参数

- [timeout] {Number} 超时设置

请求成功后的返回结果:

- res {Object} 返回内容，包括
  - status {Number} 返回状态码
  - headers {Object} 返回的头信息
  - size {Number} 返回内容的长度
  - rt {Number} 请求耗时 (ms)
- bucket {String} bucket 名称
- name {String} 储存在oss的object名称
- etag {String} part etag 内容e.g.: “5B3C1A2E053D763E1B002CC607C5A0FE”。后续交验需要。

范例：

```
  let name = 'object';
let result = await store.initMultipartUpload(name);
let uploadId = result.uploadId;
let file; //the data you want to upload, is a File or FileName(only in node)
//if file part is 10
let partSize = 100 * 1024;
let fileSize = 10 * partSize;//you need to calculate
let dones = [];
for (let i = 1; i <= 10; i++) {
let start = partSize * (i -1);
let end = Math.min(start + partSize, fileSize);
let part = await store.uploadPart(name, uploadId, i, file, start, end);
dones.push({
number: i,
etag: part.etag
});
console.log(part);
}

//end need to call completeMultipartUpload api
```

completeMultipartUpload(name, uploadId, parts[, options])

在将所有数据Part都上传完成后，必须调用Complete Multipart Upload API来完成整个文件的Multipart Upload。在执行该操作时，用户必须提供所有有效的数据Part的列表（包括part号码和ETAG）；OSS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，OSS将把这些数据part组合成一个完整的Object。

相关参数:

- name {String} object 名称
- uploadId {String} 分片上传任务id
- parts {Array} 有效的数据Part的列表（包括分片序列号和ETAG）:
  - number {Number} 分片序列号

- etag {String} object etag ", e.g.: "5B3C1A2E053D763E1B002CC607C5A0FE"
- [options] {Object} 额外参数
    - [timeout] {Number} 超时设置
    - [headers] {Object} 更多头信息, 详见 RFC 2616
    - [callback] {Object} callback回调设置。具体内容参见：Callback
        - url {String} 和oss server交互的回调服务器地址。对应于CallBack参数中的callbackUrl。必须有。
        - body {String} 发起回调时请求body的值。jason格式。对应于CallBack参数中的callbackBody。必须有。
        - [host] {String} 发起回调请求时Host头的值。对应于CallBack参数中的callbackHost。
        - [contentType] {String} 发起回调请求的Content-Type。对应于CallBack参数中的callbackBodyType 。
        - [customValue] {Object} 发起回调请求自定义参数。对应于CallBack参数中的callback-var。

请求成功后的返回结果:

- res {Object} 返回内容，包括
    - status {Number} 返回状态码
    - headers {Object} 返回的头信息
    - size {Number} 返回内容的长度
    - rt {Number} 请求耗时 (ms)
- bucket {String} bucket 名称
- name {String} 储存在oss的object名称
- etag {String} 上传文件的 etag 内容e.g.: "5B3C1A2E053D763E1B002CC607C5A0FE" 。
- data {Object} 回调服务器返回内容。

```
 //init multipart
let name = 'object';
try {
let result = await store.initMultipartUpload(name);
} catch (e) {
console.log(e);
}

//upload part
let file; //the data you want to upload, this example size is 10 * 100 * 1024
let fileSize;//you need to calculate
let partSize = 100 * 1024;//100kb
let done = [];
//if file part is 10
for (var i = 1; i <= 10; i++) {
let start = partSize * (i -1);
let end = Math.min(start + partSize, fileSize);
let data = file.slice(start, end);
try {
let part = await store.uploadPart(name, result.uploadId, i, data);
```

```
} catch (e) {
console.log(e);
}
console.log(part);
done.push({
number: i,
etag: part.res.headers.etag
});
}

//complete
let completeData = await store.completeMultipartUpload(name, result.uploadId, done);
console.log(completeData);
```

暂停分片上传

1. 分片上传暂停后，恢复上传时需要将之前记录的checkpoint参数传入即可.
2. 每次进行分片上传建议重新new一个新的OSS实例.

```
let OSS = require('ali-oss')

let ossConfig = {
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
}

let client = new OSS(ossConfig);

let tempCheckpoint;

// 定义上传方法
async funtion multipartUpload () {
try {
let result = await client.multipartUpload('object-key', 'local-file', { progress: async function (p, checkpoint) {
// 记录断点, 如果关闭了浏览器，然后重新启动继续上传的话，是不行的，请参考上边对file对象的描述
tempCheckpoint = checkpoint;
}
})
} catch(e){
console.log(e);
}
}

//开始上传
multipartUpload();

// 暂停分片上传方法
client.cancel();

// 恢复上传
let resumeclient = new OSS(ossConfig);
```

```
async function resumeUpload () {
try {
let result = await resumeclient.multipartUpload('object-key', 'local-file', {
progress: async function (p, checkpoint) {
tempCheckpoint = checkpoint;
},
checkpoint: tempCheckpoint
})
} catch (e) {
console.log(e);
}
}

resumeUpload();
```

我们用InitiateMultipartUploadRequest来指定上传文件的名字和所属存储空间（Bucket）。在InitiateMultipartUploadRequest中，您也可以设置ObjectMeta，但是不必指定其中的ContentLength 。initiateMultipartUpload 的返回结果中含有UploadId ，它是区分分片上传事件的唯一标识，在后面的操作中，我们将用到它。

下载文件

用户可以通过以下方式从OSS中下载文件：

- HTTP下载（浏览器下载）

浏览器中请使用signatureUrl方法生成可下载的HTTP地址，URL的有效时间默认为半个小时：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

let url = client.signatureUrl('object-key');
console.log(url);

let url = client.signatureUrl('object-key', {expires: 3600});
console.log(url);
```

管理文件

一个Bucket下可能有非常多的文件，SDK提供一系列的接口方便用户管理文件。

查看所有文件

通过list来列出当前Bucket下的所有文件。主要的参数如下：

- prefix 指定只列出符合特定前缀的文件
- marker 指定只列出文件名大于marker之后的文件
- delimiter 用于获取文件的公共前缀
- max-keys 用于指定最多返回的文件个数

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function list (dir) {
try {
// 不带任何参数，默认最多返回1000个文件
let result = await client.list();
console.log(result);

// 根据nextMarker继续列出文件
if (result.isTruncated) {
let result = await client.list({ marker : result.nextMarker });
}

// 列出前缀为'my-'的文件
let result = await client.list({
prefix: 'my-'
});
console.log(result);

// 列出前缀为'my-'且在'my-object'之后的文件
let result = await client.list({
prefix: 'my-',
marker: 'my-object'
});
console.log(result);
} catch (e) {
console.log(e);
}
```

```
}

list();
```

模拟目录结构

OSS是基于对象的存储服务，没有目录的概念。存储在一个Bucket中所有文件都是通过文件的key唯一标识，并没有层级的结构。这种结构可以让OSS的存储非常高效，但是用户管理文件时希望能够像传统的文件系统一样把文件分门别类放到不同的“目录”下面。通过OSS提供的“公共前缀”的功能，也可以很方便地模拟目录结构。公共前缀的概念请参考 列出Object

假设Bucket中已有如下文件：

```
foo/x
foo/y
foo/bar/a
foo/bar/b
foo/hello/C/1
foo/hello/C/2
...
foo/hello/C/9999
```

接下来我们实现一个函数叫listDir，列出指定目录下的文件和子目录：

```
let OSS = require('ali-oss');

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function listDir () {
try {
let result = await client.list({
prefix: dir,
delimiter: '/'
});

result.prefixes.forEach(function (subDir) {
console.log('SubDir: %s', subDir);
});
result.objects.forEach(function (obj) {
console.log(Object: %s', obj.name);
});
} catch (e) {
console.log(e);
}
}
```

运行结果如下：

```
> listDir('foo/')
=> SubDir: foo/bar/
SubDir: foo/hello/
Object: foo/x
Object: foo/y

> listDir('foo/bar/')
=> Object: foo/bar/a
Object: foo/bar/b

> listDir('foo/hello/C/')
=> Object: foo/hello/C/1
Object: foo/hello/C/2
...
Object: foo/hello/C/9999
```

文件元信息

向OSS上传文件时，除了文件内容，还可以指定文件的一些属性信息，称为“元信息”。这些信息在上传时与文件一起存储。

因为文件元信息在上传时附在HTTP Headers中， HTTP协议规定不能包含复杂字符。因此元信息**只能是简单的ASCII可见字符，不能包含换行。** 所有元信息的总大小不能超过8KB。

使用put，和multipartUpload时都可以通过指定meta参数来指定文件的元信息：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function multipartUploadWithMeta () {
try {
let result = await client.multipartUpload('object-key', 'local-file', {
meta: {
year: 2017,
people: 'mary'
}
});
console.log(result);
} catch (e) {
console.log(e);
}
}

multipartUploadWithMeta();
```

删除文件

通过delete来删除某个文件：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function delete () {
try {
let result = await client.delete('object-key');
console.log(result);
} catch (e) {
console.log(e);
}
}

delete();
```

批量删除文件

通过deleteMulti来删除一批文件，用户可以通过quiet参数来指定是否返回删除的结果：

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

async function deleteMulti () {
try {
let result = await client.deleteMulti(['obj-1', 'obj-2', 'obj-3']);
console.log(result);

let result = await client.deleteMulti(['obj-1', 'obj-2', 'obj-3'],{
quiet: true
});
console.log(result);
} catch (e) {
console.log(e);
}
}
```

```
deleteMulti();
```

自定义域名绑定

OSS支持用户将自定义的域名绑定到OSS服务上，这样能够支持用户无缝地将存储迁移到OSS上。例如用户的域名是my-domain.com，之前用户的所有图片资源都是形如http://img.my-domain.com/x.jpg的格式，用户将图片存储迁移到OSS之后，通过绑定自定义域名，仍可以使用原来的地址访问到图片：

- 开通OSS服务并创建Bucket
- 修改域名的DNS配置，增加一个CNAME记录，将img.my-domain.com指向OSS服务的endpoint（如my-bucket.oss-cn-hangzhou.aliyuncs.com）
- 在官网控制台将img.my-domain.com与创建的Bucket绑定
- 将图片上传到OSS的这个Bucket中

这样就可以通过原地址http://img.my-domain.com/x.jpg访问到存储在OSS上的图片。绑定自定义域名请参考自定义域名绑定

在使用SDK时，也可以使用自定义域名作为endpoint，这时需要将cname参数设置为true，如下面的例子：

```
let OSS = require('ali-oss')

let client = new OSS({
endpoint: '<Your endpoint>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
cname: true
});

client.useBucket('my-bucket')
```

注意：

- 使用CNAME时，无法使用list_buckets接口。（因为自定义域名已经绑定到某个特定的Bucket）

设置访问权限

OSS允许用户对Object分别设置访问权限，方便用户控制自己的资源可以被如何访问。

对于Object，有四种访问权限：

- default 继承所属的Bucket的访问权限，即与所属Bucket的权限值一样
- public-read-write 允许匿名用户读写该Object
- public-read 允许匿名用户读该Object
- private 不允许匿名访问，所有的访问都要经过签名

创建Object时，默认为default权限。之后用户可以通过putACL来设置Object的权限。

```
let OSS = require('ali-oss')

let client = new OSS({
region: '<Your region>'
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: '<Your bucket name>'
});

async function getACL () {
try {
let result = await client.getACL('my-object');
console.log(result.acl); // default

await client.putACL('my-object', 'public-read');
let result = await client.getACL('my-object');
console.log(result.acl); // public-read
} catch (e) {
console.log(e);
}
}

getACL();
```

需要注意的是：

1. 如果设置了Object的权限（非default），则访问该Object时进行权限认证时会优先判断Object的权限，而Bucket的权限设置会被忽略。

   允许匿名访问时（设置了public-read或者public-read-write权限），用户可以直接通过浏览器访问，例如：

   ```
   http://bucket-name.oss-cn-hangzhou.aliyuncs.com/object.jpg
   ```

更多关于访问权限控制的内容请参考 访问控制

图片处理

OSS图片处理，是OSS对外提供的海量、安全、低成本、高可靠的图片处理。用户将原始图片上传保存到OSS，通过简单的 RESTful 接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。图片处理提供图片处理接口，图片上传请使用上传接口。基于OSS图片处理，用户可以搭建自己的图片处理服务。

图片处理基础功能

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放、裁剪、旋转
- 图片效果
- 图片添加图片、文字、图文混合水印
- 自定义图片处理样式，在控制台的 图片处理 -> 样式管理 中定义
- 通过级联处理调用多个图片处理功能

图片处理使用

图片处理使用标准的 HTTP GET 请求来访问，所有的处理参数是编码在 URL 中的QueyString。

匿名访问

如果图片文件（Object）的访问权限是 公共读 ，如下表所示的权限，则可以匿名访问图片服务。

Bucket权限	Object权限
公共读私有写（public-read）或 公共读写（public-read-write）	默认（default）
任意权限	公共读私有写（public-read）或 公共读写（public-read-write）

通过如下格式的三级域名匿名访问图片处理：

```
http://bucket.<endpoint>/object?x-oss-process=image/action,param_value
```

- bucket：用户的存储空间（bucket）名称
- endpoint：用户存储空间所在数据中心的访问域名
- object：用户上传在OSS上的图片文件
- image：图片处理保留标志符
- action：用户对图片做的操作，如缩放、裁剪、旋转等
- param：用户对图片做的操作所对应的参数

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

自定义样式，使用如下格式的三级域名匿名访问图片处理：

```
http://bucket.<endpoint>/object?x-oss-process=x-oss-process=style/name
```

- style：用户自定义样式系统保留标志符
- name：自定义样式名称，即控制台定义样式的 规则名

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=style/oss-pic-style-w-100
```

通过级联处理，可以对一张图片顺序实施多个操作，格式如下：

```
http://bucket.<endpoint>/object?x-oss-process=image/action,param_value/action,param_value/...
```

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100/rotate,90
```

图片服务也支持HTTPS访问，例如：

```
https://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

授权访问

对私有权限的文件（Object），如下表所示的权限，必须通过授权才能访问图片服务。

Bucket权限	Object权限
私有读写（private）	默认权限（default）
任意权限	私有读写（private）

生成带签名的图片处理的URL代码如下：

```
let OSS = require('ali-oss');

let client = new OSS({
accessKeyId: '<accessKeyId>',
accessKeySecret: '<accessKeySecret>',
bucket: '<bucketName>',
```

```
endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});

// 过期时间10分钟, 图片处理式样"image/resize,w_300"
let signUrl = client.signatureUrl('example.jpg', {expires: 600, 'process' : 'image/resize,w_300'});
console.log("signUrl="+signUrl);
```

提示：

- 授权访问支持 自定义样式、 HTTPS、 级联处理
- 指定过期时间expires的单位是秒

SDK访问

对于任意权限的图片文件，都可以直接使用 SDK 访问图片、进行处理。

提示：

- SDK处理图片文件支持 自定义样式、 HTTPS、 级联处理

基础操作

图片处理的基础操作包括，获取图片信息、格式转换、缩放、裁剪、旋转、效果、水印等。

```
let OSS = require('ali-oss');

let client = new OSS({
accessKeyId: '<accessKeyId>',
accessKeySecret: '<accessKeySecret>',
bucket: '<bucketName>',
endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});

// 缩放
async function scale () {
try {
let result = await client.signatureUrl('example.jpg', {expires: 3600, process: 'image/resize,m_fixed,w_100,h_100'})
} catch (e) {
console.log(e);
}
}

scale();

// 裁剪
async function crop () {
try {
let result = await client.signatureUrl('example.jpg', {expires: 3600, process:
'image/crop,w_100,h_100,x_100,y_100,r_1'})
} catch (e) {
```

```
console.log(e);
}
}

crop();

// 旋转
async function rotate() {
try {
let result = await client.signatureUrl('example.jpg', {expires: 3600, process: 'image/rotate,90'})
} catch (e) {
console.log(e);
}
}

rotate();

// 锐化
async function sharpen () {
try {
let result = await client.signatureUrl('example.jpg', {expires: 3600, process: 'image/sharpen,100'})
} catch (e) {
console.log(e);
}
}

sharpen();

// 水印
async function watermark () {
try {
let result = await client.signatureUrl('example.jpg', {expires: 3600, process: 'image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ'})
} catch (e) {
console.log(e);
}
}

watermark();

// 格式转换
async function format () {
try {
let result = await client.signatureUrl('example.jpg', {expires: 3600, process: 'image/format,png'})
} catch (e) {
console.log(e);
}
}

format();

// 图片信息
async function info () {
try {
let result = await client.signatureUrl('example.jpg', {expires: 3600, process: 'image/info})
} catch (e) {
```

```
console.log(e)
}
}

info();
```

自定义样式

提示：需要事先到oss控制台添加自定义式样example-sytle

```
let OSS = require('ali-oss');

let client = new OSS({
accessKeyId: '<accessKeyId>',
accessKeySecret: '<accessKeySecret>',
bucket: '<bucketName>',
endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});

// 自定义样式
async function style () {
try {
let result = await client.signatureUrl('example.jpg', {expires: 3600, process: 'style/example-sytle'});
} catch (e) {
console.log(e);
}
}

style();
```

级联处理

```
let OSS = require('ali-oss');

let client = new OSS({
accessKeyId: '<accessKeyId>',
accessKeySecret: '<accessKeySecret>',
bucket: '<bucketName>',
endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});

// 级联处理
async function resize () {
try {
let result = await client.signatureUrl('example.jpg', {expires: 3600, process:
'image/resize,m_fixed,w_100,h_100/rotate,90'})
} catch (e) {
console.log(e);
}
}
```

```
resize();
```

图片处理工具

- 可视化图片处理工具 ImageStyleViever ，可以直观的看到OSS图片处理的结果
- OSS图片处理的功能、使用演示 页面

浏览器应用

浏览器支持

- IE(>=10)和Edge
- 主流版本的Chrome/Firefox/Safari
- 主流版本的Android/iOS/WindowsPhone

注意：

在IE中使用需要在引入SDK之前引入promise库：

浏览器低版本支持

SDK是通过File API进行文件操作，在一些较低版本的浏览器运行会出现问题。可以利用一些第三方的上传插件，通过对OSS API的实现，实现OSS的上传、下载等操作。具体范例请参见Web端直传实践。

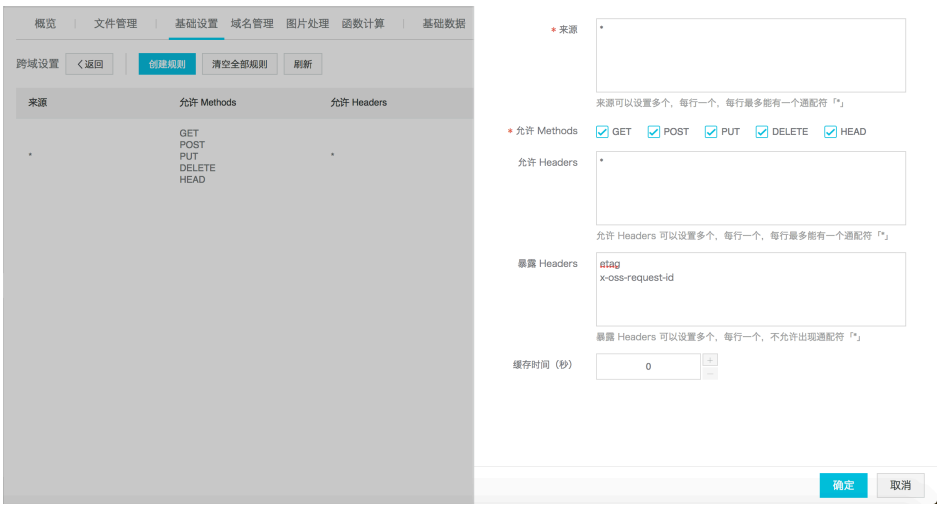
设置

Bucket设置

从浏览器中直接访问OSS，需要对Bucket的CORS属性进行设置，设置方式如下所示：

- 将 **来源** 设置成 *
- 将 **Method** 设置成 GET, POST, PUT, DELETE, HEAD
- 将 **Allowed Header** 设置成 *
- 将 **Expose Header** 设置成 etag

注意：请将该条CORS规则设置成所有CORS规则的第一条。



STS设置

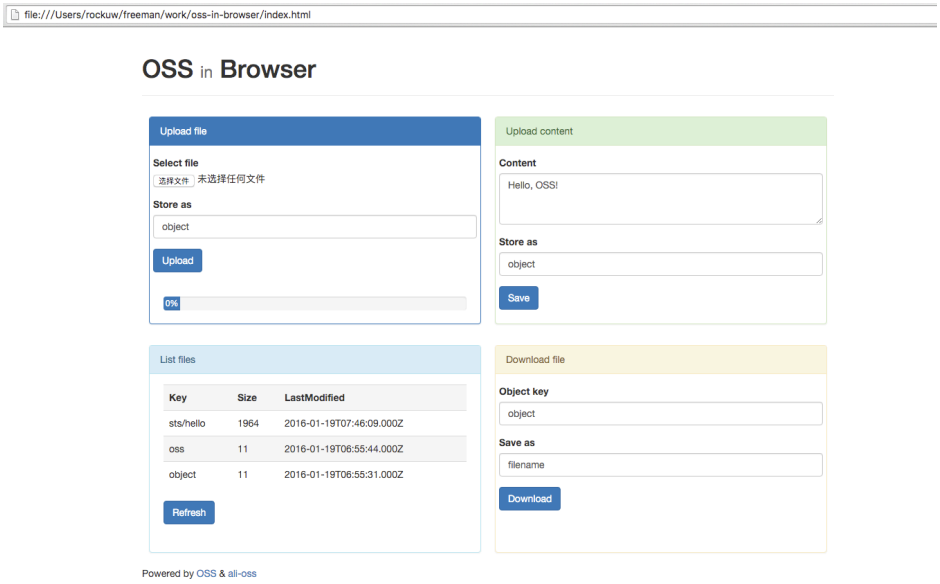
为了不在网页中暴露AccessKeyId和AccessKeySecret，我们采用STS进行临时授权。授权服务器由用户维护，只有认证的用户才能向授权服务器申请临时权限。设置子账号和STS的Policy，参考Node.js STS AppServer搭建自己的授权服务器。

例子

下面我们将使用SDK开发一个网页应用，包含4个功能：

- 上传文件
- 上传文本
- 列出文件
- 下载文件

完整的代码可以在oss-in-browser找到。最终的效果如下：



下面的介绍将以“上传文件”为例：

1. 创建网页

对于“上传文件”的功能，在网页上需要4个控件：

- 一个文件选择框，用于选择需要上传的文件
- 一个文本框，用于输入上传到OSS的Object名字
- 一个按钮，用于开始上传
- 一个进度条，用于显示上传的进度

下面的代码创建了这4个控件，其中class="xxx"是为了使用Bootstrap样式，可以先忽略。

```
<div class="form-group">
<label>Select file</label>
<input type="file" id="file" />
</div>
<div class="form-group">
<label>Store as</label>
<input type="text" class="form-control" id="object-key-file" value="object" />
</div>
<div class="form-group">
<input type="button" class="btn btn-primary" id="file-button" value="Upload" />
</div>
<div class="form-group">
<div class="progress">
<div id="progress-bar"
class="progress-bar"
role="progressbar"
aria-valuenow="0"
aria-valuemin="0"
aria-valuemax="100" style="min-width: 2em;">
0%
</div>
```

```
</div>
</div>
```

2. 添加样式

接下来为网页添加一些样式，让它更好看一些，这里我们用到了Bootstrap。在网页的<head>标签中加入样式：

```
<head>
<title>OSS in Browser</title>
<link rel="stylesheet" href="bootstrap.min.css" />
</head>
```

其中bootstrap.min.css可以在bootstrap的官网下载。

3. 添加脚本

到目前为止，网页是静态的，点击上面的按钮也不会有反应。接下来的重点是为它添加一些JavaScript脚本，让它能够上传文件/下载文件/列出文件。

首先在<head>标签中引入SDK的开发包：

```
<head>
<title>OSS in Browser</title>
<link rel="stylesheet" href="bootstrap.min.css" />
<script type="text/javascript" src="http://gosspublic.alicdn.com/aliyun-oss-sdk-x.x.x.min.js"></script>
<script type="text/javascript" src="app.js"></script>
</head>
```

其中app.js是真正执行上传文件的代码，内容如下：

```
const appServer = 'http://localhost:3000';
const bucket = 'js-sdk-bucket-sts';
const region = 'oss-cn-hangzhou';

const urllib = OSS.urllib;

const applyTokenDo = function (func) {
const url = appServer;
return urllib.request(url, {
method: 'GET'
}).then(function (result) {
const creds = JSON.parse(result.data);
const client = new OSS({
region: region,
accessKeyId: creds.AccessKeyId,
accessKeySecret: creds.AccessKeySecret,
stsToken: creds.SecurityToken,
bucket: bucket
});
```

```
return func(client);
});
};

let currentCheckpoint;
const progress = async function progress(p, checkpoint) {
currentCheckpoint = checkpoint;
const bar = document.getElementById('progress-bar');
bar.style.width = `${Math.floor(p * 100)}%`;
bar.innerHTML = `${Math.floor(p * 100)}%`;
};

let uploadFileClient;
const uploadFile = function (client) {
if (!uploadFileClient || Object.keys(uploadFileClient).length === 0) {
uploadFileClient = client;
}

const file = document.getElementById('file').files[0];
const key = document.getElementById('object-key-file').value.trim() || 'object';
console.log(`${file.name} => ${key}`);

const options = {
progress,
partSize: 100 * 1024,
meta: {
year: 2017,
people: 'test',
},
};

return client.multipartUpload(key, file, options).then( (res) => {
console.log('upload success: %j', res);
currentCheckpoint = null;
uploadFileClient = null;
}).catch((err) => {
if (uploadFileClient && uploadFileClient.isCancel()) {
console.log('stop-upload!');
} else {
console.error(err);
}
});
};

window.onload = function () {
document.getElementById('file-button').onclick = function () {
applyTokenDo(uploadFile);
}
};
```

上传一个文件分为以下步骤：

1. 向授权服务器申请临时权限。其中授权服务器是**网站开发者**构建的用于向**终端用户**提供临时授权的服务。开发者可以在授权时为不同的用户提供不同的权限（参考OSS使用STS)。授权服务器可以参考这

个例子，为了简便，例子中授权服务器直接向用户返回临时凭证。
2. 使用临时密钥创建OSS Client。
3. 通过multipartUpload上传选中的文件，并通过progress参数设置进度条。

4. 其他功能

上传文本内容、获取文件列表、下载文件等功能请参考代码示例：OSS in Browser。

异常

使用SDK时如果请求出错，会有相应的异常抛出。服务器端异常通常会包含以下信息：

- status: 出错请求的HTTP状态码
- code: OSS的错误码
- message: OSS的错误信息
- requestId: 标识该次请求的UUID；当您无法解决问题时，可以凭这个RequestId来请求OSS开发工程师的帮助

OSS中常见的错误信息请参考 OSS错误响应

调试

在浏览器环境中您可以通过在console中设置localStorage.debug变量来开启调试：

```
localStorage.debug = 'ali-oss'
```

常见问题

如何调用 STS

浏览器是不受信任的环境，如果把 AccessKeyId 和 AccessKeySecret 直接保存在浏览器端，存在极高的风险。建议在浏览器环境下使用 STS 模式进行 OSS 接口调用。

浏览器中使用 STS Server 相关文档。

获取 STS token 后，就可进行 SDK 初始化操作。

```
<script type="text/javascript">
$.ajax("http://your_sts_server/",{method: 'GET'},function (err, result) {
let client = new OSS({
accessKeyId: result.AccessKeyId,
accessKeySecret: result.AccessKeySecret,
stsToken: result.SecurityToken,
endpoint: '<oss endpoint>',
bucket: '<Your bucket name>'
});
});
</script>
```

如何 HTTPS 访问

初始化 SDK 时，可传入以下几个参数：

- region: 参数是指您申请 OSS 服务时的区域，例如oss-cn-hangzhou。完整的区域列表可以在 OSS 服务节点查看。
- internal: 配合region使用，如果指定 internal 为 true，则访问内网节点。
- secure: 配合region使用，如果指定了secure为true，则使用 HTTPS 访问。
- endpoint: 例如http://oss-cn-hangzhou.aliyuncs.com，如果指定了endpoint，则region会被忽略，endpoint可以指定HTTPS，也可以是IP形式。

浏览器跨域问题如何解决

在浏览器中使用 SDK 前，先要设置 Bucket 的 CORS 属性。具体步骤，请查看相关文档。

如何设置上传文件的用户自定义数据(meta)，文件类型(mime)和请求头(header)

参考浏览器分片上传

关于浏览器端断点续传的说明

可以将checkpoint保存到浏览器的localstorage，下次再调用的时候传入checkpoint参数，就可以实现断点续传功能。

如何获取上传进度

使用分片上传时，可获取上传进度。相关文档。

如何获取下载进度

浏览器中无法获取进度，可调用 signatureUrl 方法，获取下载地址，可查看相关文档。

如何上传文件到指定目录

给要上传的 object 名称前加指定目录前缀即可，可参考: OSS 和文件系统对比。

```
let OSS = require('ali-oss')
let client = new OSS({
region: '<Your region>',
accessKeyId: '<Your AccessKeyId>',
accessKeySecret: '<Your AccessKeySecret>',
bucket: 'Your bucket name'
});

client.multipartUpload('base-dir/' +'object-key', 'local-file', {
progress: async function (p) {
console.log('Progress: ' + p);
}
});
console.log(result);
}).catch((err) => {
console.log(err);
});
```

如何上传base64编码的图片

base64先转码成指定格式图片，然后调用OSS上传接口进行上传，具体可参考Github示例。

```
/**
* base64 to file
* @param dataurl base64 content
* @param filename set up a meaningful suffix, or you can set mime type in options
* @returns {File|*}
*/
const dataURLtoFile = function dataURLtoFile(dataurl, filename) {
const arr = dataurl.split(',');
const mime = arr[0].match(/:(.*?);/)[1];
const bstr = atob(arr[1]);
let n = bstr.length;
const u8arr = new Uint8Array(n);
while (n--) {
u8arr[n] = bstr.charCodeAt(n);
}
return new Blob([u8arr], { type: mime });// if env support File, also can use this: return new File([u8arr], filename, {
type: mime });
};
```

```
// client表示OSS client实例
const uploadBase64Img = function uploadBase64Img(client) {
// base64格式的内容
const base64Content = 'data:image:xxxxxxxxxxxxx';
const filename = 'img.png';
const imgfile = dataURLtoFile(base64Content, filename);
//key表示上传的object key ,imgFile表示dataURLtoFile处理后返回的图片
client.multipartUpload(key, imgfile).then((res) => {
console.log('upload success: %j', res);
}).catch((err) => {
console.error(err);
});
};
```

如何限制上传文件的大小

在浏览器中可以根据document.getElementById（“file”).files[0].size 获取上传文件的大小（字节数），可参考web直传的post请求。

如何获取object的签名URL

可调用 signatureUrl 方法，获取下载地址，可查看相关文档。

如何使用sdk生成的签名URL并进行资源上传

签名URL常用于授权给第三方进行资源的下载和上传操作。下载请参见上一条。sdk中提供signatureUrl API，用于返回一个经过签名的url，用户直接使用这个url上传或者下载资源即可。利用签名URL上传资源请参考sdk工程示例：签名url上传资源示例

如何使用表单上传方式上传资源到OSS服务器

可参考 Web 端直传实践。

常见错误参考

- SDK 开启异常日志
- OSS 常见错误

PHP-SDK

前言

本文档基于PHP 5.6.22版本编写。

环境准备

安装环境

您需要安装PHP和cURL扩展。

- 在Windows系统中，请参见Windows下编译使用阿里云 OSS PHP SDK来安装PHP和cURL扩展。在Windows环境中，如果提示找不到指定模块，请在php.ini文件中指定extension_dir为C:/Windows/System32/。
- 在Ubuntu系统中，请使用apt-get包管理器安装PHP的cURL扩展 sudo apt-get install php-curl。
- 在CentOS系统中，请使用yum包管理器安装PHP的cURL扩展 sudo yum install php-curl。

查看版本

PHP 5.3+
您可以通过php -v命令查看当前的PHP版本。

cURL 扩展
您可以通过php -m命令查看cURL扩展是否已经安装好。

SDK下载

- 通过GitHub下载
- 历史版本下载

更多信息请参见OSS API文档。

说明：建议您使用最新版本的SDK。OSS PHP SDK 2.0.0以下版本的文档请从此处下载。

SDK安装

请参见OSS PHP SDK安装。

示例程序

您可以通过以下步骤运行示例程序：

解压下载的SDK包。

修改samples目录中的Config.php文件：

- OSS_ACCESS_ID是您从OSS获得的AccessKeyId。
- OSS_ACCESS_KEY是您从OSS获得的AccessKeySecret。
- OSS_ENDPOINT是您选定的OSS数据中心的访问域名，如 http://oss-cn-hangzhou.aliyuncs.com。
- OSS_TEST_BUCKET是您运行示例程序所使用的存储空间。示例程序会在这个存储空间中创建一些文件。

在samples目录中执行php RunAll.php，也可以单独运行某个示例文件。

示例程序包含以下内容：

示例文件	示例内容
Object.php	文件的相关操作，包括上传、下载、复制、删除、列举、元信息等
MultipartUpload.php	大文件上传和分片上传
Signature.php	URL签名授权访问
Callback.php	上传回调
Image.php	图片处理
LiveChannel.php	LiveChannel的相关操作
Bucket.php	存储空间的相关操作，包括创建、删除、列举、权限等
BucketLifecycle.php	设置、读取和清除存储空间的生命周期
BucketLogging.php	设置、读取和清除存储空间的日志
BucketReferer.php	设置、读取和清除存储空间的防盗链
BucketWebsite.php	设置、读取和清除存储空间的静态网站托管
BucketCors.php	设置、读取和清除存储空间的跨域资源访问

安装

有三种方式安装SDK：

- composer方式
- phar方式
- 源码方式

composer方式

composer方式安装SDK的步骤如下：

在项目的根目录运行composer require aliyuncs/oss-sdk-php，或者在composer.json文件中添加依赖关系如下：

```
"require": {
"aliyuncs/oss-sdk-php": "~2.x.x"
}
```

运行composer install，安装依赖。安装完成后，目录结构如下：

```
.
├── app.php
├── composer.json
├── composer.lock
└── vendor
```

其中app.php是您的应用程序，vendor/目录下包含了所依赖的库。您需要在app.php中添加依赖关系如下：

```
require_once __DIR__ . '/vendor/autoload.php';
```

说明:

- 如果您的项目中已经引用过autoload.php，则添加了SDK的依赖关系之后，不需要再次引入。
- 如果使用composer出现网络错误，可以使用composer中国区的镜像源。方法是在命令行执行 composer config -g repositories.packagist composer http://packagist.phpcomposer.com。

phar方式

phar方式安装SDK的步骤如下：

在GitHub中选择相应的版本并下载打包好的phar文件。

在代码中引入phar文件：

```
require_once '/path/to/oss-sdk-php.phar';
```

源码方式

源码方式安装SDK的步骤如下：

在GitHub中选择相应版本并下载打包好的zip文件。

解压后的根目录中包含一个autoload.php文件，在代码中引入此文件：

```
require_once '/path/to/oss-sdk/autoload.php';
```

初始化

OssClient是OSS的PHP客户端，用于管理存储空间和文件等OSS资源。

新建OssClient

新建OSSClient时，需要指定Endpoint。有关Endpoint的更多信息，请参见访问域名和数据中心和自定义访问域名。

使用OSS域名新建OssClient

以下代码用于使用OSS域名新建OssClient：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
```

```
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
} catch (OssException $e) {
print $e->getMessage();
}
```

使用自定义域名新建OssClient

以下代码用于使用自定义域名新建OssClient：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
$endpoint = "<yourEndpoint>";

try {
// true为开启CNAME。CNAME是指将自定义域名绑定到存储空间上。
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, true);
} catch (OssException $e) {
print $e->getMessage();
}
```

说明：使用自定义域名时，无法使用listBuckets方法。

使用STS新建OssClient

以下代码用于使用STS新建OssClient：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$securityToken = "<yourSecurityToken>";

try {
$ossClient = new OssClient(
$accessKeyId, $accessKeySecret, $endpoint, false, $securityToken);
} catch (OssException $e) {
print $e->getMessage();
}
```

更多信息请参见RAM和STS介绍和授权访问。

使用代理服务器新建OssClient

PHP 5.3以上版本支持使用代理服务器新建OssClient，代码如下：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// 代理服务器地址，例如http://<您的用户名>:<您的密码>@<代理ip>:<代理端口>。
```

```
$requestProxy = "<yourRequestProxy>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com>";

try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false, false, $requestProxy);
} catch (OssException $e) {
print $e->getMessage();
}
```

配置网络参数

以下代码用于配置OssClient网络参数：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

try {
$ossClient = new OssClient(
$accessKeyId, $accessKeySecret, $endpoint);
// 设置Socket层传输数据的超时时间，单位秒，默认5184000秒。
$ossClient->setTimeout(3600);
// 设置建立连接的超时时间，单位秒，默认10秒。
$ossClient->setConnectTimeout(10);
} catch (OssException $e) {
print $e->getMessage();
}
```

快速入门

本节介绍如何快速使用OSS PHP SDK完成常见操作，如创建存储空间、上传文件、下载文件等。

创建存储空间

以下代码用于创建存储空间：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket = "<yourBucketName>";

try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
$ossClient->createBucket($bucket);
} catch (OssException $e) {
print $e->getMessage();
}
```

存储空间的命名规范，请参见基本概念中的命名规范。创建存储空间的信息，请参见管理存储空间。

上传文件

以下代码用于上传文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
```

```
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket= " <yourBucketName>";
// 文件名称
$object = " <yourObjectName>";
$content = "Hi, OSS.";

try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
$ossClient->putObject($bucket, $object, $content);
} catch (OssException $e) {
print $e->getMessage();
}
```

上传文件详情请参见上传文件。

下载文件

以下代码用于下载文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket= "<yourBucketName>";
// 文件名称
$object = "<yourObjectName>";

try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
$content = $ossClient->getObject($bucket, $object);
print("object content: " . $content);
} catch (OssException $e) {
print $e->getMessage();
}
```

下载文件详情请参见下载文件。

列举文件

以下代码用于列举指定存储空间下的文件。默认列举100个文件。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket= "<yourBucketName>";

try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$listObjectInfo = $ossClient->listObjects($bucket);
$objectList = $listObjectInfo->getObjectList();
if (!empty($objectList)) {
foreach ($objectList as $objectInfo) {
print($objectInfo->getKey() . "\t" . $objectInfo->getSize() . "\t" . $objectInfo->getLastModified() . "\n");
}
}
} catch (OssException $e) {
print $e->getMessage();
}
```

列举功能详情请参见管理文件中的列出存储空间中的文件。

删除文件

以下代码用于删除指定文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
```

```
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket= "<yourBucketName>";
// 文件名称
$object = "<yourObjectName>";

try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
$ossClient->deleteObject($bucket, $object);
} catch (OssException $e) {
print $e->getMessage();
}
```

删除文件详情请参见管理文件。

管理存储空间

存储空间（Bucket）是对象（Object）的容器，对象必须隶属于某个存储空间。

以下场景的完整代码请参见GitHub。

创建存储空间

以下代码用于创建存储空间：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;
```

```
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";
try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
// 设置存储空间的存储类型为低频访问类型，默认是标准类型。
$options = array(
OssClient::OSS_STORAGE => OssClient::OSS_STORAGE_IA
);
// 设置存储空间的权限为公共读，默认是私有读写。
$ossClient->createBucket($bucket, OssClient::OSS_ACL_TYPE_PUBLIC_READ, $options);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

存储空间的命名规范，请参见基本概念中的命名规范。

判断存储空间是否存在

以下代码用于判断指定的存储空间是否存在：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";
try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$res = $ossClient->doesBucketExist($bucket);
```

```
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
if ($res === true) {
print(__FUNCTION__ . ": OK" . "\n");
} else {
print(__FUNCTION__ . ": FAILED" . "\n");
}
```

列举存储空间

以下代码用于列举存储空间：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
try{
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$bucketListInfo = $ossClient->listBuckets();
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
$bucketList = $bucketListInfo->getBucketList();
foreach($bucketList as $bucket) {
print($bucket->getLocation() . "\t" . $bucket->getName() . "\t" . $bucket->getCreatedate() . "\n");
}
```

设置存储空间的访问权限

存储空间的访问权限（ACL）有以下三类：

权限	PHP SDK对应值

private（私有读写）	OssClient::OSS_ACL_TYPE_PRIVATE
public-read（公共读私有写	OssClient::OSS_ACL_TYPE_PUBLIC_READ
public-read-write（公共读写）	OssClient::OSS_ACL_TYPE_PUBLIC_READ_WRITE

以下代码用于设置存储空间的访问权限：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";
// 设置存储空间的权限为私有读写。
$acl = OssClient::OSS_ACL_TYPE_PRIVATE;
try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$ossClient->putBucketAcl($bucket, $acl);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

获取存储空间的访问权限

以下代码用于获取存储空间的访问权限：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}
```

```
use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";

try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$res = $ossClient->getBucketAcl($bucket);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
print('acl: ' . $res);
```

获取存储空间的地域

以下代码用于获取存储空间的地域（称为Region或Location）：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";

try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$Regions = $ossClient->getBucketLocation($bucket);
} catch (OssException $e) {
```

```
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
var_dump($Regions);
```

关于地域的详细信息请参见基本概念中的地域。

获取存储空间元信息

以下代码用于获取存储空间元信息：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";

try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$Metas = $ossClient->getBucketMeta($bucket);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
var_dump($Metas);
```

删除存储空间

删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。

以下代码用于删除存储空间：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";

try{
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$ossClient->deleteBucket($bucket);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

上传文件

在OSS中，操作的基本数据单元是文件（Object）。OSS PHP SDK提供了丰富的文件上传方式：

- 字符串上传：文件最大不能超过5GB。
- 文件上传：文件最大不能超过5GB。
- 追加上传：文件最大不能超过5GB。
- 分片上传：文件最大不能超过48.8TB。当文件较大时，请使用分片上传。

说明： 各种上传方式的适用场景请参见开发指南中的上传文件。

字符串上传

以下代码用于上传字符串到OSS中：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket= "<yourBucketName>";
// 文件名称
$object = "<yourObjectName>";
// 文件内容
$content = "Hello OSS";
try{
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$ossClient->putObject($bucket, $object, $content);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

文件上传

以下代码用于上传文件到OSS中：

```
<?php
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
```

```
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket= "<yourBucketName>";
// 文件名称
$object = "<yourObjectName>";
// <yourLocalFile>由本地文件路径加文件名包括后缀组成，例如/users/local/myfile.txt
$filePath = "<yourLocalFile>";

try{
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$ossClient->uploadFile($bucket, $object, $filePath);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

追加上传

追加上传支持两种上传类型，分别为字符串和文件。追加类型的文件（Append Object）暂时不支持copyObject操作。

追加上传字符串

以下代码用于追加上传字符串到OSS中：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket= "<yourBucketName>";
// 文件名称
$object = "<yourObjectName>";
// 文件内容
```

```
$content_array = array('Hello OSS', 'Hi OSS', 'OSS OK');
try{
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$position = $ossClient->appendObject($bucket, $object, $content_array[0], 0);
$position = $ossClient->appendObject($bucket, $object, $content_array[1], $position);
$position = $ossClient->appendObject($bucket, $object, $content_array[2], $position);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

追加上传文件

以下代码用于追加上传本地文件到OSS：

```
 <?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket= "<yourBucketName>";
// 文件名称
$object = "<yourObjectName>";
// 本地文件1
$filePath = "<yourLocalFile1>";
// 本地文件2
$filePath1 = "<yourLocalFile2>";
try{
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$position = $ossClient->appendFile($bucket, $object, $filePath, 0);
$position = $ossClient->appendFile($bucket, $object, $filePath1, $position);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

分片上传

分片上传分为以下三个步骤：

1. 初始化一个分片上传事件。
   调用$ossClient->initiateMultipartUpload方法返回OSS创建的全局唯一的uploadId。

   上传分片。
   调用$ossClient->uploadPart方法上传分片数据。需要注意：

   - 对于同一个uploadId，分片号（partNumber）标识了该分片在整个文件内的相对位置。如果使用同一个分片号上传了新的数据，那么OSS上这个分片已有的数据将会被覆盖。
   - 除了最后一块Part以外，其他的Part最小为100KB。最后一块Part没有大小限制。

   完成分片上传。
   调用$ossClient->completeMultipartUpload方法将所有分片合并成完整的文件。

以下通过一个完整的示例对分片上传的流程进行逐步解析：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Core\OssUtil;

/// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$uploadFile = "<yourLocalFile>";

/**
* 步骤1：初始化一个分片上传事件，获取uploadId。
*/
try{
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

//返回uploadId，它是分片上传事件的唯一标识，您可以根据这个ID来发起相关的操作，如取消分片上传、查询分片上传等。
$uploadId = $ossClient->initiateMultipartUpload($bucket, $object);
} catch(OssException $e) {
printf(__FUNCTION__ . ": initiateMultipartUpload FAILED\n");
```

```
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": initiateMultipartUpload OK" . "\n");
/*
* 步骤2：上传分片。
*/
$partSize = 10 * 1024 * 1024;
$uploadFileSize = filesize($uploadFile);
$pieces = $ossClient->generateMultiuploadParts($uploadFileSize, $partSize);
$responseUploadPart = array();
$uploadPosition = 0;
$isCheckMd5 = true;
foreach ($pieces as $i => $piece) {
$fromPos = $uploadPosition + (integer)$piece[$ossClient::OSS_SEEK_TO];
$toPos = (integer)$piece[$ossClient::OSS_LENGTH] + $fromPos - 1;
$upOptions = array(
$ossClient::OSS_FILE_UPLOAD => $uploadFile,
$ossClient::OSS_PART_NUM => ($i + 1),
$ossClient::OSS_SEEK_TO => $fromPos,
$ossClient::OSS_LENGTH => $toPos - $fromPos + 1,
$ossClient::OSS_CHECK_MD5 => $isCheckMd5,
);
// MD5校验。
if ($isCheckMd5) {
$contentMd5 = OssUtil::getMd5SumForFile($uploadFile, $fromPos, $toPos);
$upOptions[$ossClient::OSS_CONTENT_MD5] = $contentMd5;
}
try {
// 上传分片。
$responseUploadPart[] = $ossClient->uploadPart($bucket, $object, $uploadId, $upOptions);
} catch(OssException $e) {
printf(__FUNCTION__ . ": initiateMultipartUpload, uploadPart - part#{$i} FAILED\n");
printf($e->getMessage() . "\n");
return;
}
printf(__FUNCTION__ . ": initiateMultipartUpload, uploadPart - part#{$i} OK\n");
}
// $uploadParts是由每个分片的ETag和分片号（PartNumber）组成的数组。
$uploadParts = array();
foreach ($responseUploadPart as $i => $eTag) {
$uploadParts[] = array(
'PartNumber' => ($i + 1),
'ETag' => $eTag,
);
}
/**
* 步骤3：完成上传。
*/
try {
// 在执行该操作时，需要提供所有有效的$uploadParts。OSS收到提交的$uploadParts后，会逐一验证每个分片的有效性。
当所有的数据分片验证通过后，OSS将把这些分片组合成一个完整的文件。
$ossClient->completeMultipartUpload($bucket, $object, $uploadId, $uploadParts);
} catch(OssException $e) {
printf(__FUNCTION__ . ": completeMultipartUpload FAILED\n");
printf($e->getMessage() . "\n");
```

```
return;
}
printf(__FUNCTION__ . ": completeMultipartUpload OK\n");
```

分片上传的完整代码请参见：GitHub。

取消分片上传

您可以调用$ossClient->abortMultipartUpload方法来取消分片上传事件。当一个分片上传事件被取消后，无法再使用这个uploadId做任何操作，已经上传的分片数据会被删除。

以下代码用于取消分片上传事件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$upload_id = "<yourUploadId>";

try{
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$ossClient->abortMultipartUpload($bucket, $object, $upload_id);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

分片上传本地文件

以下代码用于分片上传本地文件到OSS中：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
```

```
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$file = "<yourLocalFile>";
try{
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$ossClient->multiuploadFile($bucket, $object, $file);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

分片上传目录

以下代码用于分片上传本地目录（包含此目录下所有文件）到OSS中：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$localDirectory = ".";
$prefix = "samples/codes";
try {
```

```
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$ossClient->uploadDir($bucket, $prefix, $localDirectory);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

列举已上传的分片

以下代码用于列举已上传的分片：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$uploadId = "<yourUploadId>";

try{
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$listPartsInfo = $ossClient->listParts($bucket, $object, $uploadId);
foreach ($listPartsInfo->getListPart() as $partInfo) {
print($partInfo->getPartNumber() . "\t" . $partInfo->getSize() . "\t" . $partInfo->getETag() . "\t" . $partInfo-
>getLastModified() . "\n");
}
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

列举分片上传事件

以下代码用于列举分片上传事件：

```
<?php
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
require_once __DIR__ . '/../vendor/autoload.php';
}

use OSS\OssClient;
use OSS\Core\OssException;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维
，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";

$options = array(
'delimiter' => '/',
'max-uploads' => 100,
'key-marker' => '',
'prefix' => '',
'upload-id-marker' => ''
);
try {
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

$listMultipartUploadInfo = $ossClient->listMultipartUploads($bucket, $options);
} catch(OssException $e) {
printf(__FUNCTION__ . ": listMultipartUploads FAILED\n");
printf($e->getMessage() . "\n");
return;
}
printf(__FUNCTION__ . ": listMultipartUploads OK\n");
$listUploadInfo = $listMultipartUploadInfo->getUploads();
var_dump($listUploadInfo);
```

$options的参数说明如下：

参数	说明
delimiter	用于对文件名称进行分组的一个字符。所有名称包含指定的前缀且第一次出现delimiter字符之间的文件作为一组元素。
key-marker	所有文件名称的字母序大于key-marker参数值的分片上传事件。可以与upload-id-marker参数一同使用来指定返回结果的起始位置。
max-uploads	限定此次返回分片上传事件的最大数目，默认值和最大值均为1000。
prefix	限定返回的文件名称必须以指定的prefix作为前缀。注意使用prefix查询时，返回的文件名称中仍会

	包含prefix。
upload-id-marker	与key-marker参数一同使用来指定返回结果的起始位置。 如果未设置key-marker参数，则此参数无效。如果设置了key-marker参数，则查询结果中包含： - 名称的字母序大于key-marker参数值的所有文件。 - 文件名称等于key-marker参数值且uploadId比upload-id-marker参数值大的所有分片上传事件。

下载文件

OSS PHP SDK提供了丰富的文件下载方式，您可以通过以下方式从OSS下载文件（Object）：

- 下载文件到本地文件
- 下载文件到本地内存
- 范围下载
- 条件下载

文件下载的完整代码请参见GitHub。

下载文件到本地文件

以下代码可以下载OSS文件到本地文件：

```
<?php
/**
* get_object_to_local_file
*
* 将Object下载到指定的文件。
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getObjectToLocalFile($ossClient, $bucket)
{
$object = "oss-php-sdk-test/download-test-object-name.txt";
$localfile = "download-test-object-name.txt";
$options = array(
OssClient::OSS_FILE_DOWNLOAD => $localfile,
);
try{
$ossClient->putObject($bucket, $object, "test-content");
```

```
$ossClient->getObject($bucket, $object, $options);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK, please check localfile: 'upload-test-object-name.txt'" . "\n");
}
```

下载文件到本地内存

以下代码可以下载文件到本地内存：

```
<?php
/**
* 获取Object的内容。
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getObject($ossClient, $bucket)
{
$object = "oss-php-sdk-test/upload-test-object-name.txt";
try{
$ossClient->putObject($bucket, $object, "test-content");
$content = $ossClient->getObject($bucket, $object);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

范围下载

如果OSS文件较大，而您只需要其中一部分数据，可以使用范围下载，下载指定范围的数据。

例如指定的下载范围是0-100，则返回第0到第100，共101个字节的数据，即闭区间[0, 100]，包含两端字节。如果指定的范围无效，则下载整个文件。范围下载既可以下载到本地内存，又可以下载到本地文件。

以下代码可以下载文件[0, 4]的内容到本地内存：

```
<?php
/**
* 范围下载到本地内存。
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
```

```
* @return null
*/
function getObject($ossClient, $bucket)
{
$object = "oss-php-sdk-test/upload-test-object-name.txt";
$options = array(OssClient::OSS_RANGE => '0-4');
try{
$ossClient->putObject($bucket, $object, "test-content");
$content = $ossClient->getObject($bucket, $object, $options);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

以下代码可以下载文件[0, 4]的内容到本地文件：

```
<?php
/**
* 范围下载到本地文件。
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getObject($ossClient, $bucket)
{
$object = "oss-php-sdk-test/upload-test-object-name.txt";
$localfile = "download-test-object-name.txt";
$options = array(
OssClient::OSS_RANGE => '0-4',
OssClient::OSS_FILE_DOWNLOAD => $localfile,
);
try{
$ossClient->putObject($bucket, $object, "test-content");
$ossClient->getObject($bucket, $object, $options);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

条件下载

下载文件时，可以指定一个或多个限定条件，满足限定条件则下载，不满足则报错且不下载文件。可以使用的限定条件如下：

参数	描述	PHP SDK对应值

If-Modified-Since	如果该参数值早于实际修改时间，则正常传输文件。否则返回错误。	OssClient::OSS_IF_MODIFIED_SINCE
If-Unmodified-Since	如果该参数值等于或者晚于文件实际修改时间，则正常传输文件；否则返回错误。	OssClient::OSS_IF_UNMODIFIED_SINCE
If-Match	如果传入的ETag和Object的ETag匹配，则正常传输文件；否则返回错误。	OssClient::OSS_IF_MATCH
If-None-Match	如果传入的ETag值和Object的ETag不匹配，则正常传输文件；否则返回错误。	OssClient::OSS_IF_NONE_MATCH

说明：

- 如果If-Modified-Since的参数值不符合规范，直接返回文件，并返回200 OK。
- If-Modified-Since和If-Unmodified-Since可以同时存在，If-Match和If-None-Match也可以同时存在。
- 如果包含If-Unmodified-Since并且不符合或者包含If-Match并且不符合，返回412 Precondition Failed。
- 如果包含If-Modified-Since并且不符合或者包含If-None-Match并且不符合，返回304 Not Modified。

条件下载既可以下载文件到本地内存又可以下载文件到本地文件。条件下载的示例代码如下：

```
<?php
/**
* 如果文件在指定的时间之后修改过，则下载文件。
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getObject($ossClient, $bucket)
{
$object = "oss-php-sdk-test/upload-test-object-name.txt";
try{
$options = array(
OssClient::OSS_HEADERS => array(
OssClient::OSS_IF_MODIFIED_SINCE => "Fri, 13 Nov 2015 14:47:53 GMT"),
);
$content = $ossClient->getObject($bucket, $object, $options);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

说明： 文件的ETag值可以通过$ossClient->getObjectMeta获取。

管理文件

在OSS中，用户可以通过一系列的接口管理存储空间(Bucket)中的文件(Object)，比如ListObjects，DeleteObject，CopyObject，DoesObjectExist等。

提示：

- 以下场景的完整代码路径：GitHub

列出存储空间中的文件

您可以使用OssClient::listObjects列出存储中间中的文件:

```
<?php
/**
* 列出Bucket内所有目录和文件， 根据返回的nextMarker循环调用listObjects接口得到所有文件和目录
*
* @param OssClient $ossClient OssClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function listAllObjects($ossClient, $bucket)
{
//构造dir下的文件和虚拟目录
for ($i = 0; $i < 100; $i += 1) {
$ossClient->putObject($bucket, "dir/obj" . strval($i), "hi");
$ossClient->createObjectDir($bucket, "dir/obj" . strval($i));
}

$prefix = 'dir/';
$delimiter = '/';
$nextMarker = '';
$maxkeys = 30;
while (true) {
$options = array(
'delimiter' => $delimiter,
'prefix' => $prefix,
'max-keys' => $maxkeys,
'marker' => $nextMarker,
);
var_dump($options);
try {
$listObjectInfo = $ossClient->listObjects($bucket, $options);
```

```
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
// 得到nextMarker，从上一次listObjects读到的最后一个文件的下一个文件开始继续获取文件列表
$nextMarker = $listObjectInfo->getNextMarker();
$listObject = $listObjectInfo->getObjectList();
$listPrefix = $listObjectInfo->getPrefixList();
var_dump(count($listObject));
var_dump(count($listPrefix));
if ($nextMarker === '') {
break;
}
}
}
```

上述例子中提到的$options参数说明:

key	说明
delimiter	用于对Object名字进行分组的字符。所有名字包含指定的前缀且第一次出现delimiter字符之间的object作为一组元素——CommonPrefixes
prefix	限定返回的object key必须以prefix作为前缀。注意使用prefix查询时，返回的key中仍会包含prefix
max-keys	限定此次返回object的最大数，如果不设定，默认为100，max-keys取值不能大于1000
marker	设定结果从marker之后按字母排序的第一个开始返回

提示：

- prefix/delimiter/marker/max-keys都是可选参数。

判断文件是否存在

您可以使用OssClient::doesObjectExist判断文件是否存在：

```
<?php
/**
* 判断object是否存在
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket bucket名字
* @return null
*/
function doesObjectExist($ossClient, $bucket)
{
```

```
$object = "oss-php-sdk-test/upload-test-object-name.txt";
try{
$exist = $ossClient->doesObjectExist($bucket, $object);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
var_dump($exist);
}
```

创建虚拟目录

OSS是没有文件夹这个概念的，所有元素都是以Object来存储。创建文件夹本质上来说是创建了一个size为0的Object。对于这个Object可以上传下载，只是控制台会对以”/“结尾的Object以文件夹的方式展示。

```
<?php
/**
* 创建虚拟目录
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function createObjectDir($ossClient, $bucket) {
try{
$ossClient->createObjectDir($bucket, "dir");
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

删除单个文件

您可以使用OssClient::deleteObject删除单个文件：

```
<?php
/**
* 删除object
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket bucket名字
* @return null
*/
function deleteObject($ossClient, $bucket)
{
```

```
$object = "oss-php-sdk-test/upload-test-object-name.txt";
try{
$ossClient->deleteObject($bucket, $object);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

注意：

- 文件删除后将无法恢复。

删除多个文件

您可以使用OssClient::deleteObjects批量删除多个文件：

```
<?php
/**
* 批量删除object
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket bucket名字
* @return null
*/
function deleteObjects($ossClient, $bucket)
{
$objects = array();
$objects[] = "oss-php-sdk-test/upload-test-object-name.txt";
$objects[] = "oss-php-sdk-test/upload-test-object-name.txt.copy";
try{
$ossClient->deleteObjects($bucket, $objects);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

拷贝文件

您可以使用OssClient::copyObject拷贝文件：

```
<?php
/**
* 拷贝object
```

```
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket bucket名字
* @return null
*/
function copyObject($ossClient, $bucket)
{
$from_bucket = $bucket;
$from_object = "oss-php-sdk-test/upload-test-object-name.txt";
$to_bucket = $bucket;
$to_object = $from_object . '.copy';
try{
$ossClient->copyObject($from_bucket, $from_object, $to_bucket, $to_object);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

注意：

- OssClient::copyObject可以拷贝小于1GB的文件。拷贝一个大于1GB的文件时，请使用OssClient::uploadPartCopy。

修改文件元信息(Object Meta)

文件元信息(Object Meta)，是对用户上传到OSS的文件的属性描述，分为两种：HTTP标准属性（HTTP Headers）和User Meta（用户自定义元信息）。文件元信息可以在各种方式上传或者拷贝文件时进行设置。

HTTP标准属性，Cache-Control、Content-Disposition、Content-Encoding、Content-Language、Expires、Content-Length、Content-Type、Last-Modified等。

为了便于用户对Object进行更多描述。OSS中规定所有以x-oss-meta-为前缀的参数，则视为User Meta比如x-oss-meta-location。一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过8k。这些User Meta信息会在下载文件/获取文件元数据时返回。

更多文件元信息的介绍，请参看 文件元信息。

可以通过拷贝操作来实现修改已有文件元信息。如果拷贝操作的源文件地址和目标文件地址相同，都会直接替换源文件的文件元信息。

```
/**
* 修改文件元信息
* 利用copyObject接口的特性：当目的object和源object完全相同时，表示修改object的文件元信息
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
```

```
function modifyMetaForObject($ossClient, $bucket)
{
$fromBucket = $bucket;
$fromObject = "oss-php-sdk-test/upload-test-object-name.txt";
$toBucket = $bucket;
$toObject = $fromObject;
$copyOptions = array(
OssClient::OSS_HEADERS => array(
'Expires' => '2018-10-01 08:00:00',
'Content-Disposition' => 'attachment; filename="xxxxxx"',
'x-oss-meta-location' => 'location',
),
);
try{
$ossClient->copyObject($fromBucket, $fromObject, $toBucket, $toObject, $copyOptions);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

提示：

- 通过修改文件的Content-Type，可以修改文件的类型。
- 通过修改文件的Content-Disposition，可以控制文件的下载行为。

获取文件的文件元信息(Object Meta)

您可以使用OssClient::getObjectMeta获取文件的文件元信息：

```
<?php
/**
* 获取object meta, 也就是getObjectMeta接口
*
* @param OssClient $ossClient OssClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getObjectMeta($ossClient, $bucket)
{
$object = "oss-php-sdk-test/upload-test-object-name.txt";
try {
$objectMeta = $ossClient->getObjectMeta($bucket, $object);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
if (isset($objectMeta[strtolower('Content-Disposition')]) &&
```

```
'attachment; filename="xxxxxx"' === $objectMeta[strtolower('Content-Disposition')]
) {
print(__FUNCTION__ . ": ObjectMeta checked OK" . "\n");
} else {
print(__FUNCTION__ . ": ObjectMeta checked FAILED" . "\n");
}
}
```

设置文件访问权限

Object有四种访问权限：default（默认），private（私有读写），public-read（公共读私有写），public-read-write（公共读写），含义如下：

权限	描述	PHP SDK对应值
默认	Objec是遵循Bucket的读写权限，即Bucket是什么权限，Object就是什么权限，Object的默认权限	default
私有读写	Object是私有资源，即只有该Object的Owner拥有该Object的读写权限，其他的用户没有权限操作该Object	private
公共读私有写	Object是公共读资源，即非Object Owner只有Object的读权限，而Object Owner拥有该Object的读写权限	public-read
公共读写	Object是公共读写资源，即所有用户拥有对该Object的读写权限	public-read-write

Object的权限优先级高于Bucket。例如Bucket是private的，而Object ACL是公共读写，则访问这个Object时，先判断Object的ACL，所有用户都拥有这个Object的访问权限，即使这个Bucket是private。如果某个Object从来没设置过ACL，则访问权限遵循Bucket ACL。

您可以使用OssClient::putObjectAcl设置文件的访问权限：

```
<?php
/**
* 获取文件范围权限
*
* @param OssClient $ossClient OssClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function putObjectAcl($ossClient, $bucket)
{
$object = "oss-php-sdk-test/upload-test-object-name.txt";
$acl = "public-read";
try {
```

```
$ossClient->putObjectAcl($bucket, $object, $acl);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

获取文件访问权限

您可以使用OssClient::getObjectAcl获取文件的访问权限：

```
<?php
/**
* 获取object的访问权限
*
* @param OssClient $ossClient OssClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getObjectMeta($ossClient, $bucket)
{
$object = "oss-php-sdk-test/upload-test-object-name.txt";
try {
$objectAcl = $ossClient->getObjectAcl($bucket, $object);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
var_dump($objectAcl);
}
```

解冻归档存储类型文件

归档object的状态变换过程如下：

1. 归档类型的object初始时处于冷冻状态。
2. 提交解冻（restore）操作后，服务端执行解冻，object处于解冻中状态。
3. 解冻默认时间需要一分钟。
4. 完成解冻后，可以读取文件。
5. 解冻状态默认持续1天，最多延长7天，之后object又回到冷冻状态。

您可以调用OssClient::restoreObject方法进行归档object解冻：

```
/**
* 解冻归档类型object
```

```
*
* @param OssClient $ossClient OssClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function restoreObject($ossClient, $bucket)
{
$object = "oss-php-sdk-test/archive-test-object-name.txt";
try {
$ossClient->restoreObject($bucket, $object);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");

}
```

提示：

- 更详细解释请参看api接口Restore Object。

创建符号链接

符号链接是一种特殊的object，它指向具体的某个object,类似于windows上使用的快捷方式。

您可以调用OssClient::putSymlink方法创建符号链接：

```
<?php
/**
* 创建符号链接
*
* @param OssClient $ossClient OssClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function putSymlink($ossClient, $bucket)
{
$symlink = "test-symlink";
$object = "oss-php-sdk-test/symlink-test-object-name.txt";
try {
$ossClient->putSymlink($bucket, $symlink, $object);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");

}
```

提示：

- 更详细解释请参看api接口PutSymlink。

获取符号链接所指向的文件内容

用于获取符号链接，此操作要求用户对该符号链接有读权限。

您可以调用OssClient::getSymlink方法获取symlink所指向的的object内容：

```
<?php
/**
* 获取符号链接所指向的object内容
*
* @param OssClient $ossClient OssClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getSymlink($ossClient, $bucket)
{
$symlink = "test-symlink";
try {
$Symlinks = $ossClient->getSymlink($bucket, $symlink);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
var_dump($Symlinks);

}
```

提示：

- 更详细解释请参看api接口 GetSymlink。

授权访问

使用签名URL进行临时授权

生成签名URL

您可以将生成的签名URL提供给访客进行临时访问。生成签名URL时，您可以指定URL的过期时间，来限制访客的访问时长。授权访问的完整代码请参见：GitHub。

生成下载的签名URL

以下代码用于生成下载的签名URL：

```
<?php
/**
* 生成GetObject的签名URL,主要用于私有权限下的读访问控制。
*
* @param $ossClient OssClient OSSClient实例
* @param $bucket string 存储空间名称
* @return null
*/
function getSignedUrlForGettingObject($ossClient, $bucket)
{
$object = "test/test-signature-test-upload-and-download.txt";
$timeout = 3600; // URL的有效期是3600秒
try{
$ossClient->putObject($bucket, $object, "test-content");
$signedUrl = $ossClient->signUrl($bucket, $object, $timeout);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": signedUrl: " . $signedUrl. "\n");
/**
* 可以使用代码来访问签名的URL，也可以输入到浏览器中进行访问。
*/
//生成的URL默认以GET方式访问。
$request = new RequestCore($signedUrl);
$request->set_method('GET');
$request->send_request();
$res = new ResponseCore($request->get_response_header(), $request->get_response_body(), $request-
>get_response_code());
if ($res->isOK()) {
print(__FUNCTION__ . ": OK" . "\n");
} else {
print(__FUNCTION__ . ": FAILED" . "\n");
};
}
```

生成上传的签名URL

以下代码用于生成上传的签名URL：

```
<?php
/**
* 生成PutObject的签名URL,用于私有权限下的写访问控制。
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
* @throws OssException
*/
function getSignedUrlForPuttingObject($ossClient, $bucket)
{
$object = "test/test-signature-test-upload-and-download.txt";
$timeout = 3600;
$options = NULL;
try{
$signedUrl = $ossClient->signUrl($bucket, $object, $timeout, "PUT");
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": signedUrl: " . $signedUrl. "\n");
$content = file_get_contents(__FILE__);
$request = new RequestCore($signedUrl);
$request->set_method('PUT');
$request->add_header('Content-Type', '');
$request->add_header('Content-Length', strlen($content));
$request->set_body($content);
$request->send_request();
$res = new ResponseCore($request->get_response_header(),
$request->get_response_body(), $request->get_response_code());
if ($res->isOK()) {
print(__FUNCTION__ . ": OK" . "\n");
} else {
print(__FUNCTION__ . ": FAILED" . "\n");
};
}
```

临时凭证(STS)

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。STS更详细的解释请参见STS介绍。

使用STS凭证创建OssClient

拿到STS临时凭证后，您可以通过其中安全令牌(SecurityToken)以及临时访问密钥(AccessKeyId, AccessKeySecret)生成OssClient。

以下代码用于STS临时凭证创建一个OssClient：

```
<?php
```

```
use OSS\OssClient;
use OSS\Core\OssException;

$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
$securityToken = "<yourSecurityToken>";
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false, $securityToken);
```

静态网站托管

您可以将存储空间配置成静态网站托管模式。配置生效后，访问网站相当于访问存储空间，并且能够自动跳转至指定的索引页面和错误页面。

更多关于静态网站托管的介绍请参见 OSS静态网站托管。以下场景的完整代码请参见 GitHub。

设置静态网站托管

以下代码用于设置静态网站托管：

```
<?php
/**
* 设置静态网站托管
*
* @param $ossClient OssClient
* @param $bucket string 存储空间名称
* @return null
*/
function putBucketWebsite($ossClient, $bucket)
{
$websiteConfig = new WebsiteConfig("index.html", "error.html");
try {
$ossClient->putBucketWebsite($bucket, $websiteConfig);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

查看静态网站托管配置

以下代码用于查看静态网站托管配置：

```
<?php
/**
* 查看静态网站托管配置
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getBucketWebsite($ossClient, $bucket) {
$websiteConfig = null;
try{
$websiteConfig = $ossClient->getBucketWebsite($bucket);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
print($websiteConfig->serializeToXml() . "\n");
}
```

删除静态网站托管配置

以下代码用于删除静态网站托管配置：

```
<?php
/**
* 删除静态网站托管配置
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function deleteBucketWebsite($ossClient, $bucket) {
try{
$ossClient->deleteBucketWebsite($bucket);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

生命周期管理

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。每条规则包含：

- 规则ID。用于标识一条规则，同一存储空间内规则ID不能重复。
- 策略。有以下两种设置方式。同一存储空间内仅支持一种设置方式。
    - 按前缀匹配。此种方式允许创建多条规则，前缀不能重复。
    - 配置到整个存储空间。此种方式只能创建一条规则。
- 过期时间。有两种指定方式：
    - 指定距文件最后修改时间N天后过期。
    - 指定日期创建前的文件过期，之后的不过期。
- 是否生效。

更多关于生命周期的内容请参见 文件生命周期 。生命周期管理的完整代码请参见：GitHub

设置生命周期规则

以下代码用于设置生命周期规则：

```
<?php
/**
* 设置存储空间的生命周期规则
*
* @param OssClient $ossClient OssClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function putBucketLifecycle($ossClient, $bucket)
{
$lifecycleConfig = new LifecycleConfig();
$actions = array();
$actions[] = new LifecycleAction(OssClient::OSS_LIFECYCLE_EXPIRATION, OssClient::OSS_LIFECYCLE_TIMING_DAYS,
3);
$lifecycleRule = new LifecycleRule("delete obsoleted files", "obsoleted/", "Enabled", $actions);
$lifecycleConfig->addRule($lifecycleRule);
$actions = array();
$actions[] = new LifecycleAction(OssClient::OSS_LIFECYCLE_EXPIRATION, OssClient::OSS_LIFECYCLE_TIMING_DATE,
'2022-10-12T00:00:00.000Z');
$lifecycleRule = new LifecycleRule("delete temporary files", "temporary/", "Enabled", $actions);
$lifecycleConfig->addRule($lifecycleRule);
try {
$ossClient->putBucketLifecycle($bucket, $lifecycleConfig);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

查看生命周期规则

以下代码用于查看生命周期规则：

```
<?php
/**
* 查看存储空间的生命周期规则
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getBucketLifecycle($ossClient, $bucket)
{
$lifecycleConfig = null;
try{
$lifecycleConfig = $ossClient->getBucketLifecycle($bucket);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
print($lifecycleConfig->serializeToXml() . "\n");
}
```

清空生命周期规则

以下代码用于清空生命周期规则：

```
<?php
/**
* 清空存储空间的生命周期规则
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function deleteBucketLifecycle($ossClient, $bucket)
{
try{
$ossClient->deleteBucketLifecycle($bucket);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

设置访问日志

您可以开启存储空间的访问日志记录功能，开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。日志文件的格式为：

<TargetPrefix><SourceBucket>-YYYY-mm-DD-HH-MM-SS-UniqueString

更多关于访问日志的介绍，请参见开发指南中的**设置访问日志记录**。访问日志的完整代码请参见：GitHub。

开启访问日志记录

以下代码用于开启存储空间的访问日志记录：

```
<?php
/**
* 开启访问日志记录
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function putBucketLogging($ossClient, $bucket)
{
$option = array();
//设置存放日志文件的存储空间。
$targetBucket = $bucket;
$targetPrefix = "access.log";

try {
$ossClient->putBucketLogging($bucket, $targetBucket, $targetPrefix, $option);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

查看访问日志设置

以下代码用于查看存储空间的访问日志设置：

```
<?php
/**
```

```
* 查看访问日志设置
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getBucketLogging($ossClient, $bucket)
{
$loggingConfig = null;
$options = array();
try {
$loggingConfig = $ossClient->getBucketLogging($bucket, $options);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
print($loggingConfig->serializeToXml() . "\n");
}
```

关闭访问日志记录

以下代码用于关闭存储空间的访问日志记录：

```
<?php
/**
* 关闭访问日志记录
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function deleteBucketLogging($ossClient, $bucket)
{
try {
$ossClient->deleteBucketLogging($bucket);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

跨域资源共享

跨域资源共享（Cross-origin resource sharing，简称CORS）允许Web端的应用程序访问不属于本域的资源。OSS提供跨域资源共享接口，方便您控制跨域访问的权限。

更多关于跨域资源共享的介绍，请参见开发指南中的跨域访问。跨域资源共享的完整代码请参见：GitHub。

设置跨域资源共享（CORS）规则

以下代码用于设置指定存储空间的跨域资源共享规则，已存在的规则将被覆盖。

```
<?php
/**
* 设置存储空间的跨域资源共享规则
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function putBucketCors($ossClient, $bucket)
{
$corsConfig = new CorsConfig();
$rule = new CorsRule();
//AllowedHeaders和ExposeHeaders不支持通配符。
$rule->addAllowedHeader("x-oss-header");
//AllowedOrigins和AllowedMethods都能够最多支持一个”\*”通配符。”\*”表示对于所有的域来源或者操作都满足。
$rule->addAllowedOrigin("http://www.b.com");
$rule->addAllowedMethod("POST");
$rule->setMaxAgeSeconds(10);
//每个存储空间最多只能使用10条规则
$corsConfig->addRule($rule);

try{
$ossClient->putBucketCors($bucket, $corsConfig);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

获取跨域资源共享规则

以下代码用于获取跨域资源共享规则：

```
<?php
/**
* 获取跨域资源共享规则
*
```

```
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getBucketCors($ossClient, $bucket)
{
$corsConfig = null;
try{
$corsConfig = $ossClient->getBucketCors($bucket);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
print($corsConfig->serializeToXml() . "\n");
}
```

删除跨域资源共享规则

以下代码用于删除指定存储空间的所有跨域资源共享规则：

```
<?php
/**
* 删除存储空间的跨域资源共享规则
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function deleteBucketCors($ossClient, $bucket)
{
try{
$ossClient->deleteBucketCors($bucket);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

防盗链

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。

- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

更多关于防盗链的介绍，请参见开发指南中的**设置防盗链** 。防盗链的完整代码请参见：GitHub。

设置防盗链

以下代码用于设置防盗链：

```
<?php
/**
* 设置防盗链
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function putBucketReferer($ossClient, $bucket)
{
$refererConfig = new RefererConfig();
//设置允许空Referer。
$refererConfig->setAllowEmptyReferer(true);
//添加Referer白名单。Referer参数支持通配符“*”和“？”。
$refererConfig->addReferer("www.aliiyun.com");
$refererConfig->addReferer("www.aliiyuncs.com");
try{
$ossClient->putBucketReferer($bucket, $refererConfig);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

获取防盗链信息

以下代码用于获取防盗链信息：

```
<?php
/**
* 获取防盗链信息
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getBucketReferer($ossClient, $bucket)
{
$refererConfig = null;
```

```
try{
$refererConfig = $ossClient->getBucketReferer($bucket);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
print($refererConfig->serializeToXml() . "\n");
}
```

清空防盗链

以下代码用于清空防盗链：

```
<?php
/**
* 清空防盗链
* 防盗链不能直接清空，需要新建一个允许空Referer的规则来覆盖之前的规则。
*
* @param OssClient $ossClient OSSClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function deleteBucketReferer($ossClient, $bucket)
{
$refererConfig = new RefererConfig();
try{
$ossClient->putBucketReferer($bucket, $refererConfig);
} catch(OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

自定义域名绑定

您可以通过添加CNAME记录将自定义域名绑定到OSS的存储空间上，从而将存储无缝迁移到OSS上。

例如您的域名是my-domain.com，之前使用的所有图片资源都是形如http://img.my-domain.com/x.jpg的格式，在将图片存储迁移到OSS之后，通过绑定自定义域名，仍可以使用原来的地址访问图片。绑定自定义域名的步骤如下：

1. 开通OSS服务，并创建存储空间。

2. 修改域名的DNS配置，增加一个CNAME记录，将img.my-domain.com指向OSS服务的Endpoint。
3. 通过OSS控制台或使用SDK将img.my-domain.com与创建的存储空间绑定。
4. 将图片上传到存储空间中。

详情请参见开发指南中的绑定自定义域名。

添加CNAME记录

以下代码用于为存储空间添加CNAME记录：

```
<?php
/**
* 为存储空间添加CNAME记录。
*
* @param OssClient $ossClient OssClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function addBucketCname($ossClient, $bucket)
{
//填写您的域名，例如img.my-domain.com。
$myDomain = 'img.my-domain.com';
try {
$ossClient->addBucketCname($bucket, 'img.my-domain.com');
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

获取CNAME记录

以下代码用于获取存储空间已添加的CNAME记录：

```
<?php
/**
* 获取存储空间已添加的CNAME记录。
*
* @param OssClient $ossClient OssClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function getBucketCname($ossClient, $bucket)
{
//填写您的域名，例如img.my-domain.com。
$myDomain = 'img.my-domain.com';
```

```
try {
$cnameConfig = $ossClient->getBucketCname($bucket);
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
var_dump($cnameConfig);
print(__FUNCTION__ . ": OK" . "\n");
}
```

删除CNAME记录

以下代码用于删除存储空间的CNAME记录：

```
<?php

/**
* 删除存储空间的CNAME记录。
*
* @param OssClient $ossClient OssClient实例
* @param string $bucket 存储空间名称
* @return null
*/
function deleteBucketCname($ossClient, $bucket)
{
//填写您的域名，例如img.my-domain.com。
$myDomain = 'img.my-domain.com';
try {
$ossClient->deleteBucketCname($bucket, 'img.my-domain.com');
} catch (OssException $e) {
printf(__FUNCTION__ . ": FAILED\n");
printf($e->getMessage() . "\n");
return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

图片处理

OSS图片处理，是OSS对外提供的海量、安全、低成本、高可靠的图片图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

图片处理的详细信息请参见OSS图片处理指南。

图片处理基础

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放
- 图片裁剪
- 图片旋转
- 图片效果
- 图片水印，包括添加图片、文字、图文混合水印
- 自定义图片处理样式
- 级联处理，调用多个图片处理功能

图片处理

图片处理使用标准的HTTP GET请求。您可以在URL的QueryString中设置处理参数。

如果图片文件的访问权限为私有读写，必须通过授权才能进行访问。

匿名访问

您可以通过如下格式的三级域名匿名访问处理后的图片：

```
http://<yourBucketName>.<yourEndpoint>/<yourObjectName>?x-oss-process=image/<yourAction>,<yourParamValue>
```

参数	描述
bucket	存储空间名称
endpoint	存储空间所在地域的访问域名
object	图片文件名称
image	图片处理的保留标志符
action	对图片做的操作，如缩放、裁剪、旋转等
param	对图片做的操作所对应的参数

基础操作

例如，将图缩略成宽度为100，高度按比例处理：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

自定义样式

使用如下格式的三级域名匿名访问图片处理：

```
http://<yourBucketName>.<yourEndpoint>/<yourObjectName>?x-oss-process=style/<yourStyleName>
```

- style：自定义样式的保留标志符。
- yourStyleName：自定义样式名称，即通过控制台自定义样式的规则名称。

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=style/oss-pic-style-w-100
```

级联处理

通过级联处理，可以对一张图片顺序进行多个操作，格式如下：

```
http://<yourBucketName>.<yourEndpoint>/<yourObjectName>?x-oss-
process=image/<yourAction1>,<yourParamValue1>/<yourAction2>,<yourParamValue2>/...
```

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100/rotate,90
```

支持HTTPS访问

图片服务支持HTTPS访问，例如：

```
https://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

授权访问

以下代码用于生成带签名的图片处理URL：

```
<?php
require_once __DIR__ . '/vendor/autoload.php';

use OSS\OssClient;

//您选定的OSS所在地域的访问域名。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
$bucket = "<yourBucketName>";
$object = "<yourObjectName>";
```

```
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

/**
* 生成一个带签名的URL, URL的有效期是3600秒,可以直接使用浏览器访问。
*/
//授权访问支持 自定义样式、 HTTPS、 级联处理。
//timeout时间单位是秒。
$timeout = 3600;
$options = array(
OssClient::OSS_PROCESS => "image/resize,m_lfit,h_100,w_100" );
$signedUrl = $ossClient->signUrl($bucket, $object, $timeout, "GET", $options);

Common::println("rtmp url: \n" . $signedUrl);
```

SDK访问

对于任意权限的图片文件，都可以直接使用SDK访问和处理。

SDK处理图片文件支持自定义样式、HTTPS和级联处理。图片处理的完整代码请参见：GitHub。

基础操作

以下代码展示了图片处理的基础操作：

```
<?php
require_once __DIR__ . '/vendor/autoload.php';
use OSS\OssClient;

//您选定的OSS所在地域的访问域名。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
$bucket = "<yourBucketName>";
$object = "<yourObjectName>";
$download_file = "download.jpg";

$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

// 先把本地的example.jpg上传到指定$bucket, 命名为$object。
$ossClient->uploadFile($bucket, $object, "example.jpg");

// 图片缩放
$options = array(
OssClient::OSS_FILE_DOWNLOAD => $download_file,
OssClient::OSS_PROCESS => "image/resize,m_fixed,h_100,w_100" );
$ossClient->getObject($bucket, $object, $options);

// 图片裁剪
$options = array(
OssClient::OSS_FILE_DOWNLOAD => $download_file,
OssClient::OSS_PROCESS => "image/crop,w_100,h_100,x_100,y_100,r_1" );
$ossClient->getObject($bucket, $object, $options);
```

```
// 图片旋转
$options = array(
OssClient::OSS_FILE_DOWNLOAD => $download_file,
OssClient::OSS_PROCESS => "image/rotate,90" );
$ossClient->getObject($bucket, $object, $options);

// 图片锐化
$options = array(
OssClient::OSS_FILE_DOWNLOAD => $download_file,
OssClient::OSS_PROCESS => "image/sharpen,100" );
$ossClient->getObject($bucket, $object, $options);

// 图片水印
$options = array(
OssClient::OSS_FILE_DOWNLOAD => $download_file,
OssClient::OSS_PROCESS => "image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ" );
$ossClient->getObject($bucket, $object, $options);

// 图片格式转换
$options = array(
OssClient::OSS_FILE_DOWNLOAD => $download_file,
OssClient::OSS_PROCESS => "image/format,png" );
$ossClient->getObject($bucket, $object, $options);

// 获取图片信息。
$options = array(
OssClient::OSS_FILE_DOWNLOAD => $download_file,
OssClient::OSS_PROCESS => "image/info" );
$ossClient->getObject($bucket, $object, $options);

//最后删除上传的$object。
$ossClient->deleteObject($bucket, $object);
```

自定义样式

以下代码用于自定义图片样式：

```
<?php
require_once __DIR__ . '/vendor/autoload.php';
use OSS\OssClient;

//您选定的OSS所在地域的访问域名。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
$bucket = "<yourBucketName>";
$object = "<yourObjectName>";
$download_file = "download.jpg";

$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);

// 先把本地的example.jpg上传到指定$bucket, 命名为$object。
$ossClient->uploadFile($bucket, $object, "example.jpg");
```

```
$style = "style/oss-pic-style-w-300";
$options = array(
OssClient::OSS_FILE_DOWNLOAD => $download_file,
OssClient::OSS_PROCESS => $style);
$ossClient->getObject($bucket, $object, $options);

//最后删除上传的$object。
$ossClient->deleteObject($bucket, $object);
```

级联处理

以下代码用于级联处理图片：

```
<?php
require_once __DIR__ . '/vendor/autoload.php';
use OSS\OssClient;

//您选定的OSS数据中心访问域名。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
$bucket = "<yourBucketName>";
$object = "<yourObjectName>";
$download_file = "download.jpg";

$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);

// 先把本地的example.jpg上传到指定$bucket， 命名为$object。
$ossClient->uploadFile($bucket, $object, "example.jpg");

$style = "image/resize,m_fixed,w_100,h_100/rotate,90";
$options = array(
OssClient::OSS_FILE_DOWNLOAD => $download_file,
OssClient::OSS_PROCESS => $style);
$ossClient->getObject($bucket, $object, $options);

//最后删除上传的$object。
$ossClient->deleteObject($bucket, $object);
```

图片处理工具

您可以通过可视化图片处理工具ImageStyleViever直观地看到OSS图片处理结果。

异常处理

OSS PHP SDK异常（OssException）包括参数无效、文件不存在等错误。您可以通过getMessage方法获取错误信息。

OssException的详细信息请参见GitHub 。

异常处理示例

以下代码展示了创建一个已存在的存储空间时的异常处理，并打印出错误信息（Message）。

```
    try {
$ossClient->createBucket($bucket);
} catch (OssException $e) {
print("Exception:" . $e->getMessage() . "\n");
}
```

您还可以获取以下信息：

参数	说明
HTTPStatus	HTTP状态码，通过方法getHTTPStatus获取。
ErrorCode	OSS返回的错误码，通过方法getErrorCode获取。
ErrorMessage	OSS返回的错误信息，通过方法getErrorMessage获取。
RequestId	用于唯一标识该请求的UUID。当您无法解决问题时，可以提供RequestId来请求OSS开发工程师的帮助。通过方法getRequestId获取。
Details	OSS返回的错误信息描述。通过方法getDetails获取。

OSS常见错误码

错误码	描述	HTTP状态码
AccessDenied	拒绝访问	403
BucketAlreadyExists	存储空间已经存在	409
BucketNotEmpty	存储空间不为空	409
EntityTooLarge	实体过大	400
EntityTooSmall	实体过小	400
FileGroupTooLarge	文件组过大	400
FilePartNotExist	文件分片不存在	400
FilePartStale	文件分片过时	400
InvalidArgument	参数格式错误	400

InvalidAccessKeyId	AccessKeyId不存在	403
InvalidBucketName	无效的存储空间名称	400
InvalidDigest	无效的摘要	400
InvalidObjectName	无效的文件名称	400
InvalidPart	无效的分片	400
InvalidPartOrder	无效的分片顺序	400
InvalidTargetBucketForLogging	Logging操作中有无效的目标bucket	400
InternalError	OSS内部错误	500
MalformedXML	XML格式非法	400
MethodNotAllowed	不支持的方法	405
MissingArgument	缺少参数	411
MissingContentLength	缺少内容长度	411
NoSuchBucket	存储空间不存在	404
NoSuchKey	文件不存在	404
NoSuchUpload	分片上传ID不存在	404
NotImplemented	无法处理的方法	501
PreconditionFailed	预处理错误	412
RequestTimeTooSkewed	客户端本地时间和OSS服务器时间相差超过15分钟	403
RequestTimeout	请求超时	400
SignatureDoesNotMatch	签名错误	403
InvalidEncryptionAlgorithmError	指定的熵编码加密算法错误	400

C-SDK

前言

SDK下载

版本3.5.0 （2017-08-01）：

- Linux SDK包：aliyun-oss-c-sdk-3.5.0.tar.gz

  Windows SDK包：aliyun-oss-c-sdk-3.5.0.zip

  源代码：GitHub

版本迭代详情参考这里

兼容性

3.*.* 相对于 3.0.0兼容

3.0.0 相对于 2.*.* 的兼容性

- Windows兼容
- Linux接口兼容，链表(aos_list_t)遍历接口不兼容
  - aos_list_for_each_entry
  - aos_list_for_each_entry_reverse
  - aos_list_for_each_entry_safe
  - aos_list_for_each_entry_safe_reverse

2.1.0 相对于 2.0.0兼容

2.0.0 相对于 1.0.0，以下结构体和接口不兼容，其余都兼容

- oss_request_options_t
- oss_get_object_to_buffer
- oss_get_object_to_file
- oss_get_object_to_buffer_by_url
- oss_get_object_to_file_by_url
- oss_init_multipart_upload
- oss_complete_multipart_upload

1.0.0 相对于 0.0.* 不兼容

简介

本文档主要介绍OSS C SDK的安装和使用。本文档假设您已经开通了阿里云OSS 服务，并创建了AccessKeyId

和AccessKeySecret。如果您还没有开通或者还不了解OSS，请登录OSS产品主页获取更多的帮助。

安装

SDK 安装

要求

- 开通阿里云OSS服务，并创建了AccessKeyId 和AccessKeySecret。
- 如果您还没有开通或者还不了解阿里云OSS服务，请登录OSS产品主页了解。
- 如果还没有创建AccessKeyId和AccessKeySecret，请到阿里云Access Key管理创建Access Key。

Linux

环境依赖

OSS C SDK使用curl进行网络操作，无论是作为客户端还是服务器端，都需要依赖curl。另外，OSS C SDK使用apr、apr-util库解决内存管理以及跨平台问题，使用minixml库解析请求返回的xml，OSS C SDK(Linux)并没有带上这几个外部库，您需要确认这些库已经安装，并且将它们的头文件目录和库文件目录都加入到了项目中。

安装第三方库

Ubuntu/Debian

安装CMake

执行以下命令安装CMake。

```
sudo apt-get install cmake
```

安装第三方库

执行以下命令安装第三方库。

```
sudo apt-get install libcurl4-openssl-dev libapr1-dev libaprutil1-dev libmxml-dev
```

RedHat/Aliyun/CentOS

安装CMake

执行以下命令安装CMake。

```
sudo yum install cmake
```

安装第三方库

执行以下命令安装第三方库。

```
sudo yum install curl-devel apr-devel apr-util-devel
```

minixml

请下载系统对应的rpm包:

- 64位: mxml-2.9-1.x86_64.rpm
- 32位: mxml-2.9-1.i386.rpm

执行以下命令安装minixml:

```
rpm -ivh mxml-2.9-1.x86_64.rpm
```

SuSE

安装CMake

执行以下命令安装CMake。

```
zypper install cmake
```

安装第三方库

执行以下命令安装第三方库。

```
zypper install libcurl-devel libapr1-devel libapr-util1-devel mxml-devel
```

其它Linux

CMake (建议2.6.0及以上版本)

请从这里下载，典型的安装方式如下：

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

注意：

- 执行./configure时默认是配置安装目录为/usr/local/，如果需要指定安装目录，请使用./configure —prefix=/your/install/path/

libcurl （建议 7.32.0 及以上版本）

请从这里下载，并参考libcurl 安装指南安装。典型的安装方式如下：

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

注意：

- 执行./configure时默认是配置安装目录为/usr/local/，如果需要指定安装目录，请使用./configure —prefix=/your/install/path/
- 如果要使用MEDIA-C-SDK，请确保./configure执行完后，最后一行的Protocols里面包含HTTPS，如果没有，请先安装openssl-devel等ssl开发包，然后重新安装libcurl。

apr （建议 1.5.2 及以上版本）

请从这里下载，典型的安装方式如下：

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

注意：

- 执行./configure时默认是配置安装目录为/usr/local/，如果需要指定安装目录，请使用 ./configure —prefix=/your/install/path/

apr-util（建议 1.5.4 及以上版本）

请从这里下载，安装时需要注意指定—with-apr选项，典型的安装方式如下：

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

注意：

- 执行./configure时默认是配置安装目录为/usr/local/，如果需要指定安装目录，请使用 ./configure —prefix=/your/install/path/
- 需要通过—with-apr指定apr安装目录，如果apr安装到系统目录下需要指定—with-apr=/usr/local/apr/

minixml（推荐使用 2.9 版本）

请从这里下载，典型的安装方式如下:

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

注意：

- minixml推荐使用 2.9 版本；
- 执行./configure时默认是配置安装目录为/usr/local/，如果需要指定安装目录，请使用 ./configure —prefix=/your/install/path/ 。

编译安装OSS C SDK

典型的编译命令如下：

```
cmake .
make
make install
```

注意：

- 执行cmake . 时默认会到/usr/local/下面去寻找curl，apr，apr-util，mxml的头文件和库文件。
- 默认编译是Debug类型，可以指定以下几种编译类型： Debug, Release, RelWithDebInfo和MinSizeRel，如果要使用release类型编译，则执行cmake -f CMakeLists.txt -DCMAKE_BUILD_TYPE=Release
- 如果您在安装curl，apr，apr-util，mxml时指定了安装目录，则需要在执行cmake时指定这些库的路径，
    - 比如：cmake -f CMakeLists.txt -DCURL_INCLUDE_DIR=/usr/local/include/curl/ -DCURL_LIBRARY=/usr/local/lib/libcurl.so -DAPR_INCLUDE_DIR=/usr/local/include/apr-1/ -DAPR_LIBRARY=/usr/local/lib/libapr-1.so -DAPR_UTIL_INCLUDE_DIR=/usr/local/apr/include/apr-1 -DAPR_UTIL_LIBRARY=/usr/local/apr/lib/libaprutil-1.so -DMINIXML_INCLUDE_DIR=/usr/local/include -DMINIXML_LIBRARY=/usr/local/lib/libmxml.so
- 如果要指定安装目录，则需要在cmake时增加： -DCMAKE_INSTALL_PREFIX=/your/install/path/usr/local/
- 如果执行cmake时报以下错误：Could not find apr-config/apr-1-config，原因是在默认路径里面找不到apr-1-config文件，这时候可以在执行cmake命令时，在最后面加上-DAPR_CONFIG_BIN=/path/to/bin/apr-1-config。如果报：Could not find apu-config/apu-1-config，则需要加上-DAPU_CONFIG_BIN=/path/to/bin/apu-1-config。

Linux示例工程

基于Aliyun OSS C SDK的示例工程 aliyun-oss-c-sdk-demo.tar.gz 。其中，示例oss-c-sdk-demo-specified-installation基于OSS C SDK及依赖的第三方库，都安装在/home/your/oss/csdk下；其它示例工程是基于OSS C SDK及依赖的第三方库，安装在默认目录下，即安装时不指定目录。

- oss-c-sdk-demo-centos Aliyun/CentOS下，基于默认安装路径的OSS C SDK示例工程
- oss-c-sdk-demo-debian Debian下，基于默认安装路径的OSS C SDK示例工程
- oss-c-sdk-demo-redhat RedHat下，基于默认安装路径的OSS C SDK示例工程
- oss-c-sdk-demo-suse SuSE下，基于默认安装路径的OSS C SDK示例工程
- oss-c-sdk-demo-ubuntu Ubuntu下，基于默认安装路径的OSS C SDK示例工程
- oss-c-sdk-demo-specified-installation Linux下，基于自定义安装目录的OSS C SDK示例工程

示例工程的详细说明，请参看 Linux下使用Aliyun OSS C SDK 。

解压后进入工程目录(oss-c-sdk-demo-xxx),执行make，编译示例工程；执行make clean清理编译产生的文件；执行./main运行可执行程序。

提示：

- 示例代码中的OSS_ENDPOINT、ACCESS_KEY_ID、ACCESS_KEY_SECRET、

BUCKET_NAME请更换成有效值；
- 如果OSS C SDK及依赖库的动态库不在系统目录下，执行时请使用LD_LIBRARY_PATH指定。

Windows

Windows环境下，OSS C SDK目前只支持 **Win32** 方式编译，暂不支持 **x64**。使用 **Win32** 编译的库，可以在 **Win32** 或 **x64** 工程中使用。

环境与依赖

- OSS C SDK(Windows)依赖的第三方库和Linux版本一致，分别是apr，apr-util，curl，mxml。
- SDK提供了Visual Studio 2008和Visual Studio 2010的项目工程，分别支持Visual Studio 2008，Visual Studio 2010及其以后版本。
- SDK提供了third_party，包含了需要用到的第三方库(apr, apr-util, curl, mxml)的头文件和库文件，用户可以使用这里的头文件和库文件编译，运行sample和test项目。

Visual Studio 2008版本

- 对应的项目文件为oss_c_sdk_2008.sln等包含2008的文件
- OSS C SDK需要用到stdint.h头文件，Visual C++ 2008默认是没有此头文件，如果用户没有提前安装此头文件，可以将third_party/include中的stdint.h.bak重命名为stdint.h后使用。
- 从2.0.0版本才开始支持Visual Studio 2008，之前版本均不支持

Visual Studio 2010及其以后版本

- 对应的项目文件为oss_c_sdk.sln等不包含数字的文件。
- Visual Studio 2010及其以后版本都包含了stdint.h头文件。
- 如果用户使用Visual Studio 2012及其以后版本打开时，会提示用户是否将项目升级成使用最新版的编译器和库，这里最好和用户自己的项目保持一致：如果用户的项目使用了最新版本的编译器和库，就选择升级，否则可以不升级。

Visual Studio示例工程

示例工程提供了基于Visual Studio 2008/2010/2012/2013/2015的工程，您可以用Visual Studio直接打开对应工程， 修改Endpoint/AccessKeyID/AccessKeySecret/BucketName等配置后，既可编译运行。您也可以在示例工程基础上开发您的项目。

- 示例工程：aliyun-oss-c-sdk-sample.zip
- GitHub地址：GitHub

注意：

- 运行oss-c-sdk-sample前，请在Visual Studio做如下配置。在Sulution Explore中选择工程oss-c-sdk-sample，右击选择Property，在oss-c-sdk-sample Property Pages中配置Configuration Properties -> Debugging -> Environment为 **PATH=..\oss-c-sdk\lib\Release\;%PATH%** 。

使用Visual Studio编译OSS C SDK、运行示例程序的详细步骤及常见问题，请参考 Windows下编译使用Aliyun OSS C SDK。

示例工程

C SDK提供丰富的示例程序，方便用户参考或直接使用。您可以从GitHub获取示例程序。示例程序包括以下内容：

示例文件	示例内容
oss_put_object_sample	展示了文件上传的用法，包括从内存/文件上传、携带MD5校验上传、携带元数据上传、url签名上传等
oss_get_object_sample.c	展示了文件下载的用法，包括下载到内存、下载到文件、范围下载、URL签名下载等
oss_append_object_sample.c	展示了追加上传的用法，包括从内存追加上传、从文件追加上传
oss_multipart_upload_sample.c	展示了分片上传的用法，包括从内存分片上传、从文件分片上传、取消上传分片上传等
oss_resumable_sample.c	展示了并发断点续传上传/下载的用法
oss_head_object_sample.c	展示了获取文件元信息的用法
oss_list_object_sample.c	展示了列举文件的用法
oss_delete_object_sample.c	展示了删除文件的用法，包括删除单个文件、删除多个文件
oss_callback_sample.c	展示了上传回调的用法，包括上传回调、分片上传回调
oss_progress_sample.c	展示了进度条的用法，包括上传、追加上传、分片上传、下载等
oss_crc_sample.c	展示了上传、下载使用CRC校验的用法
oss_image_sample.c	展示了图片处理的用法

初始化

确定Endpoint

Endpoint是阿里云OSS服务在各个区域的地址，目前支持两种形式

Endpoint类型	解释
OSS区域地址	使用OSS Bucket所在区域地址，各个区域Endpoint参考这里
用户自定义域名	用户自定义域名，且CNAME指向OSS域名

关于Endpoint，可以参考：点击查看。

OSS区域地址

使用OSS Bucket所在区域地址，Endpoint查询可以有下面两种方式：

- 查询Endpoint与区域对应关系详情，可以参考：点击查看。
- 您可以登录 阿里云OSS控制台，进入Bucket概览页，Bucket域名的后缀部分：如bucket-1.oss-cn-hangzhou.aliyuncs.com的oss-cn-hangzhou.aliyuncs.com部分为该Bucket的外网Endpoint。

CNAME

- 您可以将自己拥有的域名通过CNAME绑定到某个存储空间（Bucket）上，然后通过自己域名访问存储空间内的文件
- 比如您要将域名new-image.xxxxx.com绑定到深圳区域的名称为image的存储空间上：
- 您需要到您的域名xxxxx.com托管商那里设定一个新的域名解析，将http://new-image.xxxxx.com解析到 http://image.oss-cn-shenzhen.aliyuncs.com ，类型为CNAME

配置密钥

要接入阿里云OSS，您需要拥有一对有效的 AccessKey(包括AccessKeyId和AccessKeySecret)用来进行签名认证。可以通过如下步骤获得：

- 注册阿里云账号
- 申请AccessKey

在获取到 AccessKeyId和 AccessKeySecret之后，您可以按照下面步骤进行初始化

初始化请求选项

使用OSS C SDK时需要初始化请求选项(oss_request_options_t)，其中config用于存储访问OSS的基本信息，比如OSS域名、用户的AccessKeyId 、用户的AccessKeySecret。is_cname变量指定访问OSS是否使用

CNAME。 ctl用于设置访问OSS时的控制信息。

```
void init_options(oss_request_options_t *options) {
options->config = oss_config_create(options->pool);
aos_str_set(&options->config->endpoint, "<您的Endpoint>");
aos_str_set(&options->config->access_key_id, "<您的AccessKeyId>");
aos_str_set(&options->config->access_key_secret, "<您的AccessKeySecret>");
options->config->is_cname = 0;
options->ctl = aos_http_controller_create(options->pool, 0);
}

int main() {
aos_pool_t *p;
oss_request_options_t *options;

/* 全局变量初始化，应该放在程序启动时，其他所有逻辑之前 */
if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
return -1;
}

/* 初始化内存池和options */
aos_pool_create(&p, NULL);
options = oss_request_options_create(p);
init_options(options);

/* 逻辑代码 */

/* 释放内存pool资源，包括了通过pool分配的内存，比如options等*/
aos_pool_destroy(p);

/* 释放全局资源，应该放在程序结束前 */
aos_http_io_deinitialize();

return 0;
}
```

注：

- 如果想使用https，只需要设置endpoint时，前缀使用https://即可。
- 支持多线程，但是aos_http_io_initialize和aos_http_io_deinitialize只需要在主线程里面调用即可，其他线程不需要调用。
- 如果想设置oss c sdk底层libcurl通信时的一些参数，可以对请求选项中的ctl进行设置，实现控制诸如数据上传最低速度、连接超时时间、DNS缓存失效时间等目的。
- 您也可以通过ctl获得使用oss c sdk访问OSS的性能参数。以上传数据为例，用户在上传完数据后可以得到一系列的参数指标，比如开始上传数据的时间start_time，第一字节上传的时间first_byte_time，完成数据上传的时间finish_time。
- 关于如何设置请求选项，请详见oss c sdk如何设置通信时和CURL相关的一些参数

快速入门

在这一章里，您将学到如何用OSS C SDK完成一些基本的操作。

Step-1.初始化OSS C SDK运行环境

OSS C SDK使用时首先需要初始化运行环境，使用结束前需要清理运行环境，下面代码演示初始化OSS C SDK运行环境：

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

/* 调用OSS 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aos_http_io_deinitialize方法释放之前分配的全局资源 */
aos_http_io_deinitialize();
return 0;
}
```

注：

- aos_http_io_initialize初始化OSS C SDK运行环境，第一个参数可以用于个性化设置user agent的内容，用作后续统计。
- aos_http_io_deinitialize清理OSS C SDK运行环境。

Step-2.初始化请求选项

OSS C SDK的所有操作需要初始化请求选项，下面代码完成初始化请求选项:

```
    /* 等价于apr_pool_t，用于内存管理的内存池，实现代码在apr库中 */
aos_pool_t *pool;
oss_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 */
aos_pool_create(&pool, NULL);

/* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,acces_key_secret，is_cname, curl参数等全局配置信息
* options的内存是由pool分配的，后续释放掉pool后，options的内存也相当于释放掉了，不再需要单独释放内存
```

```
*/
options = oss_request_options_create(pool);
options->config = oss_config_create(options->pool);

/* aos_str_set是用char*类型的字符串初始化aos_string_t类型*/
aos_str_set(&options->config->endpoint, "<您的Endpoint>");
aos_str_set(&options->config->access_key_id, "<您的AccessKeyId>");
aos_str_set(&options->config->access_key_secret, "<您的AccessKeySecret>");

/* 是否使用了CNAME */
options->config->is_cname = 0;

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

Step-3. 新建存储空间(Bucket)

您可以按照下面的代码新建一个存储空间：

```
    aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *resp_headers;
oss_acl_e oss_acl = OSS_ACL_PRIVATE;
char *bucket_name = "<您的bucket名字>";
aos_string_t bucket;

aos_pool_create(&p, NULL);

options = oss_request_options_create(p);
init_options(options);

/* 将char*类型数据赋值给aos_string_t类型的bucket */
aos_str_set(&bucket, bucket_name);
s = oss_create_bucket(options, &bucket, oss_acl, &resp_headers);

/* 判断请求是否成功 */
if (aos_status_is_ok(s)) {
printf("create bucket succeeded\n");
} else {
printf("create bucket failed\n");
}

/* 执行完一个请求后，释放掉这个内存池，结果就是会释放掉这个请求过程中各个部分分配的内存 */
aos_pool_destroy(p);
```

注：

- Bucket的命名规范请查看OSS 基本概念
- Bucket名字不能与OSS服务中其他用户已有的存储空间重复，所以你需要选择一个独特的存储空间名字以避免创建失败
- oss_create_bucket的返回值是aos_status_t*类型，包括了code(http

code)，error_code，error_msg和req_id，req_id可以协助调查问题。其他接口，如果没有特殊说明外，也都是返回aos_status_t*类型。

Step-4. 上传文件

文件是OSS中最基本的数据单元，用下面代码可以实现一个文件的上传：

```
    aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的object名字>";
aos_string_t bucket;
aos_string_t object;
char *data = "object content";
aos_list_t buffer;
aos_buf_t *content;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&object, object_name);
aos_str_set(&bucket, bucket_name);
headers = aos_table_make(p, 0);

/* 将char*类型的数据转换为oss_put_object_from_buffer接口需要的aos_list_t类型的 */
aos_list_init(&buffer);
content = aos_buf_pack(options->pool, data, strlen(data));
aos_list_add_tail(&content->node, &buffer);

/* 上传文件 */
s = oss_put_object_from_buffer(options, &bucket, &object, &buffer, headers, &resp_headers);

/* 判断请求是否成功 */
if (aos_status_is_ok(s)) {
printf("put file succeeded\n");
} else {
printf("put file failed\n");
}

/* 释放资源 */
aos_pool_destroy(p);
```

注：

- 关于上传文件更详细的信息，参见上传文件。

Step-5. 列出存储空间中的所有文件

当您完成一系列上传后，可能需要查看某个存储空间中有哪些文件，可以通过下面的程序实现：

```
    aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
aos_string_t bucket;
oss_list_object_params_t *params;
oss_list_object_content_t *content;
int max_ret = 1000;
char *key;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&bucket, bucket_name);
params = oss_create_list_object_params(p);
params->max_ret = max_ret;
aos_str_set(&params->prefix, "<prefix>");
aos_str_set(&params->delimiter, "<delimiter>");
aos_str_set(&params->marker, "<marker>");

s = oss_list_object(options, &bucket, params, &resp_headers);

/* 判断请求是否成功 */
if (aos_status_is_ok(s)) {
printf("list file succeeded\n");
} else {
printf("list file failed\n");
}

/* 获取每个文件的名称 */
aos_list_for_each_entry(content, &params->object_list, node) {
key = apr_psprintf(p, "%.*s", content->key.len, content->key.data);
}

/* 释放资源 */
aos_pool_destroy(p);
```

注：

- 更多灵活的参数配置，可以参考管理文件

Step-6. 下载指定文件

您可以参考下面的代码简单地实现下载指定文件：

```
    aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的object名字>";
aos_string_t bucket;
aos_string_t object;
aos_list_t buffer;
aos_buf_t *content;
char *buf;
int64_t len = 0;
int64_t size = 0;
int64_t pos = 0;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&object, object_name);
aos_str_set(&bucket, bucket_name);
headers = aos_table_make(p, 1);

/* 下载文件到buffer中 */
aos_list_init(&buffer);
s = oss_get_object_to_buffer(options, &bucket, &object, headers, &buffer, &resp_headers);

/* 判断请求是否成功 */
if (aos_status_is_ok(s)) {
printf("get file succeeded\n");
} else {
printf("get file failed\n");
}

/* 从buffer中将aos_list_t类型的数据转为char*类型的，并计算读到的文件总长度 */
len = aos_buf_list_len(&buffer);
buf = aos_pcalloc(p, len + 1);
buf[len] = '\0';
aos_list_for_each_entry(content, &buffer, node) {
size = aos_buf_size(content);
memcpy(buf + pos, content->pos, size);
pos += size;
}

/* 释放资源 */
aos_pool_destroy(p);
```

注：

- 关于下载文件更详细的信息，参见下载文件

管理Bucket

Bucket是OSS上的存储空间，也是计费、权限控制、日志记录等高级功能的管理实体；存储空间名称在整个OSS服务中具有全局唯一性，且不能修改；存储在OSS上的每个文件必须都包含在某个存储空间中。

新建存储空间

通过oss_create_bucket接口，可以实现创建一个存储空间，用户需要指定存储空间的名字：

```
aos_pool_t *p;
oss_request_options_t *options;
oss_acl_e oss_acl = OSS_ACL_PRIVATE;
char *bucket_name = "<您的bucket名字>";
aos_string_t bucket;
aos_table_t *resp_headers;
aos_status_t *s;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&bucket, bucket_name);

/* 创建存储空间 */
s = oss_create_bucket(options, &bucket, oss_acl, &resp_headers);
if (aos_status_is_ok(s)) {
    printf("create bucket succeeded\n");
} else {
    printf("create bucket failed\n");
}

aos_pool_destroy(p);
```

注：

- Bucket的命名规范请查看OSS 基本概念
- 由于存储空间的名字是全局唯一的，所以必须保证您的存储空间名字不与别人的重复

删除存储空间

通过oss_delete_bucket接口，可以实现删除一个存储空间，用户需要指定存储空间的名字：

```
aos_pool_t *p;
oss_request_options_t *options;
char *bucket_name = "<您的bucket名字>";
aos_string_t bucket;
aos_table_t *resp_headers;
aos_status_t *s;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&bucket, bucket_name);

/* 删除存储空间 */
s = oss_delete_bucket (options, &bucket, &resp_headers);
if (aos_status_is_ok(s)) {
    printf("delete bucket succeeded\n");
} else {
    printf("delete bucket failed\n");
}

aos_pool_destroy(p);
```

注：

- 如果该存储空间下还有文件存在，则需要先删除所有文件才能删除存储空间
- 如果该存储空间下还有未完成的上传请求，则需要通过oss_list_multipart_upload和oss_abort_multipart_upload先取消那些请求才能删除存储空间。

存储空间访问权限

用户可以设置存储空间的访问权限，允许或者禁止匿名用户对其内容进行读写。

获取存储空间的访问权限（ACL）

通过oss_get_bucket_acl接口，可以实现查看存储空间的ACL：

```
aos_pool_t *p;
oss_request_options_t *options;
char *bucket_name = "<您的bucket名字>";
aos_string_t bucket;
```

```
aos_table_t *resp_headers;
aos_status_t *s;
char *oss_acl;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&bucket, bucket_name);

/* 获取存储空间访问权限 */
s = oss_get_bucket_acl(options, &bucket,& oss_acl, &resp_headers);
if (aos_status_is_ok(s)) {
    printf("get bucket acl succeeded\n");
} else {
    printf("get bucket acl failed\n");
}

aos_pool_destroy(p);
```

设置存储空间的访问权限（ACL）

通过oss_create_bucket接口，可以实现设置存储空间的ACL：

```
aos_pool_t *p;
oss_request_options_t *options;
oss_acl_e oss_acl = OSS_ACL_PRIVATE;
char *bucket_name = "<您的bucket名字>";
aos_string_t bucket;
aos_table_t *resp_headers;
aos_status_t *s;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&bucket, bucket_name);

/* 设置存储空间访问权限 */
s = oss_put_bucket_acl(options, &bucket, oss_acl, &resp_headers);
if (aos_status_is_ok(s)) {
    printf("put bucket acl succeeded\n");
} else {
    printf("put bucket acl failed\n");
}

aos_pool_destroy(p);
```

注：

- 操作者必须是Bucket的拥有者，否则不允许设置该存储空间的访问权限。
- 存储空间的访问权限oss_acl_e是一个枚举值，可选值包括：OSS_ACL_PRIVATE、OSS_ACL_PUBLIC_READ、OSS_ACL_PUBLIC_READ_WRITE

上传文件

在OSS中，用户操作的基本数据单元是文件(Object)。单个文件的最大允许大小根据上传数据方式不同而不同，Put Object方式文件最大不能超过5GB，使用分片上传方式文件大小不能超过48.8TB。

OSS C SDK提供了丰富的文件上传接口，用户可以通过以下方式向OSS中上传文件：

- 简单上传
- 追加上传
- 断点续传上传
- 分片上传

简单上传

从内存中上传数据到OSS

通过oss_put_object_from_buffer接口，可以实现从内存中上传数据到OSS：

```
void put_object_from_buffer()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
aos_table_t *headers = NULL;
aos_table_t *resp_headers = NULL;
oss_request_options_t *options = NULL;
aos_list_t buffer;
aos_buf_t *content = NULL;
char *str = "test oss c sdk";
aos_status_t *s = NULL;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");
```

```
/* 初始化参数 */
aos_list_init(&buffer);
content = aos_buf_pack(options->pool, str, strlen(str));
aos_list_add_tail(&content->node, &buffer);

/* 上传文件 */
s = oss_put_object_from_buffer(options, &bucket, &object,
&buffer, headers, &resp_headers);

/* 判断是否上传成功 */
if (aos_status_is_ok(s)) {
printf("put object from buffer succeeded\n");
} else {
printf("put object from buffer failed\n");
}

/* 释放资源*/
aos_pool_destroy(p);
}
```

注：

- 完整代码参考：GitHub

上传本地文件到OSS

通过oss_put_object_from_file接口，并指定fliepath参数，可以实现上传一个本地文件到OSS：

```
void put_object_from_file()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
aos_table_t *headers = NULL;
aos_table_t *resp_headers = NULL;
oss_request_options_t *options = NULL;
char *filename = __FILE__;
aos_status_t *s = NULL;
aos_string_t file;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
headers = aos_table_make(options->pool, 1);
apr_table_set(headers, OSS_CONTENT_TYPE, "image/jpeg");
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");
aos_str_set(&file, filename);
```

```
/* 上传文件 */
s = oss_put_object_from_file(options, &bucket, &object, &file,
headers, &resp_headers);

/* 判断是否上传成功 */
if (aos_status_is_ok(s)) {
printf("put object from file succeeded\n");
} else {
printf("put object from file failed\n");
}

/* 释放资源*/
aos_pool_destroy(p);
}
```

注：

- 使用该方式上传最大文件不能超过5G。如果超过可以使用分片上传。
- 完整代码参考：GitHub

追加上传

OSS支持可追加的文件类型，调用时需要指定文件追加的位置，对于新创建文件，这个位置是0；对于已经存在的文件，这个位置必须是追加前文件的长度。

- 文件不存在时，调用Append Object会创建一个可追加的文件
- 文件存在时，调用Append Object会向文件末尾追加内容

Append Object以追加写的方式上传文件，通过Append Object操作创建的文件类型为Appendable Object，而通过Put Object上传的文件类型是Normal Object。

从内存中追加数据到OSS

通过oss_append_object_from_buffer接口，可以实现从内存中上传数据到OSS：

```
void append_object_from_buffer()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
char *str = "test oss c sdk";
aos_status_t *s = NULL;
int64_t position = 0;
aos_table_t *headers1 = NULL;
aos_table_t *headers2 = NULL;
aos_table_t *resp_headers = NULL;
oss_request_options_t *options = NULL;
aos_list_t buffer;
```

```
aos_buf_t *content = NULL;
char *next_append_position = NULL;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
headers1 = aos_table_make(p, 0);
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");

/* 获取起始追加位置 */
s = oss_head_object(options, &bucket, &object, headers1, &resp_headers);
if (aos_status_is_ok(s)) {
next_append_position = (char*)(apr_table_get(resp_headers,
"x-oss-next-append-position"));
position = atoi(next_append_position);
}

/* 追加文件 */
headers2 = aos_table_make(p, 0);
aos_list_init(&buffer);
content = aos_buf_pack(p, str, strlen(str));
aos_list_add_tail(&content->node, &buffer);
s = oss_append_object_from_buffer(options, &bucket, &object,
position, &buffer, headers2, &resp_headers);

/* 判断是否追加成功 */
if (aos_status_is_ok(s))
{
printf("append object from buffer succeeded\n");
} else {
printf("append object from buffer failed\n");
}

/* 释放资源*/
aos_pool_destroy(p);
}
```

注：

- 完整代码参考：GitHub

从本地文件追加数据到OSS

通过oss_append_object_from_file接口，并指定fliepath参数， 可以实现将一个本地文件的数据追加到OSS：

```
void append_object_from_file()
{
aos_pool_t *p = NULL;
```

```
aos_string_t bucket;
aos_string_t object;
aos_table_t *headers1 = NULL;
aos_table_t *headers2 = NULL;
aos_table_t *resp_headers = NULL;
oss_request_options_t *options = NULL;
char *filename = __FILE__;
aos_status_t *s = NULL;
aos_string_t file;
int64_t position = 0;
char *next_append_position = NULL;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
headers1 = aos_table_make(options->pool, 0);
headers2 = aos_table_make(options->pool, 0);
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");
aos_str_set(&file, filename);

/* 获取起始追加位置 */
s = oss_head_object(options, &bucket, &object, headers1, &resp_headers);
if(aos_status_is_ok(s)) {
next_append_position = (char*)(apr_table_get(resp_headers,
"x-oss-next-append-position"));
position = atoi(next_append_position);
}

/* 追加文件 */
s = oss_append_object_from_file(options, &bucket, &object,
position, &file, headers2, &resp_headers);

/* 判断是否追加成功 */
if (aos_status_is_ok(s)) {
printf("append object from file succeeded\n");
} else {
printf("append object from file failed\n");
}

/* 释放资源*/
aos_pool_destroy(p);
}
```

注：

- 不能对一个非Appendable Object进行Append Object操作。例如，已经存在一个同名Normal Object时，Append Object调用返回409，错误码ObjectNotAppendable。
- 对一个已经存在的Appendable Object进行Put Object操作，那么该Appendable Object会被新的Object覆盖，类型变为Normal Object。

- Head Object操作会返回x-oss-object-type，用于表明Object的类型。对于Appendable Object来说，该值为Appendable。对Appendable Object，Head Object也会返回上述的x-oss-next-append-position和x-oss-hash-crc64ecma。
- 不能使用Copy Object来拷贝一个Appendable Object，也不能改变它的服务器端加密的属性。可以使用Copy Object来改变用户自定义元信息。
- 完整代码参考：GitHub

断点续传上传

大文件上传时，网络不稳定或者其它异常发生，则整个上传就会失败。用户不得不重新上传，造成浪费资源；在网络不稳定的情况下，往往要重试多次。断点续传上传，支持并发、断点续传。断点续传上传通过oss_resumable_clt_params_t 控制并发、断点续传的行为，它有以下参数：

- part_size 分片大小，从100KB到5GB，单位是***byte***
- thread_num 并发线程数，默认为***1***
- enable_checkpoint 是否开启断点续传，**默认不开启**
- checkpoint_path checkpoint文件的路径，默认放在上传文件目录下{upload_file_path}.cp

断点续传上传的实现原理是，将要上传文件分成若干个分片分别上传，所有分片上传成功后，完成整个文件的上传。在上传的过程中会记录当前上传的进度信息（记录在checkpoint文件中），如果上传过程中某一分片上传失败，再次上传时会从checkpoint文件中记录的断点继续上传。这要求再次调用时要指定与上次相同的checkpoint文件。上传完成后，checkpoint文件会被删除。

```
void resumable_upload()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
aos_string_t filename;
aos_status_t *s = NULL;
int is_cname = 0;
aos_table_t *headers = NULL;
aos_table_t *resp_headers = NULL;
aos_list_t resp_body;
oss_request_options_t *options = NULL;
oss_resumable_clt_params_t *clt_params;

aos_pool_create(&p, NULL);
options = oss_request_options_create(p);
init_sample_request_options(options, is_cname);
headers = aos_table_make(p, 0);
aos_str_set(&bucket, BUCKET_NAME);
aos_str_set(&object, "my_key.zip");
aos_str_set(&filename, "local_big_file.zip");
aos_list_init(&resp_body);

// 断点续传
clt_params = oss_create_resumable_clt_params_content(p, 1024 * 100, 3, AOS_TRUE, NULL);
```

```
s = oss_resumable_upload_file(options, &bucket, &object, &filename, headers, NULL,
clt_params, NULL, &resp_headers, &resp_body);

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

aos_pool_destroy(p);
}
```

注：

- 断点续传上传支持设置元信息、上传回调、进度条功能。
- 完整代码参考：GitHub。

分片上传

除了通过简单上传将文件上传到OSS以外，OSS还提供了另外一种上传模式 —— 分片上传。用户可以在如下的应用场景内（但不仅限于此），使用分片上传模式，如：

- 需要支持断点上传。
- 上传超过100MB大小的文件。
- 网络条件较差，和OSS的服务器之间的连接经常断开。
- 需要流式地上传文件。上传文件之前，无法确定上传文件的大小。

易用接口完成分片上传

为了方便用户完成分片上传，下面代码通过封装后的易用接口oss_upload_file进行分片上传文件操作。

```
void test_upload_file()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
oss_request_options_t *options = NULL;
aos_status_t *s = NULL;
int part_size = 100 * 1024;
aos_string_t upload_id;
aos_string_t filepath;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");
```

```
aos_str_null(&upload_id);
aos_str_set(&filepath, __FILE__);

/* 分片上传 */
s = oss_upload_file(options, &bucket, &object, &upload_id, &filepath,
part_size, NULL);

/* 判断是否上传成功 */
if (aos_status_is_ok(s)) {
printf("upload file succeeded\n");
} else {
printf("upload file failed\n");
}

/* 释放资源*/
aos_pool_destroy(p);
}
```

注：

- 使用oss_upload_file进行分片上传文件操作,如果是一个新的分片上传操作，设置upload_id为NULL，可以通过aos_str_null(&upload_id)实现；如果是对一个已经存在的upload_id进行续传操作，设置upload_id为指定的upload_id_str，可以通过aos_str_set(&upload_id, upload_id_str)实现。
- 使用oss_upload_file进行分片上传文件操作时，需要指定分片大小， 分片大小要大于100KB。分片号码的范围是1~10000，如果在切分文件时，发现分片号码的范围超出这个范围，oss_upload_file将自动调整分片大小。
- 使用oss_upload_file进行分片上传文件操作，如果指定的upload_id已经存在，那么本次分片上传的分片大小要和指定upload_id的分片大小保持一致。
- 完整代码参考：GitHub

分步完成分片上传

分步完成分片上传的好处是灵活，一般的流程如下:

1. 初始化一个分片上传任务（oss_init_multipart_upload）
2. 逐个或并行上传分片（oss_upload_part_from_file）
3. 完成上传（oss_complete_multipart_upload）

初始化

初始化一个分片上传事件

```
void init_multipart_upload()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
aos_table_t *headers = NULL;
```

```
aos_table_t *resp_headers = NULL;
oss_request_options_t *options = NULL;
aos_string_t upload_id;
oss_upload_file_t *upload_file = NULL;
aos_status_t *s = NULL;
oss_list_upload_part_params_t *params = NULL;
aos_list_t complete_part_list;
oss_list_part_content_t *part_content = NULL;
aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
headers = aos_table_make(p, 1);
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");

/* 初始化分片上传，获取一个上传ID */
s = oss_init_multipart_upload(options, &bucket, &object,
&upload_id, headers, &resp_headers);

/* 判断是否初始化分片上传成功 */
if (aos_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, upload_id:%.*s\n",
upload_id.len, upload_id.data);
}

/* 上传每一个分片，代码参考下一节，这里省略*/
/* 完成分片上传，代码参考后面章节，这里省略*/

/* 释放资源*/
aos_pool_destroy(p);
}
```

注：

- 返回结果中含有uplolad_id ，它是区分分片上传事件的唯一标识，在后面的操作中，我们将用到它。
- 上述代码只是演示如何初始化分片上传，只是部分代码，完整的代码参考下面的GitHub链接。
- 完整代码参考：GitHub
- 1.0.0版本中的参数顺序是...headers, upload_id...，2.0.0版本中的参数顺序是...upload_id，headers...

本地上传分片

接着，把本地文件分片上传。

```
void multipart_upload_file()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
int is_cname = 0;
aos_table_t *headers = NULL;
aos_table_t *complete_headers = NULL;
aos_table_t *resp_headers = NULL;
oss_request_options_t *options = NULL;
aos_string_t upload_id;
oss_upload_file_t *upload_file = NULL;
aos_status_t *s = NULL;
oss_list_upload_part_params_t *params = NULL;
aos_list_t complete_part_list;
oss_list_part_content_t *part_content = NULL;
oss_complete_part_content_t *complete_part_content = NULL;
int part_num1 = 1;
int part_num2 = 2;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
headers = aos_table_make(p, 1);
resp_headers = aos_table_make(options->pool, 5);
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");

/* 初始化分片上传，获取upload id，代码参考前面章节，这里省略*/

/* 上传第一个分片 */
upload_file = oss_create_upload_file(p);
aos_str_set(&upload_file->filename, MULTIPART_UPLOAD_FILE_PATH);
upload_file->file_pos = 0;
upload_file->file_last = 200 * 1024; //200k
s = oss_upload_part_from_file(options, &bucket, &object, &upload_id,
part_num1, upload_file, &resp_headers);

/* 判断是否上传分片成功 */
if (aos_status_is_ok(s)) {
printf("Multipart upload from file succeeded\n");
} else {
printf("Multipart upload from file failed\n");
}

/* 上传第二个分片 */
upload_file->file_pos = 200 *1024;//remain content start pos
upload_file->file_last = get_file_size(MULTIPART_UPLOAD_FILE_PATH);
s = oss_upload_part_from_file(options, &bucket, &object, &upload_id,
part_num2, upload_file, &resp_headers);

/* 判断是否上传分片成功 */
```

```
if (aos_status_is_ok(s)) {
printf("Multipart upload from file succeeded\n");
} else {
printf("Multipart upload from file failed\n");
}

/* 完成分片上传，代码参考下一章节，这里省略*/

/* 释放资源*/
aos_pool_destroy(p);
}
```

上面程序的核心是调用oss_upload_part_from_file接口来上传每一个分片。

注：

- oss_upload_part_from_file接口要求除最后一个分片以外，其他的分片大小都要大于100KB。
- 分片号码的范围是1~10000。如果超出这个范围，OSS将返回InvalidArgument的错误码。
- 每次上传分片时都要把流定位到此次上传分片开头所对应的位置。
- 上述代码只是演示如何初始化分片上传，只是部分代码，完整的代码参考下面的GitHub链接。
- 完整代码参考：GitHub

获取已上传的分片，完成分片上传

```
void complete_multipart_upload()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
aos_table_t *complete_headers = NULL;
aos_table_t *resp_headers = NULL;
oss_request_options_t *options = NULL;
aos_string_t upload_id;
aos_status_t *s = NULL;
oss_list_upload_part_params_t *params = NULL;
aos_list_t complete_part_list;
oss_list_part_content_t *part_content = NULL;
oss_complete_part_content_t *complete_part_content = NULL;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
headers = aos_table_make(p, 1);
resp_headers = aos_table_make(options->pool, 5);
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");
```

```
/* 初始化分片上传，获取一个Upload Id，代码参考上面章节，这里省略*/
/* 上传每个分片，代码参考上面章节，这里省略*/

/* 获取已经上传的分片
* 调用oss_complete_multipart_upload接口时需要每个分片的ETag值，这个值有两种获取途径：
* 第一种是上传每个分片的时候，返回结果里面会包含这个分片的ETag值，可以保存下来，等后面使用；
* 第二种是通过调用oss_list_upload_part接口获取已经上传的分片的ETag值。
* 下面演示的是第二种方式
*/
params = oss_create_list_upload_part_params(p);
params->max_ret = 1000;
aos_list_init(&complete_part_list);
s = oss_list_upload_part(options, &bucket, &object, &upload_id,
params, &resp_headers);

/* 判断是否获取分片列表成功 */
if (aos_status_is_ok(s)) {
printf("List multipart succeeded\n");
} else {
printf("List multipart failed\n");
}

aos_list_for_each_entry(part_content, &params->part_list, node) {
complete_part_content = oss_create_complete_part_content(p);
aos_str_set(&complete_part_content->part_number,
part_content->part_number.data);
aos_str_set(&complete_part_content->etag, part_content->etag.data);
aos_list_add_tail(&complete_part_content->node, &complete_part_list);
}

/* 完成分片上传 */
s = oss_complete_multipart_upload(options, &bucket, &object, &upload_id,
&complete_part_list, complete_headers, &resp_headers);

/* 判断完成分片上传是否成功 */
if (aos_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");
}

/* 释放资源 */
aos_pool_destroy(p);
}
```

注：

- 2.0.0相对于1.0.0版本，oss_complete_multipart_upload接口增加了headers参数，用来在完成时修改headers值
- 完整代码参考：GitHub

取消分片上传事件

```
void abort_multipart_upload()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
aos_table_t *headers = NULL;
aos_table_t *resp_headers = NULL;
oss_request_options_t *options = NULL;
aos_string_t upload_id;
aos_status_t *s = NULL;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
headers = aos_table_make(p, 1);
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");

/* 初始化分片上传，获取一个Upload Id*/
s = oss_init_multipart_upload(options, &bucket, &object,
&upload_id, headers, &resp_headers);

if (aos_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");
}

/* 取消这次分片上传 */
s = oss_abort_multipart_upload(options, &bucket, &object, &upload_id,
&resp_headers);

/* 判断取消分片上传是否成功 */
if (aos_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");
}

/* 释放资源 */
aos_pool_destroy(p);
}
```

注：

- 当一个分片上传事件被中止后，就不能再使用这个upload_id做任何操作，已经上传的分片数据也会被删除。
- 完整代码参考：GitHub

设置元信息

文件元信息(Object Meta)，是对用户上传到OSS的文件的属性描述，分为两种：HTTP标准属性（HTTP Headers）和用户自定义元信息（User Meta）。 文件元信息可以在上传（流上传、文件上传、追加上传、分片上传、断点续传），或拷贝时设置。元信息的名称大小写不敏感。更多文件元信息的介绍，请参看文件元信息。

HTTP标准属性

常用的HTTP标准属性说明如下，更详细的说明请参考RFC2616。

名称	描述	默认值
Content-MD5	文件数据校验，设置了该值后OSS会启用文件内容MD5校验，把您提供的MD5与文件的MD5比较，不一致会返回错误	无
Content-Type	文件的MIME，定义文件的类型及网页编码，决定浏览器将以什么形式、什么编码读取文件。如果用户没有指定则根据Key或文件名的扩展名生成，如果没有扩展名则填默认值	application/octet-stream
Content-Disposition	指示MIME用户代理如何显示附加的文件，打开或下载，及文件名称	无
Content-Length	上传的数据的长度，超过长度的数据会被截断，不足为实际值	数据的实际长度
Expires	缓存过期时间，OSS未使用，格式是格林威治时间（GMT）	无
Cache-Control	指定该Object被下载时的网页的缓存行为	无
Expect	HTTP客户端期望特定的行为，目前支持100-continue	与libcurl实现相关
Transfer-Encoding	传输数据时候编码方式，目前支持chunked	与libcurl实现相关

```
void put_object_from_file()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
aos_table_t *headers = NULL;
aos_table_t *resp_headers = NULL;
```

```
oss_request_options_t *options = NULL;
char *filename = __FILE__;
aos_status_t *s = NULL;
aos_string_t file;
aos_pool_create(&p, NULL);
/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);
/* 初始化参数 */
headers = aos_table_make(options->pool, 2);
apr_table_set(headers, OSS_CONTENT_TYPE, "image/jpeg");
apr_table_set(headers, OSS_CONTENT_MD5, "eB5eJF1ptWaXm4bijSPyxw==");
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");
aos_str_set(&file, filename);
/* 上传文件 */
s = oss_put_object_from_file(options, &bucket, &object, &file,
headers, &resp_headers);
/* 判断是否上传成功 */
if (aos_status_is_ok(s)) {
printf("put object from file succeeded\n");
} else {
printf("put object from file failed\n");
}
/* 释放资源*/
aos_pool_destroy(p);
}
```

注：

- 数据、文件MD5值的计算请参看Put Object。
- HTTP标准属性会改变数据传输行为，**请在明确知晓其行为后设置**。

用户自定义元信息

为了便于用户对文件进行更多描述，OSS中规定所有以x-oss-meta-为前缀的参数视为User Meta，一个文件可以有多个类似的参数。User Meta会在下载GetObject或HeadObject的时，在HTTP头部中返回。

```
void put_object_from_file()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
aos_table_t *headers = NULL;
aos_table_t *resp_headers = NULL;
oss_request_options_t *options = NULL;
char *filename = __FILE__;
aos_status_t *s = NULL;
aos_string_t file;
aos_pool_create(&p, NULL);
/* 创建并初始化options */
options = oss_request_options_create(p);
```

```
init_options(options);
/* 初始化参数 */
headers = aos_table_make(options->pool, 4);
apr_table_set(headers, OSS_CONTENT_TYPE, "image/jpeg");
apr_table_set(headers, OSS_CONTENT_MD5, "eB5eJF1ptWaXm4bijSPyxw==");
apr_table_set(headers, "x-oss-meta-author", "mingdi");
apr_table_set(headers, "x-oss-meta-category", "computer");
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");
aos_str_set(&file, filename);
/* 上传文件 */
s = oss_put_object_from_file(options, &bucket, &object, &file,
headers, &resp_headers);
/* 判断是否上传成功 */
if (aos_status_is_ok(s)) {
printf("put object from file succeeded\n");
} else {
printf("put object from file failed\n");
}
/* 释放资源*/
aos_pool_destroy(p);
}
```

上传回调

OSS在上传文件完成的时，可以提供**回调**（Callback）给应用服务器。用户只需要在发送给OSS的请求中携带相应的回调参数，即能实现回调。支持上传回调接口有：PutObject、PostObject、CompleteMultipartUpload。您想了解上传回调的更详细信息，请参考**上传回调**。下面以PutObject为例说明上传回调的用法。

```
void put_object_from_buffer_with_callback()
{
aos_pool_t *p = NULL;
char *str = "test oss c sdk";
aos_status_t *s = NULL;
int is_cname = 0;
aos_string_t bucket;
aos_string_t object;
aos_table_t *headers = NULL;
oss_request_options_t *options = NULL;
aos_table_t *resp_headers = NULL;
aos_list_t resp_body;
aos_list_t buffer;
aos_buf_t *content;
char *buf = NULL;
int64_t len = 0;
int64_t size = 0;
int64_t pos = 0;
char b64_buf[1024];
int b64_len;

/* JSON format */
char *callback = "{"
```

```
"\"callbackUrl\":\"http://callback.oss-demo.com:23450\","
"\"callbackHost\":\"oss-cn-hangzhou.aliyuncs.com\","
"\"callbackBody\":\"bucket=${bucket}&object=${object}&size=${size}&mimeType=${mimeType}\","
"\"callbackBodyType\":\"application/x-www-form-urlencoded\""
"}";

/* init sample */
aos_pool_create(&p, NULL);
options = oss_request_options_create(p);
init_sample_request_options(options, is_cname);

aos_str_set(&bucket, BUCKET_NAME);
aos_str_set(&object, OBJECT_NAME);
aos_list_init(&resp_body);
aos_list_init(&buffer);

content = aos_buf_pack(options->pool, str, strlen(str));
aos_list_add_tail(&content->node, &buffer);

/* put call into header */
b64_len = aos_base64_encode((unsigned char*)callback, strlen(callback), b64_buf);
b64_buf[b64_len] = '\0';

headers = aos_table_make(p, 1);
apr_table_set(headers, OSS_CALLBACK, b64_buf);

/* test put object */
s = oss_do_put_object_from_buffer(options, &bucket, &object, &buffer,
headers, NULL, NULL, &resp_headers, &resp_body);
if (aos_status_is_ok(s)) {
printf("put object from buffer succeeded\n");
} else {
printf("put object from buffer failed\n");
}

/* get buffer len */
len = aos_buf_list_len(&resp_body);
buf = (char *)aos_pcalloc(p, (apr_size_t)(len + 1));
buf[len] = '\0';

/* copy buffer content to memory */
aos_list_for_each_entry(aos_buf_t, content, &resp_body, node) {
size = aos_buf_size(content);
memcpy(buf + pos, content->pos, (size_t)size);
pos += size;
}

aos_pool_destroy(p);
}
```

注：

- oss_do_complete_multipart_upload、oss_resumable_upload_file都支持上传回调，代码请参考：GitHub。
- **上传回调** 的调试方法和错误排除，请参考**上传回调错误及排除**。

进度条

OSS C SDK支持进度条功能，指示上传/下载的进度。下面的代码以PutObject为例，说明进度条功能的使用方法。

```
void percentage(int64_t consumed_bytes, int64_t total_bytes)
{
assert(total_bytes >= consumed_bytes);
printf("%%%" APR_INT64_T_FMT "\n", consumed_bytes * 100 / total_bytes);
}

void put_and_get_from_file_with_progress()
{
aos_pool_t *p = NULL;
aos_status_t *s = NULL;
int is_cname = 0;
aos_string_t bucket;
aos_string_t object;
aos_string_t filename;
oss_request_options_t *options = NULL;
aos_table_t *resp_headers = NULL;
aos_list_t resp_body;
char *download_filename = "get_object_to_local_file.txt";

/* init test*/
aos_pool_create(&p, NULL);
options = oss_request_options_create(p);
init_sample_request_options(options, is_cname);

aos_str_set(&bucket, BUCKET_NAME);
aos_str_set(&object, OBJECT_NAME);
aos_str_set(&filename, __FILE__);
aos_list_init(&resp_body);

/* put object */
s = oss_do_put_object_from_file(options, &bucket, &object, &filename, NULL, NULL, percentage, &resp_headers,
&resp_body);
if (aos_status_is_ok(s)) {
printf("put object from file succeeded\n");
} else {
printf("put object from file failed\n");
aos_pool_destroy(p);
return;
}

aos_pool_destroy(p);
}
```

注：

- oss_do_put_object_from_buffer、oss_do_put_object_from_file、oss_do_append_object_from_buffer、oss_do_append_object_from_file、oss_do_upload_part_from_buffer、oss_do_upload_part_from_file、oss_resumable_upload_file支持进度条功能，代码请参考：GitHub。

下载文件

OSS C SDK提供了丰富的文件下载接口，用户可以通过以下方式从OSS下载文件：

- 下载文件到内存
- 下载文件到本地文件
- 分段下载文件
- 断点续传下载

下载文件到内存

通过oss_get_object_to_buffer接口，可以实现将文件下载到内存中：

```
aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *params;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的object名字>";
aos_string_t bucket;
aos_string_t object;
aos_list_t buffer;
aos_buf_t *content;
char *buf;
int64_t len = 0;
int64_t size = 0;
int64_t pos = 0;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&object, object_name);
aos_str_set(&bucket, bucket_name);
```

```
headers = aos_table_make(p, 0);
params = aos_table_make(p, 0);

/* 下载文件 */
aos_list_init(&buffer);
s = oss_get_object_to_buffer(options, &bucket, &object, &buffer, headers, params, &resp_headers);
if (aos_status_is_ok(s)) {
printf("get object succeeded\n");

/* 将下载内容拷贝到buffer中*/
len = aos_buf_list_len(&buffer);
buf = aos_pcalloc(p, len + 1);
buf[len] = '\0';
aos_list_for_each_entry(content, &buffer, node) {
size = aos_buf_size(content);
memcpy(buf + pos, content->pos, size);
pos += size;
}
} else {
printf("get object failed\n");
}

aos_pool_destroy(p);
```

注：

- 2.0.0相对于1.0.0版本，oss_get_object_to_buffer接口增加了params参数，同时headers和params允许为NULL，1.0.0及其之前版本不支持为NULL
- oss_get_object_to_buffer_by_url和oss_get_object_to_file_by_url参数也增加了params参数
- 完整代码参考：GitHub

下载文件到本地文件

通过oss_get_object_to_file接口，可以实现将文件下载到指定文件：

```
aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *params;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的object名字>";
char *filepath = "<本地文件路径>";
aos_string_t bucket;
aos_string_t object;
aos_string_t file;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
```

```
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
aos_str_set(&file, filepath);
headers = aos_table_make(p, 0);
params = aos_table_make(p, 0);

/* 下载文件 */
s = oss_get_object_to_file(options, &bucket, &object, &file, headers, params, &resp_headers);
if (aos_status_is_ok(s)) {
printf("get object succeeded\n");
} else {
printf("get object failed\n");
}
aos_pool_destroy(p);
```

注：

- 2.0.0相对于1.0.0版本，oss_get_object_to_file接口增加了params参数，同时headerss和params允许为NULL，1.0.0及其之前版本不支持为NULL。
- 如果本地有同名文件 filepath，本地文件将会被覆盖；
- 完整代码参考：GitHub

分段下载文件

通过设置Range指定返回文件的传输范围，可以实现文件的分段下载。以下代码实现分段下载文件到内存中：

```
aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *params;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的object名字>";
aos_string_t bucket;
aos_string_t object;
aos_list_t buffer;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
```

```
params = NULL;
headers = aos_table_make(p, 1);

/* 设置Range，读取文件的指定范围，bytes=20-100包括第20和第100个字符 */
apr_table_set(headers, "Range", "bytes=20-100");
aos_list_init(&buffer);

/* 分片下载文件 */
s = oss_get_object_to_buffer(options, &bucket, &object, headers, params, &buffer, &resp_headers);
if (aos_status_is_ok(s)) {
printf("get object succeeded\n");
} else {
printf("get object failed\n");
}

aos_pool_destroy(p);
```

断点续传下载

大文件下载时，如果网络不稳定或者发生其它异常，则整个下载会失败。用户不得不重新下载，造成浪费资源，而且在网络不稳定的情况下，往往要重试多次。断点续传下载，支持并发、断点续传，通过oss_resumable_clt_params_t 控制断点续传的行为，它有以下参数：

- part_size 分片大小，从1B到5GB，单位是***byte***
- thread_num 并发线程数，默认为***1***
- enable_checkpoint 是否开启断点续传，**默认不开启**
- checkpoint_path checkpoint文件的路径，默认放在上传文件目录{upload_file_path}.cp下

断点续传实现的原理是，将要下载文件分成若干个分片分别下载，所有分片都下载成功后，完成整个文件的下载。在下载的过程中会记录当前下载的进度信息（记录在checkpoint文件中），如果下载过程中某一分片下载失败，再次下载时会从checkpoint文件中记录的点继续下载。这要求再次调用时要指定与上次相同的checkpoint文件。下载完成后，checkpoint文件会被删除。

```
void resumable_download_with_resumable()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
aos_string_t filename;
aos_status_t *s = NULL;
int is_cname = 0;
aos_table_t *headers = NULL;
aos_table_t *resp_headers = NULL;
oss_request_options_t *options = NULL;
oss_resumable_clt_params_t *clt_params;

aos_pool_create(&p, NULL);
options = oss_request_options_create(p);
init_sample_request_options(options, is_cname);
headers = aos_table_make(p, 0);
aos_str_set(&bucket, BUCKET_NAME);
```

```
aos_str_set(&object, "my_key_2.zip");
aos_str_set(&filename, "local_big_file_2.zip");

// download
clt_params = oss_create_resumable_clt_params_content(p, 1024 * 100, 3, AOS_TRUE, NULL);
s = oss_resumable_download_file(options, &bucket, &object, &filename, headers, NULL,
clt_params, NULL, &resp_headers);

if (aos_status_is_ok(s)) {
printf("download succeeded\n");
} else {
printf("download failed\n");
}

aos_pool_destroy(p);
}
```

注：

- 断点续传下载支持进度条功能。
- 完整代码参考：GitHub。

进度条

OSS C SDK支持进度条功能，指示上传/下载的进度。下面的代码以GetObject为例，说明进度条功能的使用方法。

```
void percentage(int64_t consumed_bytes, int64_t total_bytes)
{
assert(total_bytes >= consumed_bytes);
printf("%%%" APR_INT64_T_FMT "\n", consumed_bytes * 100 / total_bytes);
}

void put_and_get_from_file_with_progress()
{
aos_pool_t *p = NULL;
aos_status_t *s = NULL;
int is_cname = 0;
aos_string_t bucket;
aos_string_t object;
aos_string_t filename;
oss_request_options_t *options = NULL;
aos_table_t *resp_headers = NULL;
aos_list_t resp_body;
char *download_filename = "get_object_to_local_file.txt";

/* init test*/
aos_pool_create(&p, NULL);
options = oss_request_options_create(p);
init_sample_request_options(options, is_cname);

aos_str_set(&bucket, BUCKET_NAME);
```

```
aos_str_set(&object, OBJECT_NAME);
aos_str_set(&filename, __FILE__);
aos_list_init(&resp_body);

/* put object */
s = oss_do_put_object_from_file(options, &bucket, &object, &filename, NULL, NULL, percentage, &resp_headers,
&resp_body);
if (aos_status_is_ok(s)) {
printf("put object from file succeeded\n");
} else {
printf("put object from file failed\n");
aos_pool_destroy(p);
return;
}

aos_pool_destroy(p);

/* get object */
aos_pool_create(&p, NULL);
options = oss_request_options_create(p);
init_sample_request_options(options, is_cname);
aos_str_set(&filename, download_filename);

s = oss_do_get_object_to_file(options, &bucket, &object, NULL, NULL, &filename, percentage, NULL);
if (aos_status_is_ok(s)) {
printf("get object to file succeeded\n");
} else {
printf("get object to file failed\n");
}

aos_pool_destroy(p);
}
```

注：

- oss_do_get_object_to_buffer、oss_do_put_object_from_file支持进度条功能，代码请参考：GitHub。

管理文件

查看所有文件

通过oss_list_object接口，可以列出当前存储空间下的所有文件。

```
aos_pool_t *p;
oss_request_options_t *options;
```

```
aos_status_t *s;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
aos_string_t bucket;
oss_list_object_params_t *params;
oss_list_object_content_t *content;
int max_ret = 1000;
char *key;

aos_pool_create(&p, NULL);

options = oss_request_options_create(p);
init_options(options);

aos_str_set(&bucket, bucket_name);
params = oss_create_list_object_params(p);
params->max_ret = max_ret;
aos_str_set(&params->prefix, "<查看文件的前缀>");
aos_str_set(&params->delimiter, "<查看文件的分隔符>");
aos_str_set(&params->marker, "<查看文件的起点>");

s = oss_list_object(options, &bucket, params, &resp_headers);
if (aos_status_is_ok(s)) {
printf("get object succeeded\n");

/* 下载文件 */
aos_list_for_each_entry(content, &params->object_list, node) {
key = apr_psprintf(p, "%.*s", content->key.len, content->key.data);
}
} else {
printf("get object failed\n");
}

aos_pool_destroy(p);
```

注：

- 默认情况下，如果存储空间中的文件数量大于1000，则只会返回1000个文件， 且返回结果中truncated为true，并返回next_marker作为下此读取的起点。若想增大返回文件数目，可以修改 max_ret参数，或者使用marker参数分次读取。
- 完整代码参考：GitHub

创建模拟目录

OSS是基于对象的存储服务，没有目录的概念。创建模拟目录本质上来说是创建了一个size为0的文件。对于这个文件照样可以上传下载,只是控制台会对以”/“结尾的文件以文件夹的方式展示，所以用户可以使用上述方式来实现创建模拟目录。

```
aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
```

```
aos_table_t *headers;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的文件夹名字/>";
aos_string_t bucket;
aos_string_t object;
char *data = "";
aos_list_t buffer;
aos_buf_t *content;

aos_pool_create(&p, NULL);

options = oss_request_options_create(p);
init_options(options);

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
headers = aos_table_make(p, 0);

aos_list_init(&buffer);
content = aos_buf_pack(options->pool, data, strlen(data));
aos_list_add_tail(&content->node, &buffer);
s = oss_put_object_from_buffer(options, &bucket, &object, &buffer, headers, &resp_headers);
if (aos_status_is_ok(s)) {
printf("create dir succeeded\n");
} else {
printf("create dir failed\n");
}
aos_pool_destroy(p);
```

注：

- 完整代码参考：GitHub

设定Object的Http Header

向OSS上传文件时，除了文件内容，还可以指定文件的一些属性信息，称为“元信息”。这些信息在上传时与文件一起存储，在下载时与文件一起返回。下面代码为文件设置了过期时间：

```
aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的object名字>";
aos_string_t bucket;
aos_string_t object;
char *data = "<object content>";
aos_list_t buffer;
aos_buf_t *content;
```

```
aos_pool_create(&p, NULL);

options = oss_request_options_create(p);
init_options(options);

aos_str_set(&object, object_name);
aos_str_set(&bucket, bucket_name);
headers = aos_table_make(p, 0);

/* 设置http header */
headers = aos_table_make(p, 1);
apr_table_set(headers, "Expires", "Fri, 28 Feb 2012 05:38:42 GMT");

/* 读取数据到buffer中 */
aos_list_init(&buffer);
content = aos_buf_pack(options->pool, data, strlen(data));
aos_list_add_tail(&content->node, &buffer);
s = oss_put_object_from_buffer(options, &bucket, &object, &buffer, headers, &resp_headers);
if (aos_status_is_ok(s)) {
printf("put object succeeded\n");
} else {
printf("put object failed\n");
}

aos_pool_destroy(p);
```

注：

- 可以设置Http Header有：Cache-Control 、 Content-Disposition 、Content-Encoding、Expires 。它们的相关介绍见 RFC2616 。
- 完整代码参考：GitHub

设置User Meta

OSS支持用户自定义Meta信息对文件进行描述。比如：

```
aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的object名字>";
aos_string_t bucket;
aos_string_t object;
char *data = "<object content>";
aos_list_t buffer;
aos_buf_t *content;

aos_pool_create(&p, NULL);

options = oss_request_options_create(p);
```

```
init_options(options);

aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
headers = aos_table_make(p, 1);

/* 设置用户自定义meta */
apr_table_set(headers, "x-oss-meta-author", "oss");

/* 上传数据到OSS中 */
aos_list_init(&buffer);
content = aos_buf_pack(options->pool, data, strlen(data));
aos_list_add_tail(&content->node, &buffer);
s = oss_put_object_from_buffer(options, &bucket, &object, &buffer, headers, &resp_headers);
if (aos_status_is_ok(s)) {
printf("put object succeeded\n");
} else {
printf("put object failed\n");
}
aos_pool_destroy(p);
```

注：

- 因为文件元信息在上传/下载时是附在HTTP Headers中， HTTP协议规定不能包含复杂字符。
- 一个文件可以有多个类似的参数，但所有的user meta总大小不能超过8KB。

获取文件的元数据

有时候，再向OSS上传文件后或者读取文件前需要获取文件的一些元数据，比如长度，文件类型等，这时候可以通过oss_object_head接口获取文件的元数据。下面代码获取文件的长度和文件类型：

```
void head_object()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
oss_request_options_t *options = NULL;
aos_table_t *headers = NULL;
aos_table_t *resp_headers = NULL;
aos_status_t *s = NULL;
char *content_length_str = NULL;
char *object_type = NULL;
int64_t content_length = 0;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&bucket, "<您的bucket名字>");
```

```
aos_str_set(&object, "<您的object名字>");
headers = aos_table_make(p, 0);

/* 获取文件元数据 */
s = oss_head_object(options, &bucket, &object, headers, &resp_headers);

if (aos_status_is_ok(s)) {
/* 获取文件长度 */
content_length_str = (char*)apr_table_get(resp_headers, OSS_CONTENT_LENGTH);
if (content_length_str != NULL) {
content_length = atoll(content_length_str);
}

/* 获取文件的类型 */
object_type = (char*)apr_table_get(resp_headers, OSS_OBJECT_TYPE);

printf("head object succeeded, object type:%s, content_length:%ld\n",
object_type, content_length);
} else {
printf("head object failed\n");
}

aos_pool_destroy(p);
}
```

注：

- oss_head_object接口不会去下载文件内容，只会读取文件的元数据，包括系统级别和用户自定义的元数据
- 完整代码参考：GitHub

拷贝文件

在同一个region中，用户可以对有操作权限的文件进行拷贝操作。以下代码通过oss_copy_object接口实现拷贝一个文件：

```
aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t * headers;
aos_table_t *resp_headers;
char *source_bucket_name = "<您的源bucket名字>";
char *source_object_name = "<您的源object名字>";
char *dest_bucket_name = "<您的目的bucket名字>";
char *dest_object_name = "<您的目的object名字>";
aos_string_t source_bucket;
aos_string_t source_object;
aos_string_t dest_bucket;
aos_string_t dest_object;

aos_pool_create(&p, NULL);
```

```
options = oss_request_options_create(p);
init_options(options);

aos_str_set(&source_bucket, source_bucket_name);
aos_str_set(&source_object, source_object_name);
aos_str_set(&dest_bucket, dest_bucket_name);
aos_str_set(&dest_object, dest_object_name);
headers = aos_table_make(p, 0);

/* 拷贝文件 */
s = oss_copy_object(options, &source_bucket, &source_object, &dest_bucket, &dest_object, headers,
&resp_headers);
if (aos_status_is_ok(s)) {
printf("copy object succeeded\n");
} else {
printf("copy object failed\n");
}

aos_pool_destroy(p);
```

注：

- 需要注意的是源bucket与目的bucket必须属于同一region。
- 如果拷贝文件超过1G，建议使用分片拷贝文件方式进行拷贝。

分片拷贝文件

当拷贝一个大于500MB的文件，建议通过oss_upload_part_copy接口来进行拷贝。以下代码实现分片拷贝一个文件

```
aos_pool_t *p;
oss_request_options_t *options;
char *source_bucket_name = "<您的源bucket名字>";
char *source_object_name = "<您的源object名字>";
char *dest_bucket_name = "<您的目的bucket名字>";
char *dest_object_name = "<您的目的object名字>";
aos_string_t dest_bucket;
aos_string_t dest_object;
aos_string_t upload_id;
aos_table_t *init_headers;
aos_table_t *copy_headers;
aos_table_t *list_part_resp_headers;
aos_table_t *complete_resp_headers
aos_table_t *resp_headers;
aos_status_t *s;
oss_list_upload_part_params_t *list_upload_part_params;
oss_upload_part_copy_params_t *upload_part_copy_params1;
aos_list_t complete_part_list;
oss_list_part_content_t *part_content;
oss_complete_part_content_t *complete_content;
int part1 = 1;
```

```
int64_t range_start1 = 0;
int64_t range_end1 = 6000000;//not less than 5MB
int max_ret = 1000;

aos_pool_create(&p, NULL);

options = oss_request_options_create(p);
init_options(options);

aos_str_set(&dest_bucket, dest_bucket_name);
aos_str_set(&dest_object, dest_object_name);
init_headers = aos_table_make(p, 0);
s = oss_init_multipart_upload(options, &dest_bucket, &dest_object, init_headers, &upload_id, &resp_headers);

/* 拷贝第一个分片数据 */
upload_part_copy_params1 = oss_create_upload_part_copy_params(p);
aos_str_set(&upload_part_copy_params1->source_bucket, source_bucket_name);
aos_str_set(&upload_part_copy_params1->source_object, source_object_name);
aos_str_set(&upload_part_copy_params1->dest_bucket, dest_bucket_name);
aos_str_set(&upload_part_copy_params1->dest_object, dest_object_name);
aos_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;
copy_headers = aos_table_make(p, 0);
s = oss_upload_part_copy(options, upload_part_copy_params1, copy_headers, &resp_headers);
if (aos_status_is_ok(s)) {
printf("upload part copy succeeded\n");
} else {
printf("upload part copy failed\n");
}

/* 继续拷贝剩余的分片，这里省略*/
...

/* 列出分片 */
list_upload_part_params = oss_create_list_upload_part_params(p);
list_upload_part_params->max_ret = max_ret;
aos_list_init(&complete_part_list);
s = oss_list_upload_part(options, &dest_bucket, &dest_object, &upload_id,
list_upload_part_params, &list_part_resp_headers);
aos_list_for_each_entry(part_content, &list_upload_part_params->part_list, node) {
complete_content = oss_create_complete_part_content(p);
aos_str_set(&complete_content->part_number, part_content->part_number.data);
aos_str_set(&complete_content->etag, part_content->etag.data);
aos_list_add_tail(&complete_content->node, &complete_part_list);
}

/* 完成分片拷贝 */
s = oss_complete_multipart_upload(options, &dest_bucket, &dest_object, &upload_id, &complete_part_list,
&complete_resp_headers);
if (aos_status_is_ok(s)) {
printf("complete multipart upload succeeded\n");
} else {
printf("complete multipart upload failed\n");
}
```

```
aos_pool_destroy(p);
```

删除文件

通过oss_delete_object接口，可以实现删除某个文件：

```
void delete_object()
{
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
oss_request_options_t *options = NULL;
aos_table_t *resp_headers = NULL;
aos_status_t *s = NULL;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");

/* 删除文件 */
s = oss_delete_object(options, &bucket, &object, &resp_headers);

/* 判断是否删除成功 */
if (aos_status_is_ok(s)) {
printf("delete object succeed\n");
} else {
printf("delete object failed\n");
}

/* 释放资源*/
aos_pool_destroy(p);
}
```

注：

- 完整代码参考：GitHub

批量删除文件

通过oss_delete_objects接口，可以实现删除一批文件，用户可以通过is_quiet参数来指定是否返回删除的结果：

```
aos_pool_t *p;
aos_status_t *s;
```

```
aos_table_t *resp_headers;
oss_request_options_t *options;
char *bucket_name = "<您的bucket名字>";
char *object_name1 = "<您的object名字1>";
char *object_name2 = "<您的object名字2>";
aos_string_t bucket;
aos_string_t object1;
aos_string_t object2;
oss_object_key_t *content1;
oss_object_key_t *content2;
aos_list_t object_list;
aos_list_t deleted_object_list;
int is_quiet = 1;

options = oss_request_options_create(p);
init_options(options);

aos_str_set(&bucket, bucket_name);
aos_str_set(&object1, object_name1);
aos_str_set(&object2, object_name2);

/* 构建待删除文件列表 */
aos_list_init(&object_list);
aos_list_init(&deleted_object_list);
content1 = oss_create_oss_object_key(p);
aos_str_set(&content1->key, object_name1);
aos_list_add_tail(&content1->node, &object_list);
content2 = oss_create_oss_object_key(p);
aos_str_set(&content2->key, object_name2);
aos_list_add_tail(&content2->node, &object_list);

/* 删除多个文件 */
s = oss_delete_objects(options, &bucket, &object_list, is_quiet, &resp_headers, &deleted_object_list);
if (aos_status_is_ok(s)) {
printf("delete objects succeeded\n");
} else {
printf("delete objects failed\n");
}

aos_pool_destroy(p);
```

批量删除指定前缀的文件

通过oss_delete_objects_by_prefix接口，可以实现删除一批指定前缀的文件：

```
aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
char *prefix_str = "<删除文件的前缀>";
aos_string_t bucket;
aos_string_t prefix;
```

```
aos_pool_create(&p, NULL);

options = oss_request_options_create(p);
init_options(options);

aos_str_set(&bucket, bucket_name);
aos_str_set(&prefix, prefix_str);

/* 删除满足特定前缀的文件*/
s = oss_delete_objects_by_prefix(options, &bucket, &prefix);
if (aos_status_is_ok(s)) {
printf("delete objects by prefix succeeded\n");
} else {
printf("delete objects by prefix failed\n");
}

aos_pool_destroy(p);
```

注：

- 批量删除指定前缀的文件可以实现删除目录的功能，比如删除dir目录，可以通过设置prefix的值为 **dir/** 实现。
- 如果设置prefix的值为空字符串（“”）或者NULL，将会删除整个bucket，请谨慎使用。

授权访问

使用STS服务临时授权

OSS可以通过阿里云STS服务，临时进行授权访问。使用STS时请按以下步骤进行：

1. 在官网控制台创建子账号，参考OSS STS
2. 在官网控制台创建STS角色并赋予子账号扮演角色的权限，参考OSS STS
3. 使用子账号的AccessKeyId/AccessKeySecret向STS申请临时token
4. 使用临时token中的认证信息创建OSS的Client
5. 使用OSS的Client访问OSS服务

使用STS凭证构造签名请求

用户的client端拿到STS临时凭证后，通过其中安全令牌(SecurityToken)以及临时访问密钥(AccessKeyId, AccessKeySecret)生成oss_request_options。以上传文件为例：

```
aos_pool_t *p;
oss_request_options_t *options;
```

```
aos_status_t *s;
aos_table_t *headers;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的object名字>";
aos_string_t bucket;
aos_string_t object;
char *data = "<object content>";
aos_list_t buffer;
aos_buf_t *content;

aos_pool_create(&p, NULL);

// init_oss_request_options using sts_token
/* 创建并用STS token初始化options */
options = oss_request_options_create(p);
options->config = oss_config_create(options->pool);
aos_str_set(&options->config->endpoint, "<您的Endpoint>");
aos_str_set(&options->config->access_key_id, "<您的临时AccessKeyId>");
aos_str_set(&options->config->access_key_secret, "<您的临时AccessKeySecret>");
aos_str_set(&options->config->sts_token, "<您的sts_token>");
options->config->is_cname = 0;
options->ctl = aos_http_controller_create(options->pool, 0);

/* 初始化参数 */
aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
headers = aos_table_make(p, 0);
aos_list_init(&buffer);
content = aos_buf_pack(options->pool, data, strlen(data));
aos_list_add_tail(&content->node, &buffer);

/* 上传文件 */
s = oss_put_object_from_buffer_s(options, &bucket, &object, &buffer, headers, &resp_headers);
if (aos_status_is_ok(s)) {
printf("put object succeeded\n");
} else {
printf("put object failed\n");
}

aos_pool_destroy(p);
```

URL签名授权

可以通过生成签名URL的形式提供给用户一个临时的访问URL。在生成URL时，可以指定URL过期的时间，从而限制用户长时间访问。

生成签名url

通过oss_gen_signed_url接口生成请求url签名。

生成下载请求的url签名

```
aos_pool_t *p;
oss_request_options_t *options;
aos_http_request_t *req;
char *url_str;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的object名字>";
aos_string_t bucket;
aos_string_t object;
apr_time_t now;
int64_t expire_time;
int one_hour = 3600; /* 单位：秒*/

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
req = aos_http_request_create(p);
req->method = HTTP_GET;
now = apr_time_now(); //millisecond
expire_time = now / 1000000 + one_hour;

/* 生成签名url */
url_str = oss_gen_signed_url(options, &bucket, &object, expire_time, req);
printf("临时下载url:%s\n", url_str);

aos_pool_destroy(p);
```

生成上传文件请求的url签名：

```
aos_pool_t *p;
oss_request_options_t *options;
aos_http_request_t *req;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的object名字>";
aos_string_t bucket;
aos_string_t object;
apr_time_t now;
int64_t expire_time;
int one_hour = 3600;
char *url_str = NULL;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
```

```
aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
req = aos_http_request_create(p);
req->method = HTTP_PUT;
now = apr_time_now(); //millisecond
expire_time = now / 1000000 + one_hour;

/* 生成签名url */
url_str = oss_gen_signed_url((options, &bucket, &object, expire_time, req);
printf("临时上传url:%s\n", url_str);

aos_pool_destroy(p);
```

使用签名URL下载文件

```
aos_pool_t *p;
oss_request_options_t *options;
aos_http_request_t *req;
aos_table_t *headers;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的object名字>";
char *filepath = "<本地文件路径>";
aos_string_t bucket;
aos_string_t object;
aos_string_t file;
char *url_str;
apr_time_t now;
int64_t expire_time;
int one_hour = 3600;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
aos_str_set(&file, filepath);
headers = aos_table_make(p, 0);
req = aos_http_request_create(p);
req->method = HTTP_GET;
now = apr_time_now(); /* 单位：微秒 */
expire_time = now / 1000000 + one_hour;

/* 生成签名url */
url_str = oss_gen_signed_url(options, &bucket, &object, expire_time, req);

/* 使用签名url下载文件 */
s = oss_get_object_to_file_by_url(options, url_str, headers, &file, &resp_headers);
if (aos_status_is_ok(s)) {
printf("get object succeeded\n");
```

```
} else {
printf("get object failed\n");
}
aos_pool_destroy(p);
```

使用URL签名的方式上传文件

```
aos_pool_t *p;
int is_oss_domain = 1;//是否使用三级域名
oss_request_options_t *options;
aos_http_request_t *req;
aos_table_t *headers;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
char *object_name = "<您的object名字>";
char *filepath = "<本地文件路径>";
aos_string_t bucket;
aos_string_t object;
aos_string_t file;
char *url_str;
apr_time_t now;
int64_t expire_time;
int one_hour = 3600;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 初始化参数 */
aos_str_set(&bucket, bucket_name);
aos_str_set(&object, object_name);
aos_str_set(&file, filepath);
headers = aos_table_make(p, 0);
req = aos_http_request_create(p);
req->method = HTTP_PUT;
now = apr_time_now(); /* 单位：微秒*/
expire_time = now / 1000000 + one_hour;

/* 生成签名url */
url_str = oss_gen_signed_url(options, &bucket, &object, expire_time, req);

/* 使用签名url上传文件 */
s = oss_put_object_from_file_by_url(options, url_str, &file, headers, &resp_headers);
if (aos_status_is_ok(s)) {
printf("put objects by signed url succeeded\n");
} else {
printf("put objects by signed url failed\n");
}

aos_pool_destroy(p);
```

生命周期管理

OSS允许用户对存储空间设置生命周期规则，以自动淘汰过期掉的文件，节省存储空间。 更多关于生命周期的内容请参考 文件生命周期

设置生命周期规则

通过oss_put_bucket_lifecycle接口，可以实现设置生命周期规则：

lifecycle的配置规则由一段xml表示。

```
<LifecycleConfiguration>
  <Rule>
    <ID>delete obsoleted files</ID>
    <Prefix>obsoleted/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>3</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

各字段解释：

- ID字段是用来唯一表示本条Rule（各个ID之间不能由包含关系，比如abc和abcd这样的）。
- Prefix指定对bucket下的符合特定前缀的文件使用规则。
- Status指定本条规则的状态，只有Enabled和Disabled，分别表示启用规则和禁用规则。
- Expiration节点里面的Days表示大于文件最后修改时间指定的天数就删除文件，Date则表示到指定的绝对时间之后就删除文件(绝对时间服从ISO8601的格式)。

可以通过下面的代码，设置上述lifecycle规则。

```
aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
aos_string_t bucket;
aos_list_t lifecycle_rule_list;
oss_lifecycle_rule_content_t *rule_content;
char *rule_name = "rule_name";

aos_pool_create(&p, NULL);
```

```
/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 创建生命周期规则并设置给存储空间 */
aos_str_set(&bucket, bucket_name);
aos_list_init(&lifecycle_rule_list);
rule_content = oss_create_lifecycle_rule_content(p);
aos_str_set(&rule_content->id, rule_name);
aos_str_set(&rule_content->prefix, "obsoleted");
aos_str_set(&rule_content->status, "Enabled");
rule_content->days = 3;
aos_list_add_tail(&rule_content->node, &lifecycle_rule_list);
s = oss_put_bucket_lifecycle(options, &bucket, &lifecycle_rule_list, &resp_headers);
if (aos_status_is_ok(s)) {
    printf("put bucket lifecycle succeeded\n");
} else {
    printf("put bucket lifecycle failed\n");
}

aos_pool_destroy(p);
```

查看生命周期规则

通过oss_get_bucket_lifecycle接口，可以实现查看生命周期规则：

```
aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
aos_string_t bucket;
aos_list_t lifecycle_rule_list;
oss_lifecycle_rule_content_t *rule_content;
char *rule_id;
char *prefix;
char *status;
int days = INT_MAX;
char* date = "";

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 获取存储空间生命周期规则并打印 */
aos_str_set(&bucket, bucket_name);
aos_list_init(&lifecycle_rule_list);
s = oss_get_bucket_lifecycle(options, &bucket, &lifecycle_rule_list, &resp_headers);
aos_list_for_each_entry(rule_content, &lifecycle_rule_list, node) {
    rule_id = apr_psprintf(p, "%.*s", rule_content->id.len, rule_content->id.data);
    prefix = apr_psprintf(p, "%.*s", rule_content->prefix.len, rule_content->prefix.data);
    status = apr_psprintf(p,"%.*s", rule_content->status.len, rule_content->status.data);
```

```
    date = apr_psprintf(p, "%.*s", rule_content->date.len, rule_content->date.data);
    days = rule_content->days;
}

aos_pool_destroy(p);
```

清空生命周期规则

通过oss_delete_bucket_lifecycle接口，实现清空生命周期规则：

```
aos_pool_t *p;
oss_request_options_t *options;
aos_status_t *s;
aos_table_t *resp_headers;
char *bucket_name = "<您的bucket名字>";
aos_string_t bucket;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_options(options);

/* 删除存储空间生命周期规则 */
aos_str_set(&bucket, bucket_name);
s = oss_delete_bucket_lifecycle(options, &bucket, &resp_headers);
if (aos_status_is_ok(s)) {
    printf("delete bucket lifecycle succeeded\n");
} else {
    printf("delete bucket lifecycle failed\n");
}

aos_pool_destroy(p);
```

图片处理

OSS图片处理，是OSS对外提供的海量、安全、低成本、高可靠的图片处理服务。用户将原始图片上传保存到OSS，通过简单的 RESTful 接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。图片处理提供图片处理接口，图片上传请使用上传接口。基于OSS图片处理，用户可以搭建自己的图片处理服务。

图片处理基础功能

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放、裁剪、旋转
- 图片效果
- 图片添加图片、文字、图文混合水印
- 自定义图片处理样式，在控制台的 图片处理 -> 样式管理 中定义
- 通过级联处理调用多个图片处理功能

图片处理使用

图片处理使用标准的 HTTP GET 请求来访问，所有的处理参数是编码在 URL 中的QueyString。

匿名访问

如果图片文件（Object）的访问权限是 公共读 ，如下表所示的权限，则可以匿名访问图片服务。

Bucket权限	Object权限
公共读私有写（public-read）或 公共读写（public-read-write）	默认（default）
任意权限	公共读私有写（public-read）或 公共读写（public-read-write）

通过如下格式的三级域名匿名访问图片处理：

```
http://bucket.<endpoint>/object?x-oss-process=image/action,parame_value
```

- bucket：用户的存储空间（bucket）名称
- endpoint：用户存储空间所在数据中心的访问域名
- object：用户上传在OSS上的图片文件
- image：图片处理保留标志符
- action：用户对图片做的操作，如缩放、裁剪、旋转等
- parame：用户对图片做的操作所对应的参数

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

自定义样式，使用如下格式的三级域名匿名访问图片处理：

```
http://bucket.<endpoint>/object?x-oss-process=x-oss-process=style/name
```

- style：用户自定义样式系统保留标志符
- name：自定义样式名称，即控制台定义样式的 规则名

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=style/oss-pic-style-w-100
```

通过级联处理，可以对一张图片顺序实施多个操作，格式如下：

```
http://bucket.<endpoint>/object?x-oss-process=image/action,parame_value/action,parame_value/...
```

例如：

```
http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100/rotate,90
```

图片服务也支持HTTPS访问，例如：

```
https://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
```

授权访问

对私有权限的文件（Object），如下表所示的权限，必须通过授权才能访问图片服务。

Bucket权限	Object权限
私有读写（private）	默认权限（default）
任意权限	私有读写（private）

生成带签名的图片处理的URL代码如下：

```
aos_pool_t *p;
oss_request_options_t *options;
aos_http_request_t *req;
char *url_str;
aos_table_t *params = NULL;
aos_string_t bucket;
aos_string_t object;
apr_time_t now;
int64_t expire_time;

aos_pool_create(&p, NULL);

/* 创建并初始化options */
options = oss_request_options_create(p);
init_sample_request_options(options, AOS_FALSE);

/* 初始化参数 */
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");
```

```
/* 图片处理 */
params = aos_table_make(p, 1);
apr_table_set(params, OSS_PROCESS, "image/resize,m_fixed,w_100,h_100");

req = aos_http_request_create(p);
req->method = HTTP_GET;
req->query_params = params;

/* 过期时间，单位秒 */
now = apr_time_now();
expire_time = now / 1000000 + 10 * 60;

/* 生成签名url */
url_str = oss_gen_signed_url(options, &bucket, &object, expire_time, req);
printf("url:%s\n", url_str);

aos_pool_destroy(p);
```

提示：

- 授权访问支持 自定义样式、 HTTPS、 级联处理
- oss_gen_signed_url 过期时间单位是**秒**

SDK访问

对于任意权限的图片文件，都可以直接使用 SDK 访问图片、进行处理。

提示：

- 图片处理的完整代码请参考：GitHub
- SDK处理图片文件支持 自定义样式、 HTTPS、 级联处理

基础操作

图片处理的基础操作包括，获取图片信息、格式转换、缩放、裁剪、旋转、效果、水印等。

```
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
oss_request_options_t *options = NULL;
aos_table_t *headers = NULL;
aos_table_t *params = NULL;
aos_table_t *resp_headers = NULL;
aos_status_t *s = NULL;
aos_string_t filename;
char *style = NULL;

aos_pool_create(&p, NULL);
options = oss_request_options_create(p);
```

```
init_sample_request_options(options, AOS_FALSE);
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");
aos_str_set(&filename, "<处理后图片保存路径>");

params = aos_table_make(p, 1);
style = "image/resize,m_fixed,w_100,h_100";
apr_table_set(params, OSS_PROCESS, "image/resize,m_fixed,w_100,h_100");

/* 下载处理后的图片到本地文件 */
s = oss_get_object_to_file(options, &bucket, &object, headers,
params, &filename, &resp_headers);
if (aos_status_is_ok(s)) {
printf("get object to file succeeded\n");
} else {
printf("get object to file failed\n");
}

aos_pool_destroy(p);
```

自定义样式

```
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
oss_request_options_t *options = NULL;
aos_table_t *headers = NULL;
aos_table_t *params = NULL;
aos_table_t *resp_headers = NULL;
aos_status_t *s = NULL;
aos_string_t filename;
char *style = NULL;

aos_pool_create(&p, NULL);
options = oss_request_options_create(p);
init_sample_request_options(options, AOS_FALSE);
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");
aos_str_set(&filename, "<处理后图片保存路径>");

/* 自定义样式 */
style = "style/oss-pic-style-w-100";
params = aos_table_make(p, 1);
apr_table_set(params, OSS_PROCESS, "image/resize,m_fixed,w_100,h_100");

/* 下载处理后的图片到本地文件 */
s = oss_get_object_to_file(options, &bucket, &object, headers,
params, &filename, &resp_headers);
if (aos_status_is_ok(s)) {
printf("get object to file succeeded\n");
} else {
printf("get object to file failed\n");
}
```

```
aos_pool_destroy(p);
```

级联处理

```
aos_pool_t *p = NULL;
aos_string_t bucket;
aos_string_t object;
oss_request_options_t *options = NULL;
aos_table_t *headers = NULL;
aos_table_t *params = NULL;
aos_table_t *resp_headers = NULL;
aos_status_t *s = NULL;
aos_string_t filename;
char *style = NULL;

aos_pool_create(&p, NULL);
options = oss_request_options_create(p);
init_sample_request_options(options, AOS_FALSE);
aos_str_set(&bucket, "<您的bucket名字>");
aos_str_set(&object, "<您的object名字>");
aos_str_set(&filename, "<处理后图片保存路径>");

/* 级联处理 */
style = "image/resize,m_fixed,w_100,h_100/rotate,90";
params = aos_table_make(p, 1);
apr_table_set(params, OSS_PROCESS, "image/resize,m_fixed,w_100,h_100");

/* 下载处理后的图片到本地文件 */
s = oss_get_object_to_file(options, &bucket, &object, headers,
params, &filename, &resp_headers);
if (aos_status_is_ok(s)) {
printf("get object to file succeeded\n");
} else {
printf("get object to file failed\n");
}

aos_pool_destroy(p);
```

图片处理工具

- 可视化图片处理工具 ImageStyleViever ，可以直观的看到OSS图片处理的结果
- OSS图片处理的功能、使用演示 页面

错误处理

异常处理

使用OSS C SDK时如果请求出错，会有相应的错误信息在aos_status_s中输出。aos_status_s有以下几个属性：

- code: 出错请求的HTTP状态码，整形类型。
- error_code: OSS的错误码，字符串类型。
- error_msg: OSS的错误信息，字符串类型。
- req_id: 标识该次请求的UUID；当您无法解决问题时，可以凭这个req_id来请求OSS开发工程师的帮助，字符串类型。

超时处理

如何判断超时

- 如果返回的aos_status_t中的code为5XX，表示服务器内部错误，可以重试。
- 如果返回的aos_status_t中的code不等于2XX，且error_code为-992或者-995时，表示链接超时或请求超时，可以重试。
- 可以使用aos_status.h中的aos_should_retry(aos_status_t *)判断返回的错误码是否需要重试，如果返回1，表示需要重试。

如何设置超时时间

- 设置链接超时：options->ctl->options->connect_timeout，单位秒，默认是10秒。
- 设置DNS超时：options->ctl->options->dns_cache_timeout，单位秒，默认是60秒。
- 设置请求超时：
    - 通过设置options->ctl->options->speed_limit的值控制能容忍的最小速率，默认是1024，即1K。
    - 通过设置options->ctl->options->speed_time的值控制能容忍的最长时间，默认是15秒。
    - 如果使用了上述默认值，表示如果连续15秒的传输速率小于1K，则超时。

常见错误码

错误码	描述
AccessDenied	拒绝访问
BucketAlreadyExists	Bucket已经存在
BucketNotEmpty	Bucket不为空
EntityTooLarge	实体过大
EntityTooSmall	实体过小
FileGroupTooLarge	文件组过大
FilePartNotExist	文件Part不存在

FilePartStale	文件Part过时
InvalidArgument	参数格式错误
InvalidAccessKeyId	AccessKeyId不存在
InvalidBucketName	无效的Bucket名字
InvalidDigest	无效的摘要
InvalidObjectName	无效的Object名字
InvalidPart	无效的Part
InvalidPartOrder	无效的part顺序
InvalidTargetBucketForLogging	Logging操作中有无效的目标bucket
InternalError	OSS内部发生错误
MalformedXML	XML格式非法
MethodNotAllowed	不支持的方法
MissingArgument	缺少参数
MissingContentLength	缺少内容长度
NoSuchBucket	Bucket不存在
NoSuchKey	文件不存在
NoSuchUpload	Multipart Upload ID不存在
NotImplemented	无法处理的方法
PreconditionFailed	预处理错误
RequestTimeTooSkewed	发起请求的时间和服务器时间超出15分钟
RequestTimeout	请求超时
SignatureDoesNotMatch	签名错误
TooManyBuckets	用户的Bucket数目超过限制

FAQ

OSS C SDK在嵌入式环境下如何编译

OSS C SDK如何设置程序日志的输出

OSS C SDK如何设置通信时和CURL相关的一些参数

OSS C SDK利用CURL提供的Callback实现上传

OSS C SDK利用CURL提供的Callback实现数据下载

OSS C SDK(windows版本)如何上传和下载包含中文名的文件

Ruby-SDK

安装

- github地址：https://github.com/aliyun/aliyun-oss-ruby-sdk
- API文档地址：http://www.rubydoc.info/gems/aliyun-sdk/
- ChangeLog：https://github.com/aliyun/aliyun-oss-ruby-sdk/blob/master/CHANGELOG.md

要求

- 开通阿里云OSS服务，并创建了AccessKeyId 和AccessKeySecret。
- 如果您还没有开通或者还不了解阿里云OSS服务，请登录 OSS产品主页了解。
- 如果还没有创建AccessKeyId和AccessKeySecret，请到 阿里云Access Key管理创 建Access Key。

安装

直接用gem安装：

```
gem install aliyun-sdk
```

如果无法访问https://rubygems.org，则可以使用淘宝的镜像源：

```
gem install aliyun-sdk --clear-sources --source https://ruby.taobao.org
```

或者通过bundler安装，首先在你的应用程序的Gemfile中添加：

```
gem 'aliyun-sdk', '~> 0.3.0'
```

然后运行：

```
```

```
# 使用淘宝的镜像源，可选
bundle config mirror.https://rubygems.org https://ruby.taobao.org

bundle install
```

https://ruby.taobao.org 是完整的rubygems.org的镜像，自动和官方源同步。 不方便访问rubygems.org的用户可以使用此源。

依赖

- Ruby版本 >= 1.9.3
- 支持Ruby运行环境的Windows/Linux/OS X系统

注意：

- SDK依赖的一些gem是本地扩展的形式，因此需要安装ruby-dev以支持编译本地 扩展的gem
- SDK依赖的处理XML的gem(nokogiri)要求环境中包含zlib库

Linux

Linux中以Ubuntu为例，安装上述依赖的方法：

```
sudo apt-get install ruby ruby-dev zlib1g-dev
```

各个Linux发行版都有自己的包管理工具，安装的方法类似。

Windows

1. 前往Ruby Installer下载RubyInstaller，双击安装，在 安装时选中"Add Ruby executables to your PATH"。**注意：请下载2.1及 以下版本。2.2版本因为存在问题无法顺利安装**
2. 前往Ruby Installer下载DEVELOPMENT KIT，注意选择相 应的版本。下载后是一个压缩包，在解压前首先在C盘根目录建立一个文件夹 C:\RubyDev，然后将压缩包解压到此文件夹。

   按Windows + R输入cmd后回车进入到命令行窗口，输入以下命令：

   ```
   cd C:\RubyDev
   ruby dk.rb init
   ruby dk.rb install
   ```

   如果最后一步install时显示“config.yml配置错误”，则用文本编辑器打开C:\RubyDev\config.yml，将其内容改为：

   ```
   ---
   ```

```
- C:/Ruby21
```

保存config.yml然后继续ruby dk.rb install。其中"Ruby21"是Ruby的安 装目录。根据具体的名字填写。完成后关闭此命令行窗口。

按Windows + R输入cmd后回车进入到命令行窗口，输入以下命令：

```
gem install aliyun-sdk
```

安装顺利完成后，输入irb进入Ruby交互式命令行，输入require 'aliyun/oss'，如果显示"true"则SDK已经顺利安装。

OS X

1. OS X系统默认已经安装了ruby，但是为了方便使用和管理，建议用户再安装 一个开发用的版本。
2. 在终端输入xcode-select --install安装"Xcode command line tools"。 如果安装失败，可选择手动下载安装（见下载的步骤）。
3. 从苹果开发者网站下载"Xcode command line tools"， 需要用您的Apple ID登录后才能下载。注意选择与您的系统匹配的版本。下 载完成后双击加载dmg文件，然后在打开的窗口中双击安装程序进行安装。在 安装的过程中需要输入您的系统密码。

安装brew，在终端输入以下命令：

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

安装ruby，在终端输入以下命令：

```
brew install ruby
exec $SHELL -l
```

安装SDK，在终端输入以下命令：

```
gem install aliyun-sdk
```

在终端输入以下命令验证SDK安装成功：

```
irb
> require 'aliyun/oss'
=> true
```

快速开始

下面介绍如何使用OSS Ruby SDK来访问OSS服务，包括查看Bucket列表，查看文件列表，上传/下载文件和删除文件。为了方便使用，下面的操作都是在Ruby的交互式命令行irb中进行。

初始化Client

在命令行中输入并回车：

```
irb
```

进入到Ruby的交互式命令行模式。接着通过require引入SDK的包：

```
> require 'aliyun/oss'
=> true
```

注：在接下来的演示中，>符号后面的内容是用户输入的命令，=>后面的内容是程序返回的内容。

接下来创建Client：

```
> client = Aliyun::OSS::Client.new(
> endpoint: 'endpoint',
> access_key_id: 'AccessKeyId',
> access_key_secret: 'AccessKeySecret')
=> #<Aliyun::OSS::Client...
```

将其中的参数替换成您实际的endpoint，AccessKeyId和AccessKeySecret。

查看Bucket列表

通过以下命令查看Bucket列表：

```
> buckets = client.list_buckets
=> #<Enumerator...
> buckets.each { |b| puts b.name }
=> bucket-1
=> bucket-2
=> ...
```

如果Bucket列表为空，则可以用以下命令创建一个Bucket：

```
> client.create_bucket('my-bucket')
=> true
```

注：

1. Bucket的命名规范请查看OSS 基本概念。
2. Bucket名字不能与OSS服务中其他用户已有的Bucket重复，所以你需要选择一个独特的Bucket名字以避免创建失败。

查看文件列表

通过以下命令查看Bucket中的文件列表：

```
> bucket = client.get_bucket('my-bucket')
=> #<Aliyun::OSS::Bucket...
> objects = bucket.list_objects
=> #<Enumerator...
> objects.each { |obj| puts obj.key }
=> object-1
=> object-2
=> ...
```

上传一个文件

通过以下命令向Bucket中上传一个文件：

```
> bucket.put_object('my-object', :file => 'local-file')
=> true
```

其中local-file是需要上传的本地文件的路径。上传成功后，可以通过list_objects来查看：

```
> objects = bucket.list_objects
=> #<Enumerator...
> objects.each { |obj| puts obj.key }
=> my-object
=> ...
```

下载一个文件

通过以下命令从Bucket中下载一个文件：

```
> bucket.get_object('my-object', :file => 'local-file')
=> #<Aliyun::OSS::Object...
```

其中local-file是文件保存的路径。下载成功后，可以打开文件查看其内容。

删除一个文件

通过以下命令从Bucket中删除一个文件：

```
> bucket.delete_object('my-object')
=> true
```

删除文件后可以通过list_objects来查看文件确实已经被删除：

```
> objects = bucket.list_objects
=> #<Enumerator...
> objects.each { |obj| puts obj.key }
=> object-1
=> ...
```

了解更多

- 管理Bucket
- 上传文件
- 下载文件
- 管理文件
- Rails应用
- 自定义域名绑定
- 使用STS访问
- 设置访问权限
- 管理生命周期
- 设置访问日志
- 静态网站托管
- 设置防盗链
- 设置跨域资源共享
- 异常

管理存储空间

管理存储空间

存储空间（Bucket）是OSS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

查看所有Bucket

使用Client#list_buckets接口列出当前用户下的所有Bucket，用户还可以指定:prefix参数，列出Bucket名字为特定前缀的所有Bucket：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

buckets = client.list_buckets
buckets.each { |b| puts b.name }

buckets = client.list_buckets(:prefix => 'my-')
buckets.each { |b| puts b.name }
```

创建Bucket

使用Client#create_bucket接口创建一个Bucket，用户需要指定Bucket的名字：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

client.create_bucket('my-bucket')
```

注意：

- Bucket的命名规范请查看OSS 基本概念。
- 由于存储空间的名字是全局唯一的，所以必须保证您的Bucket名字不与别人的重复。

删除Bucket

使用Client#delete_bucket接口删除一个Bucket，用户需要指定Bucket的名字：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

client.delete_bucket('my-bucket')
```

注意：

- 如果该Bucket下还有文件存在，则需要先删除所有文件才能删除Bucket
- 如果该Bucket下还有未完成的上传请求，则需要通过list_uploads和abort_upload先取消那些请求才能删除Bucket。用法请参考API文档。

查看Bucket是否存在

用户可以通过Client#bucket_exists?接口查看当前用户的某个Bucket是否存在：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

puts client.bucket_exists?('my-bucket')
```

Bucket访问权限

用户可以设置Bucket的访问权限，允许或者禁止匿名用户对其内容进行读写。更多关于访问权限的内容请参考访问权限。

获取Bucket的访问权限（ACL）

通过Bucket#acl查看Bucket的ACL：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
puts bucket.acl
```

设置Bucket的访问权限（ACL）

通过Bucket#acl=设置Bucket的ACL：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.acl = Aliyun::OSS::ACL::PUBLIC_READ
puts bucket.acl
```

上传文件

上传文件（Object）

OSS Ruby SDK提供了丰富的文件上传接口，用户可以通过以下方式向OSS中上传文件：

- 上传本地文件到OSS
- 流式上传
- 断点续传上传
- 追加上传
- 上传回调

上传本地文件

通过Bucket#put_object接口，并指定:file参数来上传一个本地文件到OSS：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.put_object('my-object', :file => 'local-file')
```

流式上传

在进行大文件上传时，往往不希望一次性处理全部的内容然后上传，而是希望流式地处理，一次上传一部分内容。甚至如果要上传的内容本身就来自网络，不能一次获取，那只能流式地上传。通过Bucket#put_object接口，并指定block参数来将流式生成的内容上传到OSS：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.put_object('my-object') do |stream|
100.times { |i| stream << i.to_s }
end
```

断点续传上传

当上传大文件时，如果网络不稳定或者程序崩溃了，则整个上传就失败了。用户不得不重头再来，这样做不仅浪费资源，在网络不稳定的情况下，往往重试多次还是无法完成上传。通过Bucket#resumable_upload接口来实现断点续传上传。它有以下参数：

- key 上传到OSS的Object名字
- file 待上传的本地文件路径
- opts 可选项，主要包括：
    - :cpt_file 指定checkpoint文件的路径，如果不指定则默认为与本地文件同目录下的file.cpt，其中file是本地文件的名字
    - :disable_cpt 如果指定为true，则上传过程中不会记录上传进度，失败后也无法进行续传
    - :part_size 指定每个分片的大小，默认为4MB
    - &block 如果调用时候传递了block，则上传进度会交由block处理

详细的参数请参考API文档。

其实现的原理是将要上传的文件分成若干个分片分别上传，最后所有分片都上传成功后，完成整个文件的上传。在上传的过程中会记录当前上传的进度信息（记录在checkpoint文件中），如果上传过程中某一分片上传失败，再次上传时会从checkpoint文件中记录的点继续上传。这要求再次调用时要指定与上次相同的checkpoint文件。上传完成后，checkpoint文件会被删除。

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.resumable_upload('my-object', 'local-file') do |p|
```

```
puts "Progress: #{p}"
end

bucket.resumable_upload(
'my-object', 'local-file',
:part_size => 100 * 1024, :cpt_file => '/tmp/x.cpt') { |p|
puts "Progress: #{p}"
}
```

注意：

- SDK会将上传的中间状态信息记录在cpt文件中，所以要确保用户对cpt文件有写权限。
- cpt文件记录了上传的中间状态信息并自带了校验，用户不能去编辑它，如果cpt文件损坏则上传无法继续。整个上传完成后cpt文件会被删除。
- 如果上传过程中本地文件发生了改变，则上传会失败。

追加上传

OSS支持可追加的文件类型，通过Bucket#append_object来上传可追加的文件，调用时需要指定文件追加的位置，对于新创建文件，这个位置是0；对于已经存在的文件，这个位置必须是追加前文件的长度。

- 文件不存在时，调用append_object会创建一个可追加的文件
- 文件存在时，调用append_object会向文件末尾追加内容

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
# 创建可追加的文件
bucket.append_object('my-object', 0) {}

# 向文件末尾追加内容
next_pos = bucket.append_object('my-object', 0) do |stream|
100.times { |i| stream << i.to_s }
end
next_pos = bucket.append_object('my-object', next_pos, :file => 'local-file-1')
next_pos = bucket.append_object('my-object', next_pos, :file => 'local-file-2')
```

注意：

- 只能向可追加的文件（即通过 append_object 创建的文件）追加内容。
- 可追加的文件不能被拷贝。

上传回调

用户在上传文件时可以指定“上传回调”，这样在文件上传成功后OSS会向用户提供的服务器地址发起一个HTTP POST请求，相当于一个通知机制。用户可以在收到回调的时候做相应的动作。更多有关上传回调的内容请参考OSS 上传回调。

目前OSS支持上传回调的接口只有put_object和resumable_upload。

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')

callback = Aliyun::OSS::Callback.new(
url: 'http://10.101.168.94:1234/callback',
query: {user: 'put_object'},
body: 'bucket=${bucket}&object=${object}'
)

begin
bucket.put_object('files/hello', file: '/tmp/x', callback: callback)
rescue Aliyun::OSS::CallbackError => e
puts "Callback failed: #{e.message}"
end
```

上面的例子使用put_object上传了一个文件，并指定了上传回调并将此次上传的bucket和object信息添加在body中，应用服务器收到这个回调后，就知道这个文件已经成功上传到OSS了。

resumable_upload的使用方法类似：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')

callback = Aliyun::OSS::Callback.new(
url: 'http://10.101.168.94:1234/callback',
query: {user: 'put_object'},
body: 'bucket=${bucket}&object=${object}'
)

begin
bucket.resumable_upload('files/hello', '/tmp/x', callback: callback)
rescue Aliyun::OSS::CallbackError => e
puts "Callback failed: #{e.message}"
end
```

注意：

- callback的URL**不能**包含query string，而应该在:query参数中指定。
- 可能出现文件上传成功，但是执行回调失败的情况，此时client会抛出 CallbackError，用户如果要忽略此错误，需要显式接住这个异常。
- 详细的例子请参考callback.rb。
- 接受回调的server请参考callback_server.rb。

下载文件

OSS Ruby SDK提供了丰富的文件下载接口，用户可以通过以下方式从OSS中下载 文件：

- 下载到本地文件
- 流式下载
- 断点续传下载
- HTTP下载（浏览器下载）

下载到本地文件

通过Bucket#get_object接口，并指定:file参数来下载到一个本地文件到：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.get_object('my-object', :file => 'local-file')
```

流式下载

在进行大文件下载时，往往不希望一次性处理全部的内容，而是希望流 式地处理，一次处理一部分内容。 通过Bucket#get_object接口，并指定block参数来流式处理下载的内容：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
```

```
bucket = client.get_bucket('my-bucket')
bucket.get_object('my-object') do |chunk|
  # handle_data(chunk)
  puts "Got a chunk, size: #{chunk.size}."
end
```

断点续传下载

当下载大文件时，如果网络不稳定或者程序崩溃了，则整个下载就失败了。用户 不得不重头再来，这样做不仅浪费资源，在网络不稳定的情况下，往往重试多次 还是无法完成下载。 通过Bucket#resumable_download接口来实现断点续传下载。它有以下参数：

- key 要下载的Object名字
- file 下载到本地文件的路径
- opts 可选项，主要包括：
  - :cpt_file 指定checkpoint文件的路径，如果不指定则默认为与本地文件同 目录下的file.cpt，其中file是本地文件的名字
  - :disable_cpt 如果指定为true，则下载过程中不会记录下载进度，失败后 也无法进行续传
  - :part_size 指定每个分片的大小，默认为10MB
  - &block 如果调用时候传递了block，则下载进度会交由block处理

详细的参数请参考API文档。

其实现的原理是将要下载的Object分成若干个分片分别下载，最后所有分片都下 载成功后，完成整个文件的下载。在下载的过程中会记录当前下载的进度信息 （记录在checkpoint文件中）和已下载的分片（保存为file.part.N，其中 file是下载的本地文件的名字），如果下载过程中某一分片下载失败，再次下 载时会从checkpoint文件中记录的点继续下载。这要求再次调用时要指定与上次 相同的checkpoint文件。下载完成后，part文件和checkpoint文件都会被删除。

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.resumable_download('my-object', 'local-file') do |p|
  puts "Progress: #{p}"
end

bucket.resumable_download(
  'my-object', 'local-file',
  :part_size => 100 * 1024, :cpt_file => '/tmp/x.cpt') { |p|
  puts "Progress: #{p}"
}
```

注意：

- SDK会将下载的中间状态信息记录在cpt文件中，所以要确保用户对cpt文 件有写权限
- SDK会将已下载的分片保存在part文件中，所以要确保用户对file所在的目 录有创建文件的权限
- cpt文件记录了下载的中间状态信息并自带了校验，用户不能去编辑它，如 果cpt文件损坏则下载无法继续
- 如果下载过程中待下载的Object发生了改变（ETag改变），或者part文件丢 失或被修改，则下载会报错

HTTP下载

对于存放在OSS中的文件，在不用SDK的情况下用户也可以直接使用HTTP下载，这 包括直接使用浏览器下载，或者使用wget, curl等命令行工具下载。这时文 件的URL需要由SDK生成。使用Bucket#object_url方法生成可下载的HTTP地址， 它接受以下参数：

- key 待下载的Object的名字
- sign 是否生成带签名的URL，对于拥有public-read/public-read-write权限 的Object，不带签名的URL也可以访问；对于private权限的Object，则必须使 用带签名的URL才能访问
- expiry URL的有效时间，默认为60s

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
# 生成URL，默认带签名，有效时间为60秒
puts bucket.object_url('my-object')
# http://my-bucket.oss-cn-hangzhou.aliyuncs.com/my-
object?Expires=1448349966&OSSAccessKeyId=5viOHfldyK6K72ht&Signature=aM2HpBLeMq1aec6JCd7BBAKYiwI%
3D

# 不带签名的URL
puts bucket.object_url('my-object', false)
# http://my-bucket.oss-cn-hangzhou.aliyuncs.com/my-object

# 指定URL过期时间为1小时（3600秒）
puts bucket.object_url('my-object', true, 3600)
```

管理文件

一个Bucket下可能有非常多的文件，SDK提供一系列的接口方便用户管理文件。

查看所有文件

通过Bucket#list_objects来列出当前Bucket下的所有文件。主要的参数如下：

- :prefix 指定只列出符合特定前缀的文件
- :marker 指定只列出文件名大于marker之后的文件
- :delimiter 用于获取文件的公共前缀

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
# 列出所有文件
objects = bucket.list_objects
objects.each { |o| puts o.key }

# 列出前缀为'my-'的所有文件
objects = bucket.list_objects(:prefix => 'my-')
objects.each { |o| puts o.key }

# 列出前缀为'my-'且在'my-object'之后的所有文件
objects = bucket.list_objects(:prefix => 'my-', :marker => 'my-object')
objects.each { |o| puts o.key }
```

模拟目录结构

OSS是基于对象的存储服务，没有目录的概念。存储在一个Bucket中所有文件都 是通过文件的key唯一标识，并没有层级的结构。这种结构可以让OSS的存储非常 高效，但是用户管理文件时希望能够像传统的文件系统一样把文件分门别类放到 不同的“目录”下面。通过OSS提供的“公共前缀”的功能，也可以很方便地模拟目录 结构。公共前缀的概念请参考 查看对象列表

假设Bucket中已有如下文件：

```
foo/x
foo/y
foo/bar/a
foo/bar/b
foo/hello/C/1
foo/hello/C/2
...
foo/hello/C/9999
```

接下来我们实现一个函数叫list_dir，列出指定目录下的文件和子目录：

```
require 'aliyun/oss'
```

```
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')

def list_dir(dir)
  objects = bucket.list_objects(:prefix => dir, :delimiter => '/')
  objects.each do |obj|
    if obj.is_a?(OSS::Object) # object
      puts "Object: #{obj.key}"
    else # common prefix
      puts "SubDir: #{obj}"
    end
  end
end
```

运行结果如下：

```
> list_dir('foo/')
=> SubDir: foo/bar/
   SubDir: foo/hello/
   Object: foo/x
   Object: foo/y

> list_dir('foo/bar/')
=> Object: foo/bar/a
   Object: foo/bar/b

> list_dir('foo/hello/C/')
=> Object: foo/hello/C/1
   Object: foo/hello/C/2
   ...
   Object: foo/hello/C/9999
```

文件元信息

向OSS上传文件时，除了文件内容，还可以指定文件的一些属性信息，称为“元信息”。 这些信息在上传时与文件一起存储，在下载时与文件一起返回。

在SDK中文件元信息用一个Hash表示，其他key和value都是String类型，并 且都**只能是简单的ASCII可见字符，不能包含换行。** 所有元信息的总大小不 能超过8KB。

注意：

- 因为文件元信息在上传/下载时是附在HTTP Headers中， HTTP协议规定不能 包含复杂字符。

使用Bucket#put_object，Bucket#append_object和 Bucket#resumable_upload时都可以通过指定:metas参数来指定文件的元信 息：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')

bucket.put_object(
  'my-object-1',
  :file => 'local-file',
  :metas => {'year' => '2016', 'people' => 'mary'})

bucket.append_object(
  'my-object-2', 0,
  :file => 'local-file',
  :metas => {'year' => '2016', 'people' => 'mary'})

bucket.resumable_upload(
  'my-object',
  'local-file',
  :metas => {'year' => '2016', 'people' => 'mary'})
```

通过Bucket#update_object_metas命令来更新文件元信息：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')

bucket.update_object_metas('my-object', {'year' => '2017'})
```

拷贝文件

使用Bucket#copy_object拷贝一个文件。拷贝时对文件元信息的处理有两种选 择，通过:meta_directive参数指定：

- 与源文件相同，即拷贝源文件的元信息
- 使用新的元信息覆盖源文件的信息

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
```

```
# 拷贝文件元信息
bucket.copy_object(
  'my-object', 'copy-object',
  :meta_directive => Aliyun::OSS::MetaDirective::COPY)

# 覆盖文件元信息
bucket.copy_object(
  'my-object', 'copy-object',
  :metas => {'year' => '2017'},
  :meta_directive => Aliyun::OSS::MetaDirective::REPLACE)
```

删除文件

通过Bucket#delete_object来删除某个文件：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')

bucket.delete_object('my-object')
```

批量删除文件

通过Bucket#batch_delete_object来删除一批文件，用户可以通过:quiet参 数来指定是否返回删除的结果：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')

objs = ['my-object-1', 'my-object-2']
result = bucket.batch_delete_object(objs)
puts result #['my-object-1', 'my-object-2']，默认返回删除成功的文件

objs = ['my-object-3', 'my-object-4']
result = bucket.batch_delete_object(objs, :quiet => true)
puts result #[]，不返回删除的结果
```

自定义域名绑定

OSS支持用户将自定义的域名绑定到OSS服务上，这样能够支持用户无缝地将存储 迁移到OSS上。例如用户的域名是my-domain.com，之前用户的所有图片资源都是 形如http://img.my-domain.com/x.jpg的格式，用户将图片存储迁移到OSS之 后，通过绑定自定义域名，仍可以使用原来的地址访问到图片：

- 开通OSS服务并创建Bucket
- 将img.my-domain.com与创建的Bucket绑定
- 将图片上传到OSS的这个Bucket中
- 修改域名的DNS配置，增加一个CNAME记录，将img.my-domain.com指向OSS服务 的endpoint（如my-bucket.oss-cn-hangzhou.aliyuncs.com）

这样就可以通过原地址http://img.my-domain.com/x.jpg访问到存储在OSS上 的图片。 绑定自定义域名请参考自定义域名绑定

在使用SDK时，也可以使用自定义域名作为endpoint，这时需要将:cname参数 设置为true，如下面的例子：

```
require 'aliyun/oss'

include Aliyun::OSS

client = Client.new(
  endpoint: 'ENDPOINT',
  access_key_id: 'ACCESS_KEY_ID',
  access_key_secret: 'ACCESS_KEY_SECRET',
  cname: true)

bucket = client.get_bucket('my-bucket')
```

注意：

- 使用CNAME时，无法使用list_buckets接口。（因为自定义域名已经绑定到 某个特定的Bucket）

使用STS访问

OSS可以通过阿里云STS服务，临时进行授权访问。更多有关STS的内容请参考：阿里云STS。

使用STS时请按以下步骤进行：

1. 在官网控制台创建子账号，参考OSS STS。
2. 在官网控制台创建STS角色并赋予子账号扮演角色的权限，参考OSS STS。
3. 使用子账号的AccessKeyId/AccessKeySecret向STS申请临时token。
4. 使用临时token中的认证信息创建OSS的Client。
5. 使用OSS的Client访问OSS服务。

在使用STS访问OSS时，需要设置:sts_token参数，如下面的例子所示：

```
require 'aliyun/sts'
require 'aliyun/oss'

sts = Aliyun::STS::Client.new(
access_key_id: '<子账号的AccessKeyId>',
access_key_secret: '<子账号的AccessKeySecret>')

token = sts.assume_role('<role-arn>', '<session-name>')

client = Aliyun::OSS::Client.new(
endpoint: '<endpoint>',
access_key_id: token.access_key_id,
access_key_secret: token.access_key_secret,
sts_token: token.security_token)

bucket = client.get_bucket('my-bucket')
```

在向STS申请临时token时，还可以指定自定义的STS Policy。这样申请的临时权限是**所扮演角色的权限与Policy指定的权限的交集。**下面的例子将通过指定STS Policy申请对my-bucket的只读权限，并指定临时token的过期时间为15分钟：

```
require 'aliyun/sts'
require 'aliyun/oss'

sts = Aliyun::STS::Client.new(
access_key_id: '<子账号的AccessKeyId>',
access_key_secret: '<子账号的AccessKeySecret>')

policy = Aliyun::STS::Policy.new
policy.allow(['oss:Get*'], ['acs:oss:*:*:my-bucket/*'])

token = sts.assume_role('<role arc>', '<session name>', policy, 15 * 60)

client = Aliyun::OSS::Client.new(
endpoint: 'ENDPOINT',
access_key_id: token.access_key_id,
access_key_secret: token.access_key_secret,
sts_token: token.security_token)

bucket = client.get_bucket('my-bucket')
```

更详细的用法和参数说明请参考API文档。

设置访问权限

设置访问权限（ACL）

OSS允许用户对Bucket和Object分别设置访问权限，方便用户控制自己的资源可以被如何访问。对于Bucket，有三种访问权限：

- Public-read-write 允许匿名用户向该Bucket中创建/获取/删除Object。
- Public-read 允许匿名用户获取该Bucket中的Object。
- Private 不允许匿名访问，所有的访问都要经过签名。

创建Bucket时，默认是private权限。之后用户可以通过bucket.acl=来设置Bucket的权限。

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
puts bucket.acl
```

对于Object，有四种访问权限：

- Default：继承所属的Bucket的访问权限，即与所属Bucket的权限值一样。
- Public-read-write：允许匿名用户读写该Object。
- Public-read 允许匿名用户读该Object。
- Private 不允许匿名访问，所有的访问都要经过签名。

创建Object时，默认为default权限。之后用户可以通过bucket.set_object_acl来设置Object的权限。

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')

acl = bucket.get_object_acl('my-object')
puts acl # default
bucket.set_object_acl('my-object', Aliyun::OSS::ACL::PUBLIC_READ)
acl = bucket.get_object_acl('my-object')
```

```
puts acl # public-read
```

需要注意的是：

- 如果设置了Object的权限（非default），则访问该Object时进行权限认证时 会优先判断Object的权限，而Bucket的权限设置会被忽略。

  允许匿名访问时（设置了public-read或者public-read-write权限），用户可以直接通过浏览器访问，例如：

  ```
  http://bucket-name.oss-cn-hangzhou.aliyuncs.com/object.jpg
  ```

  访问具有public权限的Bucket/Object时，也可以通过创建匿名的Client来进行：

  ```
  require 'aliyun/oss'

  # 不填access_key_id和access_key_secret，将创建匿名Client，只能访问具有
  # public权限的Bucket/Object
  client = Aliyun::OSS::Client.new(endpoint: 'endpoint')
  bucket = client.get_bucket('my-bucket')

  bucket.get_object('my-object', :file => 'local_file')
  ```

  更多关于访问权限控制的内容请参考 **访问控制**。

管理生命周期

OSS允许用户对Bucket设置生命周期规则，以自动淘汰过期掉的文件，节省存储空间。用户可以同时设置多条规则，一条规则包含：

- 规则ID，用于标识一条规则，不能重复
- 受影响的文件前缀，此规则只作用于符合前缀的文件
- 过期时间，有两种指定方式：
  - 指定距文件最后修改时间N天过期
  - 指定在具体的某一天过期，即在那天之后符合前缀的文件将会过期，**而不论文件的最后修改时间**。不推荐使用。
- 是否生效

更多关于生命周期的内容请参考**文件生命周期**。

设置生命周期规则

通过Bucket#lifecycle=来设置生命周期规则：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket.lifecycle = [
LifeCycleRule.new(
:id => 'rule1', :enabled => true, :prefix => 'foo/', :expiry => 3),
LifeCycleRule.new(
:id => 'rule2', :enabled => false, :prefix => 'bar/', :expiry => Date.new(2016, 1, 1))
]
```

查看生命周期规则

通过Bucket#lifecycle来查看生命周期规则：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

rules = bucket.lifecycle
puts rules
```

清空生命周期规则

通过Bucket#lifecycle=设置一个空的Rule数组来清空生命周期规则：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket.lifecycle = []
```

设置访问日志

设置访问日志（Logging）

OSS允许用户对Bucket设置访问日志记录，设置之后对于Bucket的访问会被记录成日志，日志存储在OSS上由用户指定的Bucket中，文件的格式为：

```
<TargetPrefix><SourceBucket>-YYYY-mm-DD-HH-MM-SS-UniqueString
```

其中TargetPrefix由用户指定。日志规则由以下3项组成：

- enable，是否开启
- target_bucket，存放日志文件的Bucket
- target_prefix，日志文件的前缀

更多关于访问日志的内容请参考Bucket访问日志。

开启Bucket日志

通过Bucket#logging=来开启日志功能：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')

bucket.logging = BucketLogging.new(
enable: true, target_bucket: 'logging_bucket', target_prefix: 'my-log')
```

查看Bucket日志设置

通过Bucket#logging来查看日志设置：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
log = bucket.logging
puts log.to_s
```

关闭Bucket日志

通过Bucket#logging=来关闭日志功能：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.logging = BucketLogging.new(enable: false)
```

静态网站托管

在自定义域名绑定中提到，OSS允许用户将自己的域名指向OSS服务的地址。这样用户访问他的网站的时候，实际上是在访问OSS的Bucket。对于网站，需要指定首页（index）和出错页（error）分别对应的Bucket中的文件名。

更多关于静态网站托管的内容请参考 OSS静态网站托管。

设置托管页面

通过Bucket#website=来设置托管页面：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.website = BucketWebsite.new(index: 'index.html', error: 'error.html')
```

查看托管页面

通过Bucket#website来查看托管页面：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
```

```
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
web = bucket.website
puts web.to_s
```

清除托管页面

通过Bucket#website=来清除托管页面：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.website = BucketWebsite.new(enable: false)
```

设置防盗链

OSS是按使用收费的服务，为了防止用户在OSS上的数据被其他人盗链，OSS支持 基于HTTP header中表头字段referer的防盗链方法。更多OSS防盗链请参考： OSS防盗链

设置Referer白名单

通过Bucket#referer=设置Referer白名单：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.referer = BucketReferer.new(
  allow_empty: true, whitelist: ['my-domain.com', '*.example.com'])
```

查看Referer白名单

通过Bucket#referer设置Referer白名单：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
ref = bucket.referer
puts ref.to_s
```

清空Referer白名单

通过Bucket#referer=设置清空Referer白名单：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.referer = BucketReferer.new(allow_empty: true, whitelist: [])
```

设置跨域资源共享

设置跨域资源共享（CORS）

跨域资源共享（CORS）允许web端的应用程序访问不属于本域的资源。OSS提供接口方便开发者控制跨域访问的权限。更多关于跨域资源共享的内容请参考OSS跨域资源共享。

OSS的跨域共享设置由一条或多条CORS规则组成，每条CORS规则包含以下设置：

- allowed_origins，允许的跨域请求的来源，如www.my-domain.com, *
- allowed_methods，允许的跨域请求的HTTP方法(PUT/POST/GET/DELETE/HEAD)
- allowed_headers，在OPTIONS预取指令中允许的header，如x-oss-test, *
- expose_headers，允许用户从应用程序中访问的响应头
- max_age_seconds, 浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间

设置CORS规则

通过Bucket#cors=设置CORS规则：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.cors = [
CORSRule.new(
:allowed_origins => ['aliyun.com', 'http://www.taobao.com'],
:allowed_methods => ['PUT', 'POST', 'GET'],
:allowed_headers => ['Authorization'],
:expose_headers => ['x-oss-test'],
:max_age_seconds => 100)
]
```

查看CORS规则

通过Bucket#cors查看CORS规则：

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
cors = bucket.cors
puts cors.map(&:to_s)
```

清空CORS规则

通过Bucket#cors=清空CORS规则

```
require 'aliyun/oss'

client = Aliyun::OSS::Client.new(
endpoint: 'endpoint',
access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')

bucket = client.get_bucket('my-bucket')
bucket.cors = []
```

异常

使用SDK时如果请求出错，会有相应的异常抛出，同时在log（默认为程序运行目 录下oss_sdk.log）中也会记录详细的出错信息。

OSS Ruby SDK中有两种异常：ClientError和ServerError，它们都是 RuntimeError的子类。

ClientError

ClientError指SDK内部出现的异常，比如参数设置错误或者断点上传/下载中出 现的文件被修改的错误。

ServerError

ServerError指服务器端错误，它来自于对服务器错误信息的解析。ServerError 有以下几个属性：

- http_code: 出错请求的HTTP状态码
- error_code: OSS的错误码
- message: OSS的错误信息
- request_id: 标识该次请求的UUID；当您无法解决问题时，可以凭这个RequestId来请求OSS开发工程师的帮助

OSS中常见的错误信息请参考 OSS错误响应

Rails应用

与Rails集成

在Rails应用中使用OSS Ruby SDK只需要在Gemfile中添加以下依赖：

```
gem 'aliyun-sdk', '~> 0.3.0
```

然后在使用OSS时引入依赖就可以了：

```
```

```
require 'aliyun/oss'
```

另外，SDK的rails/目录下提供一些方便用户使用的辅助代码。

下面我们将利用SDK来实现一个简单的OSS文件管理器（oss-manager），最终包含以下功能：

- 列出用户所有的Bucket
- 列出Bucket下所有的文件，按目录层级列出
- 上传文件
- 下载文件

1. 创建项目

先安装Rails，然后创建一个Rails应用，oss-manager：

```
gem install rails
rails new oss-manager
```

作为一个好的习惯，使用git管理项目代码：

```
cd oss-manager
git init
git add .
git commit -m "init project"
```

2. 添加SDK依赖

编辑oss-manager/Gemfile，向其中加入SDK的依赖：

```
gem 'aliyun-sdk', '~> 0.3.0'
```

然后在oss-manager/下执行：

```
bundle install
```

保存这一步所做的更改：

```
git add .
git commit -m "add aliyun-sdk dependency"
```

3. 初始化OSS Client

为了避免在项目中用到OSS Client的地方都要初始化，我们在项目中添加一个初始化文件，方便在项目中使用OSS Client：

```
# oss-manager/config/initializers/aliyun_oss_init.rb
require 'aliyun/oss'

module OSS
def self.client
unless @client
Aliyun::Common::Logging.set_log_file('./log/oss_sdk.log')

@client = Aliyun::OSS::Client.new(
endpoint:
Rails.application.secrets.aliyun_oss['endpoint'],
access_key_id:
Rails.application.secrets.aliyun_oss['access_key_id'],
access_key_secret:
Rails.application.secrets.aliyun_oss['access_key_secret']
)
end

@client
end
end
```

上面的代码在SDK的rails/目录下可以找到。这样初始化后，在项目中使用OSSClient就非常方便：

```
buckets = OSS.client.list_buckets
```

其中endpoint和AccessKeyId/AccessKeySecret保存在oss-manager/conf/secrets.yml中，例如：

```
development:
secret_key_base: xxxx
aliyun_oss:
endpoint: xxxx
access_key_id: aaaa
access_key_secret: bbbb
```

保存代码：

```
git add .
git commit -m "add aliyun-sdk initializer"
```

4. 实现List buckets功能

首先用rails生成管理Buckets的controller：

```
rails g controller buckets index
```

这样会在oss-manager中生成以下文件：

- app/controller/buckets_controller.rb Buckets相关的逻辑代码
- app/views/buckets/index.html.erb Buckets相关的展示代码
- app/helpers/buckets_helper.rb 一些辅助函数

首先编辑buckets_controller.rb，调用OSS Client，将list_buckets的结果存放在@buckets变量中：

```
class BucketsController < ApplicationController
def index
@buckets = OSS.client.list_buckets
end
end
```

然后编辑views/buckets/index.html.erb，将Bucket列表展示出来：

```
<h1>Buckets</h1>
<table class="table table-striped">
<tr>
<th>Name</th>
<th>Location</th>
<th>CreationTime</th>
</tr>

<% @buckets.each do |bucket| %>
<tr>
<td><%= link_to bucket.name, bucket_objects_path(bucket.name) %></td>
<td><%= bucket.location %></td>
<td><%= bucket.creation_time.localtime.to_s %></td>
</tr>
<% end %>
</table>
```

其中bucket_objects_path是一个辅助函数，在app/helpers/buckets_helper.rb中：

```
module BucketsHelper
def bucket_objects_path(bucket_name)
"/buckets/#{bucket_name}/objects"
end
end
```

这样就完成了列出所有Bucket的功能。在运行之前，我们还需要配置Rails的路由，使得我们在浏览器中输入的地址能够调用正确的逻辑。编辑config/routes.rb，增加一条：

```
resources :buckets do
resources :objects
end
```

好了，在oss-manager/下输入rails s以启动rails server，然后在浏览器中输入

http://localhost:3000/buckets/就能看到Bucket列表了。

最后保存一下代码：

```
git add .
git commit -m "add list buckets feature"
```

5. 实现List objects功能

首先生成一个管理Objects的controller：

```
rails g controller objects index
```

然后编辑app/controllers/objects_controller.rb:

```
class ObjectsController < ApplicationController
def index
@bucket_name = params[:bucket_id]
@prefix = params[:prefix]
@bucket = OSS.client.get_bucket(@bucket_name)
@objects = @bucket.list_objects(:prefix => @prefix, :delimiter => '/')
end
end
```

上面的代码首先从URL的参数中获取Bucket名字，为了只按目录层级显示，我们还需要一个前缀。然后调用OSS Client的list_objects接口获取文件列表。注意，这里获取的是指定前缀下，并且以’/‘为分界的文件。这样做是为也按目录层级列出文件。请参考管理文件

接下来编辑app/views/objects/index.html.erb:

```
<h1>Objects in <%= @bucket_name %></h1>
<p> <%= link_to 'Upload file', new_object_path(@bucket_name, @prefix) %></p>
<table class="table table-striped">
<tr>
<th>Key</th>
<th>Type</th>
<th>Size</th>
<th>LastModified</th>
</tr>

<tr>
<td><%= link_to '../', with_prefix(upper_dir(@prefix)) %></td>
<td>Directory</td>
<td>N/A</td>
<td>N/A</td>
</tr>

<% @objects.each do |object| %>
<tr>
```

```
<% if object.is_a?(Aliyun::OSS::Object) %>
<td><%= link_to remove_prefix(object.key, @prefix),
@bucket.object_url(object.key) %></td>
<td><%= object.type %></td>
<td><%= number_to_human_size(object.size) %></td>
<td><%= object.last_modified.localtime.to_s %></td>
<% else %>
<td><%= link_to remove_prefix(object, @prefix), with_prefix(object) %></td>
<td>Directory</td>
<td>N/A</td>
<td>N/A</td>
<% end %>
</tr>
<% end %>
</table>
```

上面的代码将文件按目录结构显示，主要逻辑是：

1. 总是在第一个显示’../‘指向上级目录
2. 对于Common prefix，显示为目录
3. 对于Object，显示为文件

上面的代码中用到了with_prefix, remove_prefix等一些辅助函数，它们定义在app/helpers/objects_helper.rb中：

```
module ObjectsHelper
def with_prefix(prefix)
"?prefix=#{prefix}"
end

def remove_prefix(key, prefix)
key.sub(/^#{prefix}/, '')
end

def upper_dir(dir)
dir.sub(/[^\/]+\/$/, '') if dir
end

def new_object_path(bucket_name, prefix = nil)
"/buckets/#{bucket_name}/objects/new/#{with_prefix(prefix)}"
end

def objects_path(bucket_name, prefix = nil)
"/buckets/#{bucket_name}/objects/#{with_prefix(prefix)}"
end
end
```

完成之后运行rails s，然后在浏览器中输入地址http://localhost:3000/buckets/my-bucket/objects/就可以查看文件列表了。

惯例保存代码：

```
git add .
git commit -m "add list objects feature"
```

6. 下载文件

注意到在上一步显示文件列表时，我们为每个文件也添加了一个链接：

```
<td><%= link_to remove_prefix(object.key, @prefix),
@bucket.object_url(object.key) %></td>
```

其中Bucket#object_url是一个为文件生成临时URL的方法，参考下载文件

7. 上传文件

在Rails这种服务端应用中，用户上传文件有两种办法：

1. 用户先将文件上传到Rails的服务器上，服务器再将文件上传到OSS。这样做需要Rails服务器作为中转，文件多拷贝了一遍，不是很高效。
2. 服务器为用户生成表单和临时凭证，用户直接上传文件到OSS。

第一种方法比较简单，与普通的上传文件一样。下面我们用的是第二种方法：

首先在app/controllers/objects_controller.rb中增加一个#new方法，用于生成上传表单：

```
def new
@bucket_name = params[:bucket_id]
@prefix = params[:prefix]
@bucket = OSS.client.get_bucket(@bucket_name)
@options = {
:prefix => @prefix,
:redirect => 'http://localhost:3000/buckets/'
}
end
```

然后编辑app/views/objects/new.html.erb:

```
<h2>Upload object</h2>

<%= upload_form(@bucket, @options) do %>
<table class="table table-striped">
<tr>
<td><label>Bucket:</label></td>
<td><%= @bucket.name %></td>
</tr>
<tr>
<td><label>Prefix:</label></td>
<td><%= @prefix %></td>
</tr>
```

```
<tr>
<td><label>Select file:</label></td>
<td><input type="file" name="file" style="display:inline" /></td>
</tr>

<tr>
<td colspan="2">
<input type="submit" class="btn btn-default" value="Upload" />
<span>  </span>
<%= link_to 'Back', objects_path(@bucket_name, @prefix) %>
</td>
</tr>
</table>
<% end %>
```

其中upload_form是SDK提供的一个方便用户生成上传表单的辅助函数，在SDK的代码rails/aliyun_oss_helper.rb中。用户需要将其拷贝到app/helpers/目录下。完成之后运行rails s，然后访问http://localhost:3000/buckets/my-bucket/objects/new就能够上传文件了。

最后记得保存代码：

```
git add .
git commit -m "add upload object feature"
```

8. 添加样式

为了让界面更好看一些，我们可以添加一点样式（CSS）。

首先下载bootstrap，解压后将bootstrap.min.css拷贝到app/assets/stylesheets/下。

然后在修改app/views/layouts/application.html.erb，将yield一行改成：

```
  <div id="main">
<%= yield %>
</div>
```

这会为每个页面添加一个id为main的 ，然后修改app/assets/stylesheets/application.css，加入以下内容：

```
body {
text-align: center;
}

div#main {
text-align: left;
width: 1024px;
margin: 0 auto;
}
```

这会让网页的主体内容居中显示。通过添加简单的样式，我们的页面是不是更加赏心悦目了呢？

至此，一个简单的demo就完成了。完整的demo代码请参看 Alibaba Cloud OSS Rails Demo。

Go-SDK

安装

SDK安装

- github地址：https://github.com/aliyun/aliyun-oss-go-sdk
- API文档地址：https://godoc.org/github.com/aliyun/aliyun-oss-go-sdk/oss
- 版本迭代：https://github.com/aliyun/aliyun-oss-go-sdk/releases

要求

- 开通阿里云OSS服务，并创建了AccessKeyId 和AccessKeySecret。
- 如果您还没有开通或者还不了解阿里云OSS服务，请登录 OSS产品主页OSS产品主页了解。
- 如果还没有创建AccessKeyId和AccessKeySecret，请到阿里云Access Key管理创建Access Key。
- 您已经安装了Go编译运行环境。如果您未安装，请参考Go安装下载安装编译运行环境。下载安装Go编译运行环境。Go安装完毕后请正确设置GOPATH变量，如果您需要了解更多GOPATH，请执行命令go help gopath。

安装

执行命令go get github.com/aliyun/aliyun-oss-go-sdk/oss安装OSS Go SDK。

说明：

- Windows在cmd中执行安装命令；Linux/Mac在终端执行。
- 安装过程中，界面不会打印提示，请耐心等待。如果安装发生超时，请再次执行go get。

示例

运行示例

- 下载代码文件。从GitHub下载OSS Go SDK的代码。
- 修改sample/config.go里的*Endpoint*、*AccessKeyId*、*AccessKeySecret*、*BucketName*配置。
- 执行go run sample.go。

示例内容

Go SDK的示例代码在sample目录下，地址是 GitHub 。示例包括以下内容：

示例文件	示例内容
new_bucket.go	展示了如何初始化Client、Bucket
put_object.go	展示了简单上传、断点续传上传的用法
append_object.go	展示了追加上传的用法
get_object.go	展示了流式下载、范围下载、断点续传下载的用法
delete_object.go	展示了删除单个文件、批量删除文件的方法
copy_object.go	展示了文件拷贝、文件断点续传拷贝的用法
list_objects.go	展示了列举文件的用法，包括默认参数列举、指定参数列举
object_meta.go	展示了如何设置、读取文件元数据（Object Meta）
object_acl.go	展示了如何设置、读取文件权限（Object ACL）
sign_url.go	展示了如何生成带签名的URL
cname_sample.go	展示了CNAME的用法
create_bucket.go	展示了如何创建存储空间
list_buckets.go	展示了列举存储空间的用法，包括默认参数列举、指定参数列举
bucket_acl.go	展示了如何读取/设置存储空间的权限（Bucket ACL）
bucket_referer.go	展示了如何设置/读取/清除存储空间的白名单（Bucket Referer）
bucket_logging.go	展示了如何设置/读取/清除存储空间的日志（Bucket Logging）
bucket_lifecycle.go	展示了如何设置/读取/清除存储空间中文件的生命周期（Bucket Lifecycle）
bucket_cors.go	展示了如何设置/读取/清除存储空间的跨域访问（Bucket CORS）

快速开始

下面介绍如何使用OSS Go SDK来访问OSS服务，包括查看Bucket列表，查看文件列表，上传/下载文件和删除文件。使用OSS Go SDK，需要引入OSS包import github.com/aliyun/aliyun-oss-go-sdk/oss。

初始化Client

初始化Client，即创建Client：

```
package main

import (
"fmt"
"os"

"github.com/aliyun/aliyun-oss-go-sdk/oss"
)

func handleError(err error) {
fmt.Println("Error:", err)
os.Exit(-1)
}

func main() {
client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
handleError(err)
}

lsRes, err := client.ListBuckets()
if err != nil {
handleError(err)
}

for _, bucket := range lsRes.Buckets {
fmt.Println("bucket:", bucket.Name)
}
}
```

说明：

- Endpoint的是OSS的访问域名，如杭州数据中的访问域名是http://oss-cn-hangzhou.aliyuncs.com，更详细的说明请参考OSS访问域名。
- AccessKeyId和AccessKeySecret是OSS的访问密钥。更详细的说明请参考OSS访问控制。
- 您运行示例程序时，请将Endpoint，AccessKeyId和AccessKeySecret替换成您的实际配置。

查看Bucket列表

通过Client.ListBuckets查看Bucket列表：

```
package main

import (
"fmt"
"os"

"github.com/aliyun/aliyun-oss-go-sdk/oss"
)

func handleError(err error) {
fmt.Println("Error:", err)
os.Exit(-1)
}

func main() {
client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
handleError(err)
}

lsRes, err := client.ListBuckets()
if err != nil {
handleError(err)
}

for _, bucket := range lsRes.Buckets {
fmt.Println("bucket:", bucket.Name)
}
}
```

注意：

- Bucket的命名规范请查看OSS 基本概念。
- Bucket名字不能与OSS服务中其他用户已有的Bucket重复，所以您需要选择一个独特的Bucket名字以避免创建失败。

获取Bucket

Bucket的操作有Client的方法完成，如创建/删除Bucket、设置/清除Bucket的权限/生命周期/防盗链等，Object的操作有Bucket的方法完成，如上传/下载/删除文件、设置Object的访问权限等。用户可以通过Client.Bucket获取指定Bucket的操作句柄。

```
package main
```

```
import (
"fmt"
"os"

"github.com/aliyun/aliyun-oss-go-sdk/oss"
)

func handleError(err error) {
fmt.Println("Error:", err)
os.Exit(-1)
}

func main() {
client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
handleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
handleError(err)
}

_, err = bucket.ListObjects()
if err != nil {
handleError(err)
}
}
```

查看文件列表

通过Bucket.ListObjects查看Bucket中的文件列表：

```
package main

import (
"fmt"
"os"

"github.com/aliyun/aliyun-oss-go-sdk/oss"
)

func handleError(err error) {
fmt.Println("Error:", err)
os.Exit(-1)
}

func main() {
client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
handleError(err)
}

bucket, err := client.Bucket("my-bucket")
```

```
if err != nil {
handleError(err)
}

lsRes, err := bucket.ListObjects()
if err != nil {
handleError(err)
}

for _, object := range lsRes.Objects {
fmt.Println("Object:", object.Key)
}
}
```

上传文件

通过Bucket.PutObjectFromFile上传文件：

```
package main

import (
"fmt"
"os"

"github.com/aliyun/aliyun-oss-go-sdk/oss"
)

func handleError(err error) {
fmt.Println("Error:", err)
os.Exit(-1)
}

func main() {
client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
handleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
handleError(err)
}

err = bucket.PutObjectFromFile("my-object", "LocalFile")
if err != nil {
handleError(err)
}
}
```

其中LocalFile是需要上传的本地文件的路径。上传成功后，可以通过Bucket.ListObjects来查看。

下载文件

通过Bucket.GetObject下载文件：

```
package main

import (
"fmt"
"os"

"github.com/aliyun/aliyun-oss-go-sdk/oss"
)

func handleError(err error) {
fmt.Println("Error:", err)
os.Exit(-1)
}

func main() {
client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
handleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
handleError(err)
}

err = bucket.GetObjectToFile("my-object", "LocalFile")
if err != nil {
handleError(err)
}
}
```

其中LocalFile是文件保存的路径。下载成功后，可以打开文件查看其内容。

删除文件

通过Bucket.DeleteObject从Bucket中删除文件：

```
package main

import (
"fmt"
"os"

"github.com/aliyun/aliyun-oss-go-sdk/oss"
)

func handleError(err error) {
```

```
fmt.Println("Error:", err)
os.Exit(-1)
}

func main() {
client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
handleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
handleError(err)
}

err = bucket.DeleteObject("my-object")
if err != nil {
handleError(err)
}
}
```

删除文件后可以通过Bucket.ListObjects来查看文件确实已经被删除。

了解更多

- 管理Bucket
- 上传文件
- 下载文件
- 管理文件
- 自定义域名绑定
- 设置访问权限
- 管理生命周期
- 设置访问日志
- 静态网站托管
- 设置防盗链
- 设置跨域资源共享
- 错误

管理存储空间

存储空间（Bucket）是OSS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

查看所有Bucket

使用Client.ListBuckets接口列出当前用户下的所有Bucket，用户还可以指定Prefix等参数，列出Bucket名字为特定前缀的所有Bucket：

说明： ListBuckets的示例代码在sample/list_buckets.go。

```
import (
"fmt"
"github.com/aliyun/aliyun-oss-go-sdk/oss"
)

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
// HandleError(err)
}

// 列出Bucket，默认100条。
lsRes, err := client.ListBuckets()
if err != nil {
// HandleError(err)
}
fmt.Println("buckets:", lsRes.Buckets)

// 指定前缀筛选
lsRes, err = client.ListBuckets(oss.Prefix("my-bucket"))
if err != nil {
// HandleError(err)
}
fmt.Println("buckets:", lsRes.Buckets)
```

创建Bucket

说明： CreateBucket的示例代码在sample/create_bucket.go。

使用Client.CreateBucket接口创建一个Bucket，用户需要指定Bucket的名字：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
// HandleError(err)
}

err = client.CreateBucket("my-bucket")
if err != nil {
// HandleError(err)
}
```

创建Bucket时不指定权限，使用默认权限oss.ACLPrivate。创建时用户可以指定Bucket的权限：

```
    err = client.CreateBucket("my-bucket", oss.ACL(oss.ACLPublicRead))
if err != nil {
// HandleError(err)
}
```

说明：

- Bucket的命名规范请查看OSS 基本概念。
- 由于存储空间的名字是全局唯一的，所以必须保证您的Bucket名字不与别人的重复。

删除Bucket

使用Client.DeleteBucket接口删除一个Bucket，用户需要指定Bucket的名字：

```
    import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
// HandleError(err)
}

err = client.DeleteBucket("my-bucket")
if err != nil {
// HandleError(err)
}
```

注意：

- 如果该Bucket下还有文件存在，则需要先删除所有文件才能删除Bucket。
- 如果该Bucket下还有未完成的上传请求，则需要通过Bucket.ListMultipartUploads和Bucket.AbortMultipartUpload先取消请求才能删除Bucket。详细信息请参考API文档。

查看Bucket是否存在

用户可以通过Client.IsBucketExist接口查看当前用户的某个Bucket是否存在：

```
    import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
// HandleError(err)
}
```

```
isExist, err := client.IsBucketExist("my-bucket")
if err != nil {
// HandleError(err)
}
```

Bucket访问权限

用户可以设置Bucket的访问权限，允许或者禁止匿名用户对其内容进行读写。更多关于访问权限的内容请参考访问权限。

说明： Bucket访问权限的示例代码sample/bucket_acl.go。

查看Bucket的访问权限

通过Client.GetBucketACL查看Bucket的ACL：

```
    import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
// HandleError(err)
}

aclRes, err := client.GetBucketACL("my-bucket")
if err != nil {
// HandleError(err)
}
fmt.Println("Bucket ACL:", aclRes.ACL)
```

设置Bucket的访问权限（ACL）

通过Client.SetBucketACL设置Bucket的ACL：

```
    import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
// HandleError(err)
}

err = client.SetBucketACL("my-bucket", oss.ACLPublicRead)
if err != nil {
// HandleError(err)
}
```

说明： Bucket有三种权限私有读写、公共读私有写、公共读写，分布对应Go sdk的常量ACLPrivate、

ACLPublicRead和ACLPublicReadWrite。

上传文件

OSS Go SDK提供了丰富的文件上传接口，用户可以通过以下方式向OSS中上传文件：

- 简单上传PutObject，适合小文件
- 分片上传UploadFile，适合大文件
- 追加上传AppendObject

简单上传

从数据流(io.Reader)中读取数据上传

通过Bucket.PutObject完成简单上传。

提示：

- 简单上传的示例代码在sample/put_object.go。

字符串上传

```
import "strings"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

err = bucket.PutObject("my-object", strings.NewReader("MyObjectValue"))
if err != nil {
    // HandleError(err)
}
```

byte数组上传

```
import "bytes"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

err = bucket.PutObject("my-object", bytes.NewReader([]byte("MyObjectValue")))
if err != nil {
    // HandleError(err)
}
```

文件流上传

```
import "os"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

fd, err := os.Open("LocalFile")
if err != nil {
    // HandleError(err)
}
defer fd.Close()

err = bucket.PutObject("my-object", fd)
if err != nil {
    // HandleError(err)
}
```

根据本地文件名上传

通过Bucket.PutObjectFromFile可以上传指定的本地文件，把本地文件内容作为Object的值。

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
```

```
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

err = bucket.PutObjectFromFile("my-object", "LocalFile")
if err != nil {
    // HandleError(err)
}
```

注意：

- 使用上述方法上传最大文件不能超过5G。如果超过请使用分片上传。

上传时指定元信息

使用数据流上传文件时，用户可以指定一个或多个文件的元信息。元数据的名称大小写不敏感，比如用户上传文件时，定义名字为“name”的meta，使用Bucket.GetObjectDetailedMeta读取结果是：“X-Oss-Meta-Name”，比较/读取时请忽略大小写。

可以指定的元信息如下：

参数	说明
CacheControl	指定该Object被下载时的网页的缓存行为。
ContentDisposition	指定该Object被下载时的名称。
ContentEncoding	指定该Object被下载时的内容编码格式。
Expires	指定过期时间。用户自定义格式，建议使用http.TimeFormat格式。
ServerSideEncryption	指定oss创建object时的服务器端加密编码算法。合法值：AES256。
ObjectACL	指定oss创建object时的访问权限。
Meta	自定义参数，以"X-Oss-Meta-"为前缀的参数。

```
import (
    "strings"
    "time"
    "github.com/aliyun/aliyun-oss-go-sdk/oss"
)

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}
```

```
bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

expires := time.Date(2049, time.January, 10, 23, 0, 0, 0, time.UTC)
options := []oss.Option{
    oss.Expires(expires),
    oss.ObjectACL(oss.ACLPublicRead),
    oss.Meta("MyProp", "MyPropVal"),
}
err = bucket.PutObject("my-object", strings.NewReader("MyObjectValue"), options...)
if err != nil {
    // HandleError(err)
}
```

提示：

- Bucket.PutObject、Bucket.PutObjectFromFile、Bucket.UploadFile、Bucket.UploadFile都支持上传时指定元数据。

创建模拟文件夹

OSS服务是没有文件夹这个概念的，所有元素都是以文件来存储。但给用户提供了创建模拟文件夹的方式，如下代码：

```
import "strings"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

err = bucket.PutObject("my-dir/", strings.NewReader(""))
if err != nil {
    // HandleError(err)
}
```

提示：

- 创建模拟文件夹本质上来说是创建了一个空文件。
- 对于这个文件照样可以上传下载,只是控制台会对以"/"结尾的文件以文件夹的方式展示。
- 所以用户可以使用上述方式来实现创建模拟文件夹。

- 而对文件夹的访问可以参看文件夹功能模拟

追加上传

OSS支持可追加的文件类型，通过Bucket.AppendObject来上传可追加的文件， 调用时需要指定文件追加的位置，对于新创建文件，这个位置是0；对于已经存 在的文件，这个位置必须是追加前文件的长度。

- 文件不存在时，调用AppendObject会创建一个可追加的文件；
- 文件存在时，调用AppendObject会向文件末尾追加内容。

提示：

- 追加上传的示例代码在sample/append_object.go。

```
import "strings"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

var nextPos int64 = 0
// 第一次追加的位置是0，返回值为下一次追加的位置
nextPos, err = bucket.AppendObject("my-object", strings.NewReader("YourObjectValue"), nextPos)
if err != nil {
    // HandleError(err)
}

// 第二次追加
nextPos, err = bucket.AppendObject("my-object", strings.NewReader("YourObjectValue"), nextPos)
if err != nil {
    // HandleError(err)
}

// 您还可以进行多次Append
```

注意：

- 只能向可追加的文件（即通过AppendObject创建的文件）追加内容
- 可追加的文件不能被拷贝

第一次追加时，即位置开始位置是0的追加，您可以指定文件(Object)的元信息；除了第一次追加，其它追加不

能指定元信息。

```
    // 第一次追加指定元信息
    nextPos, err = bucket.AppendObject("my-object", strings.NewReader("YourObjectValue"), 0, oss.Meta("MyProp",
"MyPropVal"))
    if err != nil {
        // HandleError(err)
    }
```

分片上传

当上传大文件时，如果网络不稳定或者程序崩溃了，则整个上传就失败了。用户 不得不重头再来，这样做不仅浪费资源，在网络不稳定的情况下，往往重试多次 还是无法完成上传。 通过Bucket.UploadFile接口来实现断点续传上传。它有以下参数：

- objectKey 上传到OSS的Object名字
- filePath 待上传的本地文件路径
- partSize 分片上传大小，从100KB到5GB，单位是Byte
- options 可选项，主要包括：
    - Routines 指定分片上传的并发数，默认是1，及不使用并发上传
    - Checkpoint 指定上传是否开启断点续传，及checkpoint文件的路径。默认断点续传功能关闭，checkpoint文件的路径可以指定为空，为与本地文件同 目录下的file.cp，其中file是本地文件的名字
    - 其它元信息，请参看上传时指定元信息

其实现的原理是将要上传的文件分成若干个分片分别上传，最后所有分片都上传 成功后，完成整个文件的上传。在上传的过程中会记录当前上传的进度信息（记 录在checkpoint文件中），如果上传过程中某一分片上传失败，再次上传时会从 checkpoint文件中记录的点继续上传。这要求再次调用时要指定与上次相同的checkpoint文件。上传完成后，checkpoint文件会被删除。

提示：

- 分片上传的示例代码在sample/put_object.go。

```
    import "github.com/aliyun/aliyun-oss-go-sdk/oss"

    client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
    if err != nil {
        // HandleError(err)
    }

    bucket, err := client.Bucket("my-bucket")
    if err != nil {
        // HandleError(err)
    }
```

```
// 分片大小100K，3个协程并发上传分片，使用断点续传
err = bucket.UploadFile("my-object", "LocalFile", 100*1024, oss.Routines(3), oss.Checkpoint(true, ""))
if err != nil {
    // HandleError(err)
}
```

注意：

- SDK会将上传的中间状态信息记录在cp文件中，所以要确保用户对cp文件有写权限
- cpt文件记录了上传的中间状态信息并自带了校验，用户不能去编辑它，如 果cpt文件损坏则重新上传所有分片。整个上传完成后cpt文件会被删除。
- 如果上传过程中本地文件发生了改变，则重新上传所有分片

提示：

- 指定断点续传checkpoint文件路径使用oss.Checkpoint(true, "your-cp-file.cp")
- 使用bucket.UploadFile(objectKey, localFile, 100*1024)，默认不使用分片并发上传、不启动断点续传

获取所有已上传的分片信息

您可以用Bucket.ListUploadedParts获取某个分片上传已上传的分片。

```
import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

imur, err := bucket.InitiateMultipartUpload("my-object")
if err != nil {
    // HandleError(err)
}

lsRes, err := bucket.ListUploadedParts(imur)
if err != nil {
    // HandleError(err)
}
fmt.Println("Parts:", lsRes.UploadedParts)
```

获取所有分片上传的任务

通过Bucket.ListMultipartUploads来列出当前分片上传任务。主要的参数如下：

参数	说明

Delimiter	用于对Object名字进行分组的字符。所有名字包含指定的前缀且第一次出现delimiter字符之间的object作为一组元素。
MaxUploads	限定此次返回Multipart Uploads事件的最大数目，默认为1000，max-uploads取值不能大于1000。
KeyMarker	所有Object名字的字典序大于KeyMarker参数值的Multipart事件。
Prefix	限定返回的文件名(object)必须以Prefix作为前缀。注意使用Prefix查询时，返回的文件名(Object)中仍会包含Prefix。

使用默认参数

```
import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

lsRes, err := bucket.ListMultipartUploads()
if err != nil {
    // HandleError(err)
}
fmt.Println("Uploads:", lsRes.Uploads)
```

指定前缀

```
lsRes, err := bucket.ListMultipartUploads(oss.Prefix("my-object-"))
if err != nil {
    // HandleError(err)
}
fmt.Println("Uploads:", lsRes.Uploads)
```

指定最多返回100条结果数据

```
lsRes, err := bucket.ListMultipartUploads(oss.MaxUploads(100))
if err != nil {
    // HandleError(err)
}
fmt.Println("Uploads:", lsRes.Uploads)
```

同时指定前缀和最大返回条数

```
lsRes, err := bucket.ListMultipartUploads(oss.Prefix("my-object-"), oss.MaxUploads(100))
if err != nil {
    // HandleError(err)
}
fmt.Println("Uploads:", lsRes.Uploads)
```

下载文件

OSS Go SDK提供了丰富的文件下载接口，用户可以通过以下方式从OSS中下载文件：

- 下载到流io.ReadCloser
- 下载到本地文件
- 分片下载

下载到流io.ReadCloser

提示：

- 下载的示例代码在sample/get_object.go。

下载文件到流

```
import (
    "fmt"
    "io/ioutil"
    "github.com/aliyun/aliyun-oss-go-sdk/oss"
)

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

body, err := bucket.GetObject("my-object")
if err != nil {
```

```
    // HandleError(err)
}
data, err := ioutil.ReadAll(body)
if err != nil {
    // HandleError(err)
}
body.Close()
fmt.Println("data:", string(data))
```

注意：

- io.ReadCloser数据读取完毕后，需要调用Close关闭。

下载文件到缓存

```
import (
    "bytes"
    "io"
    "github.com/aliyun/aliyun-oss-go-sdk/oss"
)

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

body, err := bucket.GetObject("my-object")
if err != nil {
    // HandleError(err)
}
buf := new(bytes.Buffer)
io.Copy(buf, body)
body.Close()
```

下载文件到本地文件流

```
import (
        "io"
        "os"
        "github.com/aliyun/aliyun-oss-go-sdk/oss"
)

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
```

```
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

body, err := bucket.GetObject("my-object")
if err != nil {
    // HandleError(err)
}
defer body.Close()

fd, err := os.OpenFile("LocalFile", os.O_WRONLY|os.O_CREATE, 0660)
if err != nil {
    // HandleError(err)
}
defer fd.Close()

io.Copy(fd, body)
```

下载到本地文件

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

err = bucket.GetObjectToFile("my-object", "LocalFile")
if err != nil {
    // HandleError(err)
}
```

分片下载

当下载大文件时，如果网络不稳定或者程序崩溃了，则整个下载就失败了。用户 不得不重头再来，这样做不仅浪费资源，在网络不稳定的情况下，往往重试多次 还是无法完成下载。 通过Bucket.DownloadFile接口来实现断点续传下载。它有以下参数：

- objectKey 要下载的Object名字
- filePath 下载到本地文件的路径
- partSize 下载分片大小，从1B到5GB，单位是Byte
- options 可选项，主要包括：

- Routines 指定分片下载的并发数，默认是1，及不使用并发下载
- Checkpoint 指定下载是否开启断点续传，及checkpoint文件的路径。默认断点续传功能关闭，checkpoint文件的路径可以指定为空，如果不指定则默认为与本地文件同 目录下的file.cpt，其中file是本地文件的名字
- 下载时限定条件，请参看指定限定条件下载

其实现的原理是将要下载的Object分成若干个分片分别下载，最后所有分片都下 载成功后，完成整个文件的下载。在下载的过程中会记录当前下载的进度信息（记录在checkpoint文件中）和已下载的分片，如果下载过程中某一分片下载失败，再次下 载时会从checkpoint文件中记录的点继续下载。这要求再次调用时要指定与上次相同的checkpoint文件。下载完成后，checkpoint文件会被删除。

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

err = bucket.DownloadFile("my-object", "LocalFile", 100*1024, oss.Routines(3), oss.Checkpoint(true, ""))
if err != nil {
    // HandleError(err)
}
```

注意：

- SDK会将下载的中间状态信息记录在cp文件中，所以要确保用户对cpt文 件有写权限
- cpt文件记录了下载的中间状态信息并自带了校验，用户不能去编辑它，如 果cpt文件损坏则重新下载文件
- 如果下载过程中待下载的Object发生了改变（ETag改变），或者part文件丢 失或被修改，则重新下载文件

提示：

- 指定断点续传checkpoint文件路径使用oss.Checkpoint(true, "your-cp-file.cp")
- 使用bucket.DownloadFile(objectKey, localFile, 100*1024)，默认不使用分片并发下载、不启动断点续传

指定限定条件下载

下载文件时，用户可以指定一个或多个限定条件，所有的限定条件都满足时下载，不满足时报错不下载文件。可以使用的限定条件如下：

参数	说明

IfModifiedSince	如果指定的时间早于实际修改时间，则正常传送。否则返回错误。
IfUnmodifiedSince	如果传入参数中的时间等于或者晚于文件实际修改时间，则正常传输文件；否则返回错误。
IfMatch	如果传入期望的ETag和object的 ETag匹配，则正常传输；否则返回错误。
IfNoneMatch	如果传入的ETag值和Object的ETag不匹配，则正常传输；否则返回错误。

```
import "time"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

date := time.Date(2015, time.November, 10, 23, 0, 0, 0, time.UTC)

// 限定条件不满足，不下载文件
err = bucket.GetObjectToFile("my-object", "LocalFile", oss.IfModifiedSince(date))
if err == nil {
    // HandleError(err)
}

// 满足限定条件，下载文件
err = bucket.GetObjectToFile("my-object", "LocalFile", oss.IfUnmodifiedSince(date))
if err != nil {
    // HandleError(err)
}
```

提示：

- ETag的值可以通过Bucket.GetObjectDetailedMeta获取。
- Bucket.GetObject，Bucket.GetObjectToFile，Bucket.DownloadFile都支持限定条件。

文件压缩下载

文件可以压缩下载，目前支持GZIP压缩。Bucket.GetObject、Bucket.GetObjectToFile支持压缩功能。

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"
```

```
client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

err = bucket.GetObjectToFile("my-object.txt", "LocalFile.gzip", oss.AcceptEncoding("gzip"))
if err != nil {
    // HandleError(err)
}
```

管理文件

一个Bucket下可能有非常多的文件，SDK提供一系列的接口方便用户管理文件。

列出存储空间中的文件

通过Bucket.ListObjects来列出当前Bucket下的文件。主要的参数如下：

参数	说明
Delimiter	用于对Object名字进行分组的字符。所有名字包含指定的前缀且第一次出现delimiter字符之间的object作为一组元素CommonPrefixes
Prefix	限定返回的object key必须以prefix作为前缀。注意使用prefix查询时，返回的key中仍会包含prefix
MaxKeys	限定此次返回object的最大数，如果不设定，默认为100，max-keys取值不能大于1000
Marker	设定结果从marker之后按字母排序的第一个开始返回

提示：

- ListObjects的示例代码在sample/list_objects.go

使用默认参数获取存储空间的文件列表，默认返回100条Object

```
import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

lsRes, err := bucket.ListObjects()
if err != nil {
    // HandleError(err)
}

fmt.Println("Objects:", lsRes.Objects)
```

指定最大返回数量，最多不能超过1000条

```
lsRes, err := bucket.ListObjects(oss.MaxKeys(200))
if err != nil {
    // HandleError(err)
}
fmt.Println("Objects:", lsRes.Objects)
```

返回指定前缀的Object，默认最多返回100条

```
lsRes, err := bucket.ListObjects(oss.Prefix("my-object-"))
if err != nil {
    // HandleError(err)
}
fmt.Println("Objects:", lsRes.Objects)
```

指定从某个Object(my-object-xx)后返回，默认最多100条

```
lsRes, err := bucket.ListObjects(oss.Marker("my-object-xx"))
if err != nil {
    // HandleError(err)
}
fmt.Println("Objects:", lsRes.Objects)
```

分页获取所有Object，每次返回200条

```
marker := oss.Marker("")
```

```
for {
    lsRes, err := bucket.ListObjects(oss.MaxKeys(200), marker)
    if err != nil {
        HandleError(err)
    }
    marker = oss.Marker(lsRes.NextMarker)

    fmt.Println("Objects:", lsRes.Objects)

    if !lsRes.IsTruncated {
        break
    }
}
```

分页所有获取从特定Object后的所有的Object，每次返回50条

```
marker = oss.Marker("my-object-xx")
for {
    lsRes, err := bucket.ListObjects(oss.MaxKeys(50), marker)
    if err != nil {
        // HandleError(err)
    }

    marker = oss.Marker(lsRes.NextMarker)

    fmt.Println("Objects:", lsRes.Objects)

    if !lsRes.IsTruncated {
        break
    }
}
```

分页所有获取指定前缀为的Object，每次返回80个。

```
prefix := oss.Prefix("my-object-")
marker := oss.Marker("")
for {
    lsRes, err := bucket.ListObjects(oss.MaxKeys(80), marker, prefix)
    if err != nil {
        // HandleError(err)
    }

    prefix = oss.Prefix(lsRes.Prefix)
    marker = oss.Marker(lsRes.NextMarker)

    fmt.Println("Objects:", lsRes.Objects)

    if !lsRes.IsTruncated {
        break
    }
}
```

模拟目录结构

OSS是基于对象的存储服务，没有目录的概念。存储在一个Bucket中所有文件都 是通过文件的key唯一标识，并没有层级的结构。这种结构可以让OSS的存储非常 高效，但是用户管理文件时希望能够像传统的文件系统一样把文件分门别类放到 不同的“目录”下面。通过OSS提供的“公共前缀”的功能，也可以很方便地模拟目录 结构。公共前缀的概念请参考 列出Object 。

假设Bucket中已有如下文件：

```
foo/x
foo/y
foo/bar/a
foo/bar/b
foo/hello/C/1
foo/hello/C/2
```

通过ListObjects，列出指定目录下的文件和子目录：

```
lsRes, err := bucket.ListObjects(oss.Prefix("foo/"), oss.Delimiter("/"))
if err != nil {
    // HandleError(err)
}
fmt.Println("Objects:", lsRes.Objects,"SubDir:", lsRes.CommonPrefixes)
```

结果中lsRes.Objects为文件，包括foo/x、foo/y；lsRes.CommonPrefixes即子目录，包括foo/bar/、foo/hello/。

判断文件是否存在

通过Bucket.IsObjectExist来判断文件是否存在。

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

isExist, err := bucket.IsObjectExist("my-object")
if err != nil {
    // HandleError(err)
}
```

删除单个文件

通过Bucket.DeleteObject来删除某个文件：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

err = bucket.DeleteObject("my-object")
if err != nil {
    // HandleError(err)
}
```

删除多个文件

通过Bucket.DeleteObjects来删除多个文件，用户可以通过DeleteObjectsQuiet参 数来指定是否返回删除的结果。默认返回删除结果。

提示：

- 删除文件的示例代码在sample/delete_object.go

```
import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

// 默认返回删除成功的文件
delRes, err := bucket.DeleteObjects([]string{"my-object-1", "my-object-2"})
if err != nil {
    // HandleError(err)
}
fmt.Println("Deleted Objects:", delRes.DeletedObjects)
```

```
// 不返回删除的结果
_, err = bucket.DeleteObjects([]string{"my-object-3", "my-object-4"},
    oss.DeleteObjectsQuiet(true))
if err != nil {
    // HandleError(err)
}
```

注意：

- Bucket.DeleteObjects至少有一个ObjectKey，不能为空。
- Bucket.DeleteObjects使用的Go的xml包，该包实现了XML1.0标准，XML1.0不支持的特性请不要使用。

获取文件的元信息(Object Meta)

OSS上传/拷贝文件时，除了文件内容，还可以指定文件的一些属性信息，称为“元信息”。这些信息在上传时与文件一起存储，在下载时与文件一起返回。

在SDK中文件元信息用一个Map表示，其他key和value都是string类型，并 且都**只能是简单的ASCII可见字符，不能包含换行。** 所有元信息的总大小不 能超过8KB。

注意：

- 因为文件元信息在上传/下载时是附在HTTP Headers中， HTTP协议规定不能 包含复杂字符。
- 元数据的名称大小写不敏感，比较/读取时请忽略大小写。

使用Bucket.GetObjectDetailedMeta来获取Object的元信息。

提示：

- 元信息的示例代码在sample/object_meta.go

```
import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

props, err := bucket.GetObjectDetailedMeta("my-object")
```

```
if err != nil {
    // HandleError(err)
}
fmt.Println("Object Meta:", props)
```

提示：

- Bucket.GetObjectMeta的结果中不包括Object的权限，获取Object权限通过Bucket.GetObjectACL。

修改文件元信息(Object Meta)

用户一次修改一条或多条元信息，可用元信息如下：

元信息	说明
CacheControl	指定新Object被下载时的网页的缓存行为。
ContentDisposition	指定新Object被下载时的名称。
ContentEncoding	指定新Object被下载时的内容编码格式。
Expires	指定新Object过期时间，建议使用GMT格式。
Meta	自定义参数，以"X-Oss-Meta-"为前缀的参数。

使用Bucket.SetObjectMeta来设置Object的元信息。

```
import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

// 一次修改一条Meta
err = bucket.SetObjectMeta("my-object", oss.Meta("MyMeta", "MyMetaValue"))
if err != nil {
    // HandleError(err)
}

// 修改多条Meta
options := []oss.Option{
    oss.Meta("MyMeta", "MyMetaValue"),
    oss.Meta("MyObjectLocation", "HangZhou"),
}
```

```
err = bucket.SetObjectMeta("my-object", options...)
if err != nil {
    // HandleError(err)
}
```

拷贝文件

使用Bucket.CopyObject或Bucket.CopyObjectToBucket拷贝文件，前者是同一个Bucket内的文件拷贝，后者是Bucket之间的文件拷贝。

提示：

- 拷贝文件的示例代码在sample/copy_objects.go

同一个Bucket内的文件拷贝

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

_, err = bucket.CopyObject("my-object", "descObjectKey")
if err != nil {
    // HandleError(err)
}
```

不同Bucket之间的文件拷贝

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

_, err = bucket.CopyObjectToBucket("my-object", "my-bucket-desc", "descObjectKey")
```

```
if err != nil {
    // HandleError(err)
}
```

拷贝时处理文件元信息

文件拷贝(CopyObject/CopyObjectToBucket)时对文件元信息的处理有两种选择，通过MetadataDirective参数指定：

- oss.MetaCopy 与源文件相同，即拷贝源文件的元信息
- oss.MetaReplace 使用新的元信息覆盖源文件的信息

默认值是oss.MetaCopy。

COPY时，MetadataDirective为MetaReplace时，用户可以指定新对象的如下的元信息：

元信息	说明
CacheControl	指定新Object被下载时的网页的缓存行为。
ContentDisposition	指定新Object被下载时的名称。
ContentEncoding	指定新Object被下载时的内容编码格式。
Expires	指定新Object过期时间（seconds）更详细描述请参照RFC2616。
ServerSideEncryption	指定OSS创建新Object时的服务器端加密编码算法。
ObjectACL	指定OSS创建新Object时的访问权限。
Meta	自定义参数，以"X-Oss-Meta-"为前缀的参数。

```
import (
    "time"
    "github.com/aliyun/aliyun-oss-go-sdk/oss"
)

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

expires := time.Date(2049, time.January, 10, 23, 0, 0, 0, time.UTC)
options := []oss.Option{
    oss.MetadataDirective(oss.MetaReplace),
    oss.Expires(expires),
    oss.ObjectACL(oss.ACLPublicRead),
```

```
    oss.Meta("MyMeta", "MyMetaValue")}

_, err = bucket.CopyObject("my-object", "descObjectKey", options...)
if err != nil {
    // HandleError(err)
}
```

限定拷贝条件

文件拷贝时可以设置限定条件，条件满足时拷贝，不满足时报错不拷贝。可以使用的限定条件如下：

参数	说明
CopySourceIfMatch	如果源Object的ETAG值和用户提供的ETAG相等，则执行拷贝操作；否则返回错误。
CopySourceIfNoneMatch	如果源Object的ETAG值和用户提供的ETAG不相等，则执行拷贝操作；否则返回错误。
CopySourceIfModifiedSince	如果传入参数中的时间等于或者晚于源文件实际修改时间，则正常拷贝；否则返回错误。
CopySourceIfUnmodifiedSince	如果源Object自从用户指定的时间以后被修改过，则执行拷贝操作；否则返回错误。

```
import (
    "fmt"
    "time"
    "github.com/aliyun/aliyun-oss-go-sdk/oss"
)

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

date := time.Date(2015, time.November, 10, 23, 0, 0, 0, time.UTC)
// 约束条件不满足，拷贝没有执行
_, err = bucket.CopyObject("my-object", "descObjectKey", oss.CopySourceIfModifiedSince(date))
fmt.Println("CopyObjectError:", err)

// 约束条件满足，拷贝执行
_, err = bucket.CopyObject("my-object", "descObjectKey", oss.CopySourceIfUnmodifiedSince(date))
if err != nil {
    // HandleError(err)
}
```

提示：

- Bucket.CopyObject、Bucket.CopyObjectToBucket都支持拷贝时处理文件元信息、限定拷贝条件。

自定义域名绑定

OSS支持用户将自定义的域名绑定到OSS服务上，这样能够支持用户无缝地将存储 迁移到OSS上。例如用户的域名是my-domain.com，之前用户的所有图片资源都是 形如http://img.my-domain.com/xx.jpg的格式，用户将图片存储迁移到OSS之 后，通过绑定自定义域名，仍可以使用原来的地址访问到图片：

- 开通OSS服务并创建Bucket
- 将img.my-domain.com与创建的Bucket绑定
- 将图片上传到OSS的这个Bucket中
- 修改域名的DNS配置，增加一个CNAME记录，将img.my-domain.com指向OSS服务 的endpoint（如my-bucket.oss-cn-hangzhou.aliyuncs.com）

这样就可以通过原地址http://img.my-domain.com/x.jpg访问到存储在OSS上 的图片。 绑定自定义域名请参考自定义域名绑定

在使用SDK时，可以使用自定义域名作为endpoint，这时需要将UseCname参数 设置为true，如下面的例子：

提示：

- 跨域资源共享的示例代码在sample/cname_sample.go。

```
import (
    "fmt"
    "io/ioutil"
    "strings"
    "github.com/aliyun/aliyun-oss-go-sdk/oss"
)

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret", oss.UseCname(true))
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
    // HandleError(err)
}

err = bucket.PutObject("my-object", strings.NewReader("MyObjectValue"))
if err != nil {
    // HandleError(err)
```

```
}

body, err := bucket.GetObject("my-object")
if err != nil {
    // HandleError(err)
}
data, err := ioutil.ReadAll(body)
if err != nil {
    // HandleError(err)
}
body.Close()
data = data  // 处理数据

lsRes, err := bucket.ListObjects()
if err != nil {
    // HandleError(err)
}
fmt.Println("Objects:", lsRes.Objects)

err = bucket.DeleteObject("my-object")
if err != nil {
    // HandleError(err)
}
```

注意：

- 使用Cname时，无法使用list_buckets接口。（因为自定义域名已经绑定到 某个特定的Bucket）

设置访问权限

OSS允许用户对Bucket和Object分别设置访问权限，方便用户控制自己的资源可 以被如何访问。对于Bucket，有三种访问权限：

- public-read-write 允许匿名用户向该Bucket中创建/获取/删除Object
- public-read 允许匿名用户获取该Bucket中的Object
- private 不允许匿名访问，所有的访问都要经过签名

创建Bucket时，默认是private权限。之后用户可以通过Client.SetBucketACL来设置 Bucket的权限。上面三种权限分布对应Go SDK中的常量ACLPublicReadWrite、ACLPublicRead、ACLPrivate。

Bucket访问权限

提示：

- Bucket访问权限设置的示例代码在sample/bucket_acl.go。

```
import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

// 设置Bucket ACL
err = client.SetBucketACL("my-bucket", oss.ACLPublicRead)
if err != nil {
    // HandleError(err)
}

// 查看Bucket ACL
aclRes, err := client.GetBucketACL("my-bucket")
if err != nil {
    // HandleError(err)
}
fmt.Println("Bucket ACL:", aclRes.ACL)
```

Object访问权限

对于Object，有四种访问权限：

- default 继承所属的Bucket的访问权限，即与所属Bucket的权限值一样
- public-read-write 允许匿名用户读写该Object
- public-read 允许匿名用户读该Object
- private 不允许匿名访问，所有的访问都要经过签名

创建Object时，默认为default权限。之后用户可以通过 Bucket.SetObjectACL来设置Object的权限。上面四种权限分布对应 Go SDK中的常量ACLDefault、ACLPublicReadWrite、ACLPublicRead、ACLPrivate。

提示：

- Object访问权限设置的示例代码在sample/object_acl.go。

```
import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

bucket, err := client.Bucket("my-bucket")
if err != nil {
```

```
    // HandleError(err)
}

// 设置Object的访问权限
err = bucket.SetObjectACL("my-object", oss.ACLPrivate)
if err != nil {
    // HandleError(err)
}

// 查看Object的访问权限
aclRes, err := bucket.GetObjectACL("my-object")
if err != nil {
    // HandleError(err)
}
fmt.Println("Object ACL:", aclRes.ACL)
```

注意：

- 如果设置了Object的权限（非default），则访问该Object时进行权限认证时 会优先判断Object的权限，而Bucket的权限设置会被忽略。
- 允许匿名访问时（设置了public-read或者public-read-write权限），用户 可以直接通过浏览器访问，例如：http://bucket-name.oss-cn-hangzhou.aliyuncs.com/object.jpg

更多关于访问权限控制的内容请参考 访问控制

授权访问

使用URL签名授权访问

通过生成签名URL的形式提供给用户一个临时的访问URL。在生成URL时，您可以指定URL过期的时间，从而限制用户长时间访问。

提示：

- 以下场景的完整代码请参考：GitHub

使用私有的下载链接

生成GetObject的签名url示例如下：

```
package main
```

```
import (
"fmt"
"io/ioutil"

"github.com/aliyun/aliyun-oss-go-sdk/oss"
)

func HandleError(err error) {
// Handle Error
}

func main() {
client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
HandleError(err)
}

bucket, err := client.Bucket("BucketName")
if err != nil {
HandleError(err)
}

// get object
signedURL, err := bucket.SignURL("ObjectName", oss.HTTPGet, 60)
if err != nil {
HandleError(err)
}

body, err := bucket.GetObjectWithURL(signedURL)
if err != nil {
HandleError(err)
}
// read content
data, err := ioutil.ReadAll(body)
body.Close()
data = data // use data

err = bucket.GetObjectToFileWithURL(signedURL, "localFile")
if err != nil {
HandleError(err)
}
}
```

提示：

- 生成的URL可以直接通过浏览器访问相关内容。

使用私有的上传链接

如果您想允许用户临时进行其它操作（比如上传，删除文件），可能需要签名其它方法的URL，如下：

```
package main
```

```
import (
"fmt"
"strings"

"github.com/aliyun/aliyun-oss-go-sdk/oss"
)

func HandleError(err error) {
// Handle Error
}

func main() {
client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
HandleError(err)
}

bucket, err := client.Bucket("BucketName")
if err != nil {
HandleError(err)
}

// put object
signedURL, err := bucket.SignURL("ObjectName", oss.HTTPPut, 60)
if err != nil {
HandleError(err)
}

var val = "花间一壶酒，独酌无相亲。 举杯邀明月，对影成三人。"
err = bucket.PutObjectWithURL(signedURL, strings.NewReader(val))
if err != nil {
HandleError(err)
}

// put object with option
options := []oss.Option{
oss.Meta("myprop", "mypropval"),
oss.ContentType("image/tiff"),
}

signedURL, err = bucket.SignURL("ObjectName", oss.HTTPPut, 60, options...)
if err != nil {
HandleError(err)
}

err = bucket.PutObjectFromFileWithURL(signedURL, "localFile", options...)
if err != nil {
HandleError(err)
}
}
```

临时凭证(STS)上传和下载

介绍

OSS可以通过阿里云STS服务，临时进行授权访问。阿里云STS (Security Token Service) 是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或联邦用户（用户身份由您自己管理）颁发一个自定义时效和权限的访问凭证。STS更详细的解释请参考 STS介绍。

使用STS凭证创建OssClient

用户的client端拿到STS临时凭证后，通过其中安全令牌(SecurityToken)以及临时访问密钥(AccessKeyId, AccessKeySecret)生成Client。

通过下面代码可以使用STS临时凭证创建Client：

```
package main

import (
"bytes"
"fmt"

"github.com/aliyun/aliyun-oss-go-sdk/oss"
)

func HandleError(err error) {
// Handle Error
}

func main() {
client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret", oss.SecurityToken("StsToken"))
if err != nil {
HandleError(err)
}

bucket, err := client.Bucket("BucketName")
if err != nil {
HandleError(err)
}

err = bucket.PutObject("my-object", bytes.NewReader([]byte("MyObjectValue")))
if err != nil {
HandleError(err)
}

fmt.Printf("Completed\n")
}
```

管理生命周期

OSS允许用户对Bucket设置生命周期规则，以自动淘汰过期掉的文件，节省存储 空间。用户可以同时设置多条规则，一条规则包含：

- 规则ID，用于标识一条规则，不能重复
- 受影响的文件前缀，此规则只作用于符合前缀的文件
- 过期时间，有两种指定方式：
    i. 指定距文件最后修改时间N天过期
    ii. 指定在具体的某一天过期，即在那天之后符合前缀的文件将会过期，**而 不论文件的最后修改时间。**不推荐使用。
- 是否生效

更多关于生命周期的内容请参考 文件生命周期

提示：

- 管理生命周期的示例代码在sample/bucket_lifecycle.go。

设置生命周期规则

通过Client.SetBucketLifecycle来设置生命周期规则：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

// id:"rule1", enable:true, prefix:"foo/", expiry:Days 3
rule1 := oss.BuildLifecycleRuleByDays("rule1", "foo/", true, 3)
// id:"rule2", enable:false, prefix:"bar/", expiry:Date 2016/1/1
rule2 := oss.BuildLifecycleRuleByDate("rule2", "bar/", true, 2016, 1, 1)
rules := []oss.LifecycleRule{rule1, rule2}

err = client.SetBucketLifecycle("my-bucket", rules)
if err != nil {
    // HandleError(err)
}
```

查看生命周期规则

通过Client.GetBucketLifecycle来查看生命周期规则：

```
import "fmt"
```

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

lcRes, err := client.GetBucketLifecycle("my-bucket")
if err != nil {
    // HandleError(err)
}
fmt.Println("Lifecycle Rules:", lcRes.Rules)
```

清空生命周期规则

通过Client.DeleteBucketLifecycle设置一个空的Rule数组来清空生命周期规则：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

err = client.DeleteBucketLifecycle("my-bucket")
if err != nil {
    // HandleError(err)
}
```

设置访问日志

OSS允许用户对Bucket设置访问日志记录，设置之后对于Bucket的访问会被记录 成日志，日志存储在OSS上由用户指定的Bucket中，文件的格式为：

```
<TargetPrefix><SourceBucket>-YYYY-mm-DD-HH-MM-SS-UniqueString
```

其中TargetPrefix由用户指定。日志规则由以下3项组成：

- enable，是否开启
- target_bucket，存放日志文件的Bucket
- target_prefix，指定最终被保存的访问日志文件前缀

更多关于访问日志的内容请参考 Bucket访问日志

提示：

- Bucket访问权限设置的示例代码在sample/bucket_logging.go。

开启Bucket日志

通过Client.SetBucketLogging来开启日志功能：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

// target_bucket:"my-target-bucket", target_prefix:"my-object-", enable: true
err = client.SetBucketLogging("my-bucket", "my-target-bucket", "my-object-", true)
if err != nil {
    // HandleError(err)
}
```

查看Bucket日志设置

通过Client.GetBucketLogging来查看日志设置：

```
import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

logRes, err := client.GetBucketLogging("my-bucket")
if err != nil {
    // HandleError(err)
}
fmt.Println("Target Bucket:", logRes.LoggingEnabled.TargetBucket,
        "Target Prefix:", logRes.LoggingEnabled.TargetPrefix)
```

关闭Bucket日志

通过Bucket.DeleteBucketLogging来关闭日志功能：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"
```

```
client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

err = client.DeleteBucketLogging("my-bucket")
if err != nil {
    // HandleError(err)
}
```

静态网站托管

在自定义域名绑定中提到，OSS 允许用户将自己的域名指向OSS服务的地址。这样用户访问他的网站的时候，实 际上是在访问OSS的Bucket。对于网站，需要指定首页(index)和出错页(error) 分别对应的Bucket中的文件名。

更多关于静态网站托管的内容请参考 静态网站托管

设置托管页面

通过Client.SetBucketWebsite来设置托管页面：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

// bucketName:"my-bucket", indexWebsite:"index.html", errorWebsite:"error.html"
err = client.SetBucketWebsite("my-bucket", "index.html", "error.html")
if err != nil {
    // HandleError(err)
}
```

查看托管页面

通过Client.GetBucketWebsite来查看托管页面：

```
import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
```

```
if err != nil {
    // HandleError(err)
}

wsRes, err := client.GetBucketWebsite("my-bucket")
if err != nil {
    // HandleError(err)
}

fmt.Println("indexWebsite:", wsRes.IndexDocument.Suffix,
        "errorWebsite:", wsRes.ErrorDocument.Key)
```

清除托管页面

通过Client.DeleteBucketWebsite来清除托管页面：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

err = client.DeleteBucketWebsite("my-bucket")
if err != nil {
    // HandleError(err)
}
```

设置防盗链

OSS是按使用收费的服务，为了防止用户在OSS上的数据被其他人盗链，OSS支持 基于HTTP header中表头字段referer的防盗链方法。更多OSS防盗链请参考： OSS 防盗链

提示：

- 设置防盗链的示例代码在sample/bucket_referer.go。

设置Referer白名单

通过Bucket.SetBucketReferer设置Referer白名单，该函数有三个参数：

- bucketName 存储空间名称。
- referers 访问白名单列表。一个bucket可以支持多个referer参数。eferer参数支持通配符"*"和"?"。

- allowEmptyReferer 指定是否允许referer字段为空的请求访问。默认配置为true。

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

referers := []string{"http://www.aliyun.com",
                     "http://www.???.aliyuncs.com",
                     "http://www.*.com"}
err = client.SetBucketReferer("my-bucket", referers, false)
if err != nil {
    // HandleError(err)
}
```

查看Referer白名单

通过Bucket.GetBucketReferer设置Referer白名单：

```
import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

refRes, err := client.GetBucketReferer("my-bucket")
if err != nil {
    // HandleError(err)
}
fmt.Println("Referers:", refRes.RefererList,
            "AllowEmptyReferer:", refRes.AllowEmptyReferer)
```

清空Referer白名单

清空Referer白名单，即把白名单设置成空，allowEmptyReferer为true：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

err = client.SetBucketReferer("my-bucket", []string{}, true)
if err != nil {
    // HandleError(err)
```

```
}
```

设置跨域资源共享

跨域资源共享(CORS)允许web端的应用程序访问不属于本域的资源。OSS提供接口 方便开发者控制跨域访问的权限。更多关于跨域资源共享的内容请参考 OSS 跨域资源共享

OSS的跨域共享设置由一条或多条CORS规则组成，每条CORS规则包含以下设置：

- allowed_origins，允许的跨域请求的来源，如www.my-domain.com, *
- allowed_methods，允许的跨域请求的HTTP方法(PUT/POST/GET/DELETE/HEAD)
- allowed_headers，在OPTIONS预取指令中允许的header，如x-oss-test, *
- expose_headers，允许用户从应用程序中访问的响应头
- max_age_seconds, 浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间

提示：

- 跨域资源共享的示例代码在sample/bucket_cors.go。

设置CORS规则

通过Client.SetBucketCORS设置CORS规则：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

rule1 := oss.CORSRule{
    AllowedOrigin: []string{"*"},
    AllowedMethod: []string{"PUT", "GET"},
    AllowedHeader: []string{},
    ExposeHeader:  []string{},
    MaxAgeSeconds: 200,
}

rule2 := oss.CORSRule{
    AllowedOrigin: []string{"http://www.a.com", "http://www.b.com"},
    AllowedMethod: []string{"POST"},
    AllowedHeader: []string{"Authorization"},
    ExposeHeader:  []string{"x-oss-test", "x-oss-test1"},
```

```
    MaxAgeSeconds: 100,
}

err = client.SetBucketCORS("my-bucket", []oss.CORSRule{rule1, rule2})
if err != nil {
    // HandleError(err)
}
```

查看CORS规则

通过Client.GetBucketCORS查看CORS规则：

```
import "fmt"
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

corsRes, err := client.GetBucketCORS("my-bucket")
if err != nil {
    // HandleError(err)
}
fmt.Println("Bucket CORS:", corsRes.CORSRules)
```

清空CORS规则

通过Client.DeleteBucketCORS清空CORS规则

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"

client, err := oss.New("Endpoint", "AccessKeyId", "AccessKeySecret")
if err != nil {
    // HandleError(err)
}

err = client.DeleteBucketCORS("my-bucket")
if err != nil {
    // HandleError(err)
}
```

错误

Go中调用出错，会统一返回接口error，该接口定义如下：

```
type error interface {
    Error() string
}
```

其它的错误继承于该接口。比如HTTP错误请求返回的错误如下：

```
//net.Error
type Error interface {
    error
    Timeout() bool   // Is the error a timeout
    Temporary() bool // Is the error temporary
}
```

使用OSS Go SDK时如果请求出错，会有相应的error返回。HTTP请求、IO等错误会返回Go预定的错误。 OSS Server处理请求出错，返回如下的错误，该错误实现了error接口。

```
type ServiceError struct {
    Code       string    // OSS返回给用户的错误码
    Message    string    // OSS给出的详细错误信息
    RequestId  string    // 用于唯一标识该次请求的UUID
    HostId     string    // 用于标识访问的OSS集群
    StatusCode int       // HTTP状态码
}
```

如果OSS返回的HTTP状态码与预期不符返回如下错误，该错误也实现了error接口。

```
type UnexpectedStatusCodeError struct {
    allowed []int    // 预期OSS返回HTTP状态码
    got     int      // OSS实际返回HTTP状态码
}
```

OSS的错误码

OSS的错误码列表如下：

错误码	描述	HTTP状态码
AccessDenied	拒绝访问	403
BucketAlreadyExists	Bucket已经存在	409
BucketNotEmpty	Bucket不为空	409
EntityTooLarge	实体过大	400
EntityTooSmall	实体过小	400

FileGroupTooLarge	文件组过大	400
InvalidLinkName	Object Link与指向的Object同名	400
LinkPartNotExist	Object Link中指向的Object不存在	400
ObjectLinkTooLarge	Object Link中Object个数过多	400
FieldItemTooLong	Post请求中表单域过大	400
FilePartInterity	文件Part已改变	400
FilePartNotExist	文件Part不存在	400
FilePartStale	文件Part过时	400
IncorrectNumberOfFilesInPOSTRequest	Post请求中文件个数非法	400
InvalidArgument	参数格式错误	400
InvalidAccessKeyId	AccessKeyId不存在	403
InvalidBucketName	无效的Bucket名字	400
InvalidDigest	无效的摘要	400
InvalidEncryptionAlgorithmError	指定的熵编码加密算法错误	400
InvalidObjectName	无效的Object名字	400
InvalidPart	无效的Part	400
InvalidPartOrder	无效的part顺序	400
InvalidPolicyDocument	无效的Policy文档	400
InvalidTargetBucketForLogging	Logging操作中有无效的目标bucket	400
InternalError	OSS内部发生错误	500
MalformedXML	XML格式非法	400
MalformedPOSTRequest	Post请求的body格式非法	400
MaxPOSTPreDataLengthExceededError	Post请求上传文件内容之外的body过大	400
MethodNotAllowed	不支持的方法	405
MissingArgument	缺少参数	411
MissingContentLength	缺少内容长度	411
NoSuchBucket	Bucket不存在	404
NoSuchKey	文件不存在	404
NoSuchUpload	Multipart Upload ID不存在	404
NotImplemented	无法处理的方法	501

PreconditionFailed	预处理错误	412
RequestTimeTooSkewed	发起请求的时间和服务器时间超出15分钟	403
RequestTimeout	请求超时	400
RequestIsNotMultiPartContent	Post请求content-type非法	400
SignatureDoesNotMatch	签名错误	403
TooManyBuckets	用户的Bucket数目超过限制	400
InvalidEncryptionAlgorithmError	指定的熵编码加密算法错误	400

提示：

- 上表的错误码即OssServiceError.Code，HTTP状态码即OssServiceError.StatusCode。
- 如果试图以OSS不支持的操作来访问某个资源，返回405 Method Not Allowed错误。

Media-C-SDK

前言

简介

不少情况下，我们都需要将摄像头拍摄的视频快速存储到云端（OSS），但是我们也有一些因素要考虑：

- 设备上不能存储永久access key id和access key secret，因为可能泄露。
- 设备上只允许上传或者下载，不允许删除、修改配置等管理权限。
- 可以提供网页让用户去管理自己的视频。
- 对设备的权限精准控制。
- 对设备的权限存在有效期，不能让设备永久持有某种权限。
- 希望摄像机输出的音视频可以通过HLS协议直接被用户观看。

针对以上考虑，我们推出了OSS MEDIA C SDK，构建于OSS C SDK版本之上，可以方便的解决上述问题，为音视频行业提供更完善易用的解决方案。

- 本文档主要介绍OSS MEDIA C SDK的安装和使用，适用于OSS MEDIA C SDK版本2.0.1。
- 并且假设您已经开通了阿里云OSS 服务，并创建了AccessKeyId 和AccessKeySecret。
- 如果您还没有开通或者还不了解阿里云OSS服务，请登录OSS产品主页了解。
- 如果还没有创建AccessKeyId和AccessKeySecret，请到阿里云Access Key管理创建Access Key。

SDK下载

Linux（2016-03-02）V2.0.2

- SDK包： aliyun-media-c-sdk-2.0.2.tar.gz
- 源代码： GitHub
- 版本迭代： Releases

注:

- OSS MEDIA C SDK不支持Windows。

兼容性

- 对于1.x.x 系列SDK，由于OSS C SDK中list相关的接口发生变化后不兼容，其他接口兼容。

安装

版本依赖

Linux

- OSS C SDK = 3.x.x

Windows

- 不支持

Linux环境安装

- 先安装OSS C SDK，安装步骤参考：安装C SDK
- 从这里下载SDK)包或者下载源代码，应该包括src，sample，test三个目录和CMakeList.txt文件。

安装到系统目录

如果OSS C SDK和其依赖都是安装在系统目录下(/usr/local/或/usr/)，且希望OSS MEDIA C SDK也安装到系统目录下，执行下列命令编译安装：

```
cmake .
make
make install
```

上面命令执行成功后，OSS MEDIA C SDK会自动安装到/usr/local/下面

安装到自定义目录（依赖包安装到系统目录）

如果OSS C SDK和其依赖都是安装到系统目录下(/usr/local/或/usr/)，但希望OSS MEDIA C SDK安装到自定义目录，比如/home/user/aliyun/oss/install/，执行下列命令编译安装：

```
cmake . -DCMAKE_INSTALL_PREFIX=/home/user/aliyun/oss/install/usr/local/
make
make install
```

上面命令执行成功后，OSS MEDIA C SDK会自动安装到
/home/user/aliyun/oss/install/usr/local/下面

安装到自定义目录（依赖包安装在自定义目录）

如果OSS C SDK或某些依赖包安装到了自定义目录，此时编译OSS MEDIA C SDK时默认是找不到这些包的头文件和库文件，所以需要在执行cmake时指定路径，比如已经将OSS C SDK安装到了/home/user/aliyun/oss/install/目录，则执行下列命令编译安装：

```
cmake . -DCMAKE_INSTALL_PREFIX=/home/user/aliyun/oss/install/usr/local/ -
DOSS_C_SDK_INCLUDE_DIR=/home/user/aliyun/oss/install/usr/local/include/ -
DOSS_C_SDK_LIBRARY=/home/user/aliyun/oss/install/usr/local/lib/liboss_c_sdk.so
make
make install
```

上面命令执行成功后，OSS MEDIA C SDK会自动安装到
/home/user/aliyun/oss/install/usr/local/下面

- 其他依赖包相关参数名称
:APR_UTIL_LIBRARY，APR_LIBRARY，CURL_LIBRARY，APR_INCLUDEDIRS，APU_INCLUDEDIR

S，OSS_C_SDK_INCLUDE_DIR，CURL_INCLUDEDIRS等

仅编译安装客户端SDK

默认是同时编译安装客户端和服务端的sdk的，如果仅需要编译安装客户端的SDK，则执行下列命令编译安装

```
 cmake . -DONLY_BUILD_CLIENT=ON
make
make install
```

如果仅需要编译安装服务端，将ONLY_BUILD_CLIENT修改为ONLY_BUILD_SERVER即可。

- 只有同时编译客户端和服务端时才会编译测试用例

其他编译安装方式和问题

编译模式：目前支持四种，分别是Debug，Release，MinSizeRef，RelWithDebInfo，指定使用某种编译类型，使用参数-DCMAKE_BUILD_TYPE，比如使用Debug模式编译，则在cmake是增加参数-DCMAKE_BUILD_TYPE=Debug：cmake . -DCMAKE_BUILD_TYPE=Debug，默认是Release模式。

- Debug：没有做任何代码优化，支持gdb，一般用来调试程序
- Release：使用了更高级别的优化，一般适用于生产环境
- MinSizeRef：生成最小大小的库文件，一般用于嵌入式环境
- RelWithDebInfo：使用了更高级的优化，但附带了调试信息，一般也用于生产环境

执行cmake时出现”Targets may link only to libraries. CMake is dropping the item”的warnning，原因是指定的library路径不对，library路径应该指定到*.so，比如/path/to/xxx.so。

- 如果需要使用OSS C SDK的静态库，则在执行cmake时指定-DOSS_C_SDK_LIBRARY=/home/user/aliyun/oss/install/usr/local/lib/liboss_c_sdk_static.a即可。其他库类似。
- 执行cmake时出现”CMake Error: The following variables are used in this project, but they are set to NOTFOUND.”，原因是相应的库无法从默认路径中找到，需要用户指定，参考<安装到自定义目录>

初始化

确定Endpoint

Endpoint是阿里云OSS服务在各个区域的域名地址，目前支持两种形式。

Endpoint类型	解释
OSS区域地址	使用OSS Bucket所在区域地址，各个区域Endpoint参考这里
用户自定义域名	用户自定义域名，且CNAME指向OSS域名

OSS区域地址

使用OSS Bucket所在区域地址，Endpoint查询可以有下面两种方式：

- 查询Endpoint与区域对应关系详情，可以参考：点击查看。
- 您可以登录 阿里云OSS控制台，进入Bucket概览页，Bucket域名的后缀部分：如bucket-1.oss-cn-hangzhou.aliyuncs.com的oss-cn-hangzhou.aliyuncs.com部分为该Bucket的外网Endpoint。

配置密钥

要接入阿里云OSS，您需要拥有一个有效的 Access Key(包括AccessKeyId和AccessKeySecret)用来进行签名认证。可以通过如下步骤获得：

- 注册阿里云账号
- 申请AccessKey

使用RAM和STS服务

如果您的产品需要从设备端（比如手机、平板电脑、摄像机等）上传、下载文件，设备端应该只获取到临时授权，而非永久授权。这时候，需要用到RAM和STS服务。

1. 前往RAM，开通RAM服务。
2. 创建一个角色，可以给这个角色授予AliyunOSSFullAccess和AliyunSTSAssumeRoleAccess等权限。
3. 创建成功后，在角色详情里面有一个Arn，类似于acs:ram:xxxx:role/yyyy，这个就是role_arn，在后续获取临时token时需要使用。

初始化

- 客户端的初始化参考：客户端操作
- 服务端的初始化参考：服务端操作

客户端

OSS MEDIA C SDK分为客户端，服务端和HLS三部分，下面主要介绍客户端的相关操作，其他的操作请访问后面章节

接口

客户端相关的操作接口都位于oss_media_file.h中，目前提供的接口有：

- oss_media_file_open
- oss_media_file_stat
- oss_media_file_tell
- oss_media_file_seek
- oss_media_file_read
- oss_media_file_write
- oss_media_file_close

下面会详细介绍各个接口的功能和注意事项

基础结构体介绍

```
/**
* OSS MEDIA FILE的元数据，包括文件长度，位置和类型
*/
typedef struct {
int64_t length;
int64_t pos;
char *type;
} oss_media_file_stat_t;


/**
* OSS MEDIA FILE的属性信息
*/
typedef struct oss_media_file_s {
void *ipc;
char *endpoint;
int8_t is_cname;
char *bucket_name;
char *object_key;
char *access_key_id;
char *access_key_secret;
char *token;
char *mode;
oss_media_file_stat_t _stat;

time_t expiration;
auth_fn_t auth_func;
} oss_media_file_t;
```

注：

- type，文件类型, Normal或者Appendable
- ipc，用于设置设备唯一标识，根据用户的需要可以在auth func中使用，如果用不到可以忽略。
- endpoint，比如oss-cn-hangzhou.aliyuncs.com。
- is_cname，是否使用了CNAME
- bucket_name，OSS上存储空间的名称
- object_key，OSS上文件的名称
- access_key_id，阿里云提供的访问控制的access key id，这里有两种使用方式，第一种，使用主账号或者子账号的永久access key id，此时后面的token需要设置为NULL，第二种使用通过get_token获取到的临时access key id。
- access_key_secret，阿里云提供的访问控制的access key secret，这里也有两种使用方式，第一种，使用主账号或者子账号的永久access key secret，此时后面的token需要设置为NULL，第二种使用通过get_token获取到的临时access key secret。access_key_id和access_key_secret必须同时使用同种方式。
- token，如果是使用主账号或者子账号的永久access key id和access key secret，这里需要设置为NULL。如果是终端设备上传下载资源，此时需要应用服务器给终端设备提供临时的访问权限，可以通过服务端的get_token获取到临时的access_key_id，access_key_secret和token。如果用户设置了token不为NULL，则会认为是使用了临时access key id，临时access key secrety和临时token，所以，如果不想使用临时token，请将其设置为NULL。
- expiration，授权失效的时间，首次授权后，后续超过失效时间后才会再次授权
- auth，授权函数，用户需要实现一个函数，在这个函数内部为host，bucket，token等赋值
- mode，读写模式

初始化

```
/**
* @brief 初始化oss media
* @note 在程序开始的时候应该首先调用此接口，初始化OSS MEDIA C SDK
* @return:
* 返回0时表示成功
* 否则, 表示出现了错误，可能导致失败的原因包括：内存不足，apr、curl版本太低等
*/
int oss_media_init(aos_log_level_e log_level);
```

注：

- 示例代码参考：GitHub

销毁

```
/**
* @brief 销毁oss meida
```

```
* @note 在程序结束的时候应该最后调用此接口，销毁OSS MEDIA C SDK
*/
void oss_media_destroy();
```

注：

- 示例代码参考：GitHub

打开文件

```
/**
* @brief 打开一个oss文件
* @param[in] bucket_name oss上存储文件的存储空间名称
* @param[in] object_key oss上的文件名称
* @param[in] mode:
* 'r': 读模式
* 'w': 覆盖写模式
* 'a': 追加写模式
* notes: 不允许组合使用
* @param[in] auth_func 授权函数，设置access_key_id/access_key_secret等
* @return:
* 返回非NULL时成功，否则失败
*/
oss_media_file_t* oss_media_file_open(char *bucket_name,
char *object_key,
char *mode,
auth_fn_t auth_func);
```

注：

- mode，支持只读、覆盖写、追加写三种模式，**不支持组合模式。**读模式，打开文件后，一次读取部分文件内容，完整内容一般需要多次读取；覆盖写，打开文件后，一次写入完整的文件内容，**可以多次写入，但是最后一次写会覆盖的前面写入的内容**；追加写，打开文件后，可以多次追加写文件。
- 示例代码参考：GitHub

关闭文件

```
/**
* @brief 关闭oss文件
*/
void oss_media_file_close(oss_media_file_t *file);
```

注：

- 示例代码参考：GitHub

写文件

```
/**
* @brief 写文件到oss上
* @return:
* 成功时返回写入的数据大小，返回-1表示写失败
*/
int64_t oss_media_file_write(oss_media_file_t *file, const void *buf, int64_t nbyte);
```

示例程序：

```
/* 授权函数 */
static void auth_func(oss_media_file_t *file) {
file->endpoint = "your endpoint";
file->is_cname = 0;
file->access_key_id = "阿里云提供的access key id或者临时access key id";
file->access_key_secret = "阿里云提供的access key secret或者临时access key secret";
file->token = "通过get_token接口获取到得临时token";

/* 本次授权的有效时间 */
file->expiration = time(NULL) + 300;
}

static void write_file() {
oss_clean(g_filename);

int64_t write_size;
oss_media_file_t *file;
char *bucket_name = "<your bucket name>";
char *key = "<your object key>";
char *content = "aliyun oss media c sdk";

/* 打开一个文件 */
file = oss_media_file_open(bucket_name, key, "w", auth_func);
if (!file) {
printf("open media file failed\n");
return;
}

/* 写文件 */
write_size = oss_media_file_write(file, content, strlen(content));
if (-1 != write_size) {
printf("write %" PRId64 " bytes succeeded\n", write_size);
} else {
oss_media_file_close(file);
printf("write failed\n");
return;
}

/* 关闭文件，释放资源 */
oss_media_file_close(file);
```

```
}
```

注：

- 示例中，打开文件的模式为覆盖写("w")，如果多次写入，后一次写会覆盖前一次写；如果需要追加写，打开文件的模式请指定为追加写("a")；。
- 示例代码参考：GitHub

读文件

```
/**
* @brief 读取固定数目nbyte的数据
* @note buf的大小应该大于等于nbyte + 1
* @return:
* 返回0时表示成功
* 否则, 返回-1时表示出现了错误，可能导致失败的原因包括：不是以读模式打开的文件，无法连接OSS，无权限读OSS等
*/
int64_t oss_media_file_read(oss_media_file_t *file, void *buf, int64_t nbyte);
```

示例程序：

```
    int ntotal, nread, nbuf = 16;
char buf[nbuf];
char *content;
char *bucket_name = "<your bucket name>";
char *key = "<your object key>";

oss_media_file_t *file;

/* 打开文件 */
file = oss_media_file_open(bucket_name, key, "r", auth_func);
if (!file) {
printf("open media file failed\n");
return;
}

/* 读文件 */
content = malloc(stat.length + 1);
ntotal = 0;
while ((nread = oss_media_file_read(file, buf, nbuf)) > 0) {
memcpy(content + ntotal, buf, nread);
ntotal += nread;
}
content[ntotal] = '\0';

/* 关闭文件 */
oss_media_file_close(file);
free(content);

printf("oss media c sdk read object succeeded\n");
}
```

注：

- 示例代码参考：GitHub

管理文件

OSS MEDIA FILE也支持tell，seek和stat操作

```
/**
* @brief 获取当前OSS MEDIA FILE的位置
* @return:
* 返回0时表示成功
* 否则, 返回-1时表示出现了错误，可能导致失败的原因包括：不是以读模式打开的文件，无法连接OSS，无权限读OSS等
*/
int64_t oss_media_file_tell(oss_media_file_t *file);

/**
* @brief 设置OSS MEDIA FILE的指针到指定位置
* @return:
* 返回0时表示成功
* 否则, 返回-1时表示出现了错误，可能导致失败的原因包括：不是以读模式打开的文件，无法连接OSS，无权限读OSS等
*/
int64_t oss_media_file_seek(oss_media_file_t *file, int64_t offset);

/**
* @brief 获取OSS MEDIA FILE的元数据
* @return:
* 返回0时表示成功
* 否则, 返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限读OSS等
*/
int oss_media_file_stat(oss_media_file_t *file, oss_media_file_stat_t *stat);
```

示例程序：

```
static void seek_tell_stat_file() {
int ntotal, nread, nbuf = 16;
char buf[nbuf];
char *bucket_name = "<your bucket name>";
char *key = "<your object key>";

oss_media_file_t *file;

/* 打开文件 */
file = oss_media_file_open(bucket_name, key, "r", auth_func);
if (!file) {
printf("open media file failed\n");
return;
}

/* 获取文件meta信息 */
oss_media_file_stat_t stat;
if (0 != oss_media_file_stat(file, &stat)) {
```

```
oss_media_file_close(file);
oss_media_file_free(file);

printf("stat media file[%s] failed\n", file->object_key);
return;
}

printf("file [name=%s, length=%ld, type=%s]",
file->object_key, stat.length, stat.type);

/* tell */
printf("file [position=%" PRId64 "]", oss_media_file_tell(file));

/* seek */
oss_media_file_seek(file, stat.length / 2);

/* 关闭文件 */
oss_media_file_close(file);

printf("oss media c sdk seek object succeeded\n");
}
```

注：

- 示例代码参考：GitHub

服务端

OSS MEDIA C SDK分为客户端，服务端和HLS三部分，下面主要介绍服务端的相关操作，其他的操作请访问后面章节

接口

服务端相关的操作接口都位于oss_media.h中，目前提供的接口有：

- oss_media_create_bucket
- oss_media_delete_bucket
- oss_media_create_bucket_lifecycle
- oss_media_get_bucket_lifecycle
- oss_media_delete_bucket_lifecycle
- oss_media_delete_file
- oss_media_list_files
- oss_media_get_token
- oss_media_get_token_from_policy

下面会详细介绍各个接口的功能和注意事项

基础结构体介绍

```
typedef struct oss_media_config_s {
    char *endpoint;
    int is_cname;
    char *access_key_id;
    char *access_key_secret;
    char *role_arn;
} oss_media_config_t;

typedef struct oss_media_files_s {
    char *path;
    char *marker;
    int  max_size;

    char *next_marker;
    int  size;
    char **file_names;
} oss_media_files_t;
```

注：

- endpoint，比如oss-cn-hangzhou.aliyuncs.com。
- is_cname，是否使用了CNAME
- access_key_id，阿里云提供的访问控制的access key id
- access_key_secret，阿里云提供的访问控制的access key secret
- role_arn，阿里云访问控制中创建的角色的Arn，具体位置在：访问控制RAM控制台 -> 角色管理 -> 点击相应角色名称 -> 基本信息 -> Arn，格式类似于acs:ram::xxxxxx:role/yyyy。如果还没有角色，需要创建一个新的角色，并赋予AliyunOSSFullAccess和AliyunSTSAssumeRoleAccess等权限。
- marker，设定结果从marker之后按字母排序的第一个开始返回。
- max_size，设定返回的最大数，取值不能大于1000。
- next_marker，下次执行时开始的位置

初始化

```
/**
 *  @brief  初始化oss media
 *  @note   在程序开始的时候应该首先调用此接口，初始化OSS MEDIA C SDK
 *  @return:
 *      返回0时表示成功
 *      否则, 表示出现了错误，可能导致失败的原因包括：内存不足，apr、curl版本太低等
 */
int oss_media_init();
```

注：

- 示例代码参考：GitHub

销毁

```
/**
 * @brief 销毁oss meida
 * @note 在程序结束的时候应该最后调用此接口，销毁OSS MEDIA C SDK
 */
void oss_media_destroy();
```

注：

- 示例代码参考：GitHub

创建存储空间

```
/**
 * @brief 创建新的存储空间
 * @paramp[in] oss_media_acl_t
 *    OSS_ACL_PRIVATE        私有读写
 *    OSS_ACL_PUBLIC_READ    公共读，私有写
 *    OSS_ACL_PUBLIC_READ_WRITE 公共读写
 * @return:
 *     返回0时表示成功
 *     否则, 返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限等
 */
int oss_media_create_bucket(oss_media_config_t *config, const char *bucket_name, oss_media_acl_t acl);
```

示例程序：

```
static void init_media_config(oss_media_config_t *config) {
    config->endpoint = "your endpoint";
    config->access_key_id = "阿里云提供的access key id";
    config->access_key_secret = "阿里云提供的access key secret";
    config->role_arn = "阿里云访问控制RAM提供的role arn";
    config->is_cname = 0;
}

void create_bucket() {
    int ret;
    char *bucket_name;
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>";
    init_media_config(&config);
```

```
    /* 创建存储空间 */
    ret = oss_media_create_bucket(&config, bucket_name, OSS_ACL_PRIVATE);

    if (0 == ret) {
        printf("create bucket[%s] succeeded.\n", bucket_name);
    } else {
        printf("create bucket[%s] failed.\n", bucket_name);
    }
}
```

注：

- 示例代码参考：GitHub

删除存储空间

```
/*
 *  @brief  删除存储空间
 *  @return:
 *      返回0时表示成功
 *      否则, 返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限等
 */
int oss_media_delete_bucket(oss_media_config_t *config, const char *bucket_name);
```

示例程序：

```
void delete_bucket() {
    int ret;
    char *bucket_name;
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>";
    init_media_config(&config);

    /* 删除存储空间 */
    ret = oss_media_delete_bucket(&config, bucket_name);

    if (0 == ret) {
        printf("delete bucket[%s] succeeded.\n", bucket_name);
    } else {
        printf("delete bucket[%s] failed.\n", bucket_name);
    }
}
```

注：

- 示例代码参考：GitHub

创建存储生命周期规则

```
/**
 * @brief 创建存储空间生命周期规则
 * @note 这些规则可以控制文件的自动删除时间
 * @return:
 *     返回0时表示成功
 *     否则, 返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限等
 */
int oss_media_create_bucket_lifecycle(oss_media_config_t *config, const char *bucket_name,
oss_media_lifecycle_rules_t *rules);
```

示例程序：

```
void create_bucket_lifecycle() {
    int ret;
    char *bucket_name;
    oss_media_lifecycle_rules_t *rules;
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>";
    init_media_config(&config);

    /* 创建生命周期规则 */
    rules = oss_media_create_lifecycle_rules(2);
    oss_media_lifecycle_rule_t rule1;
    rule1.name = "example-1";
    rule1.path = "/example/1";
    rule1.status = "Enabled";
    rule1.days = 1;
    oss_media_lifecycle_rule_t rule2;
    rule2.name = "example-2";
    rule2.path = "/example/2";
    rule2.status = "Disabled";
    rule2.days = 2;

    rules->rules[0] = &rule1;
    rules->rules[1] = &rule2;

    /* 设置存储空间的生命周期规则 */
    ret = oss_media_create_bucket_lifecycle(&config,
            bucket_name, rules);

    if (0 == ret) {
        printf("create bucket[%s] lifecycle succeeded.\n", bucket_name);
    } else {
        printf("create bucket[%s] lifecycle failed.\n", bucket_name);
    }

    /* 释放资源 */
    oss_media_free_lifecycle_rules(rules);
}
```

注：

- 示例代码参考：GitHub

获取存储空间的生命周期规则

```
/**
 *  @brief  获取存储空间的生命周期规则
 *  @return:
 *      返回0时表示成功
 *      否则, 返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS等
 */
int oss_media_get_bucket_lifecycle(oss_media_config_t *config, const char *bucket_name,
oss_media_lifecycle_rules_t *rules);
```

示例程序：

```
void get_bucket_lifecycle() {
    int ret, i;
    char *bucket_name;
    oss_media_lifecycle_rules_t *rules;
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>";
    init_media_config(&config);

    /* 获取生命周期规则 */
    rules = oss_media_create_lifecycle_rules(0);
    ret = oss_media_get_bucket_lifecycle(&config, bucket_name, rules);

    if (0 == ret) {
        printf("get bucket[%s] lifecycle succeeded.\n", bucket_name);
    } else {
        printf("get bucket[%s] lifecycle failed.\n", bucket_name);
    }

    for (i = 0; i < rules->size; i++) {
        printf(">>>> rule: [name:%s, path:%s, status=%s, days=%d]\n",
               rules->rules[i]->name, rules->rules[i]->path,
               rules->rules[i]->status, rules->rules[i]->days);
    }

    /* 释放资源 */
    oss_media_free_lifecycle_rules(rules);
}
```

注：

- 示例代码参考：GitHub

删除存储空间的生命周期规则

```
/**
 *  @brief  删除存储空间的生命周期规则
 *  @return:
 *      返回0时表示成功
 *      否则, 返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限等
 */
int oss_media_delete_bucket_lifecycle(oss_media_config_t *config, const char *bucket_name);
```

示例程序：

```
void delete_bucket_lifecycle()
{
    int ret;
    char *bucket_name;
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>";
    init_media_config(&config);

    /* 删除生命周期规则 */
    ret = oss_media_delete_bucket_lifecycle(&config, bucket_name);

    if (0 == ret) {
        printf("delete bucket[%s] lifecycle succeeded.\n", bucket_name);
    } else {
        printf("delete bucket[%s] lifecycle failed.\n", bucket_name);
    }
}
```

注：

- 示例代码参考：GitHub

删除文件

```
/**
 *  @brief  删除存储空间中特定的文件
 *  @return:
 *      返回0时表示成功
 *      否则, 返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限等
 */
int oss_media_delete_file(oss_media_config_t *config, const char *bucket_name, const char *key);
```

示例程序：

```
void delete_file() {
    int ret;
    oss_media_config_t config;
    char *file;
    char *bucket_name;

    /* 初始化变量 */
    file = "oss_media_file";
    bucket_name = "<your bucket name>";
    init_media_config(&config);

    /* 删除文件 */
    ret = oss_media_delete_file(&config, bucket_name, file);

    if (0 == ret) {
        printf("delete file[%s] succeeded.\n", file);
    } else {
        printf("delete file[%s] lifecycle failed.\n", file);
    }
}
```

注：

- 示例代码参考：GitHub

列出文件

```
/**
 *  @brief  列出特定存储空间内的文件
 *  @return:
 *      返回0时表示成功
 *      否则, 返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限等
 */
int oss_media_list_files(oss_media_config_t *config, const char *bucket_name, oss_media_files_t *files);
```

示例程序：

```
void list_files() {
    int ret, i;
    char *bucket_name;
    oss_media_files_t *files;
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>>";
    init_media_config(&config);

    files = oss_media_create_files();
    files->max_size = 50;

    /* 列举文件 */
```

```
    ret = oss_media_list_files(&config, bucket_name, files);

    if (0 == ret) {
        printf("list files succeeded.\n");
    } else {
        printf("list files lifecycle failed.\n");
    }

    for (i = 0; i < files->size; i++) {
        printf(">>>>file name: %s\n", files->file_names[i]);
    }

    /* 释放资源 */
    oss_media_free_files(files);
}
```

注：

- 示例代码参考：GitHub

获取临时token

```
/*
 *  @brief  获取临时token
 *  @param[in]:
      mode:
 *      'r': 读模式
 *      'w': 覆盖写模式
 *      'a': 追加写模式
 *    expiration: 临时token的有效时间，最小15分钟，最长1小时，单位秒
 *  @return:
 *      返回0时表示成功
 *      否则, 返回-1时表示出现了错误，可能导致失败的原因包括：无法连接阿里云STS服务，没有开通RAM、STS，没有创建
Role，参数不合法等
 */
int oss_media_get_token(oss_media_config_t *config,
                        const char *bucket_name,
                        const char *path,
                        const char *mode,
                        int64_t expiration,
                        oss_media_token_t *token);
```

示例程序：

```
void get_token() {
    int ret;
    char *bucket_name;
    oss_media_token_t token;
    oss_media_config_t config;

    /* 初始化变量 */
```

```
    bucket_name = "<your bucket name>";
    init_media_config(&config);

    /* 获取临时token */
    ret = oss_media_get_token(&config, bucket_name, "/*", "rwa",
                              3600, &token);

    if (0 == ret) {
        printf("get token succeeded, access_key_id=%s, access_key_secret=%s, token=%s\n",
               token.tmpAccessKeyId, token.tmpAccessKeySecret, token.securityToken);
    } else {
        printf("get token failed.\n");
    }
}
```

注：

- 示例代码参考：GitHub

通过自定义policy获取token

```
/**
 *  @brief  通过指定自定义的policy获取token
 *  @param[in]:
 *      expiration:  临时token的有效时间，最小15分钟，最长1小时，单位秒
 *  @return:
 *      返回0时表示成功
 *      否则, 返回-1时表示出现了错误，可能导致失败的原因包括：无法连接阿里云STS服务，没有开通RAM、STS，没有创建Role，参数不合法等
 */
int oss_media_get_token_from_policy(oss_media_config_t *config,
                                    const char *policy,
                                    int64_t expiration,
                                    oss_media_token_t *token);
```

示例程序：

```
void get_token_from_policy() {
    int ret;
    oss_media_token_t token;
    char *policy;
    oss_media_config_t config;

    /* 初始化变量 */
    init_media_config(&config);

    /* 设置自定义policy */
    policy = "{\"Version\":\"1\",\"Statement\":[{\"Effect\":\"Allow\", "
             "\"Action\":\"*\", \"Resource\":\"*\"}]}";

    /* 获取token */
```

```
    ret = oss_media_get_token_from_policy(&config, policy, 3600, &token);

    if (0 == ret) {
        printf("get token succeeded, access_key_id=%s, access_key_secret=%s, token=%s\n",
               token.tmpAccessKeyId, token.tmpAccessKeySecret, token.securityToken);
    } else {
        printf("get token failed.\n");
    }
}
```

注：

- 示例代码参考：GitHub

HLS基础接口

OSS MEDIA C SDK 客户端部分支持将接收到的H.264、AAC格式封装为TS、M3U8格式，然后写到OSS上，用户通过对应的m3u8地址就可以欣赏视频音频了。

接口

HLS相关基础接口都位于oss_media_hls.h中，目前提供的接口有：

- oss_media_hls_open
- oss_media_hls_write_frame
- oss_media_hls_begin_m3u8
- oss_media_hls_write_m3u8
- oss_media_hls_end_m3u8
- oss_media_hls_flush
- oss_media_hls_close

下面详细介绍各个接口的功能和注意事项

基础结构体介绍

```
/**
 *  OSS MEDIA HLS FRAME的元数据
 */
typedef struct oss_media_hls_frame_s {
    stream_type_t stream_type;
    frame_type_t frame_type;
    uint64_t pts;
```

```
    uint64_t dts;
    uint32_t continuity_counter;
    uint8_t  key:1;
    uint8_t *pos;
    uint8_t *end;
} oss_media_hls_frame_t;

/**
 *  OSS MEDIA HLS的描述信息
 */
typedef struct oss_media_hls_options_s {
    uint16_t video_pid;
    uint16_t audio_pid;
    uint32_t hls_delay_ms;
    uint8_t encrypt:1;
    char    key[OSS_MEDIA_HLS_ENCRYPT_KEY_SIZE];
    file_handler_fn_t handler_func;
    uint16_t pat_interval_frame_count;
} oss_media_hls_options_t;

/**
 *  OSS MEDIA HLS FILE的描述信息
 */
typedef struct oss_media_hls_file_s {
    oss_media_file_t *file;
    oss_media_hls_buf_t *buffer;
    oss_media_hls_options_t options;
    int64_t frame_count;
} oss_media_hls_file_t;
```

注：

- stream_type，流类型, 目前支持st_h264和st_aac两种
- frame_type，帧类型，目前支持ft_non_idr，ft_idr，ft_sei，ft_sps，ft_pps，ft_aud等
- pts，显示时间戳
- dts，解码时间戳
- continuity_counter，递增计数器，从0-15，起始值不一定取0，但必须是连续的
- key，是否是关键帧
- pos，当前帧数据的起始位置(含)
- end，当前帧数据的结束位置(不含)
- video_pid，视频的pid
- audio_pid，音频的pid
- hls_delay_ms，显示延迟毫秒数
- encrypt，是否使用AES-128加密，目前暂不支持
- key，使用加密时的秘钥，目前暂不支持
- handler_func，文件操作回调函数
- pat_interval_frame_count，隔多少帧插入一个pat，mpt表

打开HLS文件

```
/**
 * @brief 打开一个OSS HLS文件
 * @param[in] bucket_name oss上存储文件的存储空间名称
 * @param[in] object_key oss上的文件名称
 * @param[in] auth_func 授权函数，设置access_key_id/access_key_secret等
 * @return:
 *     返回非NULL时成功，否则失败
 */
oss_media_hls_file_t* oss_media_hls_open(char *bucket_name, char *object_key, auth_fn_t auth_func);
```

注：

- 示例代码参考：GitHub

关闭HLS文件

```
/**
 * @brief 关闭OSS HLS文件
 */
int oss_media_hls_close(oss_media_hls_file_t *file);
```

注：

- 示例代码参考：GitHub

写HLS文件

```
/**
 * @brief 写H.264或者AAC的一帧数据到oss上
 * @param[in] frame h.264或者aac格式的一帧数据
 * @param[out] file hls file
 * @return:
 *     返回0时表示成功
 *     否则, 表示出现了错误
 */
int oss_media_hls_write_frame(oss_media_hls_frame_t *frame, oss_media_hls_file_t *file);
```

示例程序：

```
static void write_frame(oss_media_hls_file_t *file) {
    oss_media_hls_frame_t frame;
    FILE    *file_h264;
    uint8_t *buf_h264;
    int     len_h264, i;
    int     cur_pos = -1;
    int     last_pos = -1;
```

```
    int     video_frame_rate = 30;
    int     max_size = 10 * 1024 * 1024;
    char    *h264_file_name = "/path/to/example.h264";

    /* 读取H.264文件 */
    buf_h264 = calloc(max_size, 1);
    file_h264 = fopen(h264_file_name, "r");
    len_h264 = fread(buf_h264, 1, max_size, file_h264);

    /* 初始化frame结构体 */
    frame.stream_type = st_h264;
    frame.pts = 0;
    frame.continuity_counter = 1;
    frame.key = 1;

    /* 遍历H.264的数据，抽取出每帧数据，然后写入oss */
    for (i = 0; i < len_h264; i++) {
        /*  判断当前位置是否下一帧数据的开头，也就是当前帧的结尾 */
        if ((buf_h264[i] & 0x0F)==0x00 && buf_h264[i+1]==0x00
            && buf_h264[i+2]==0x00 && buf_h264[i+3]==0x01)
        {
            cur_pos = i;
        }

    /* 如果获取到完整的一帧数据，就调用接口转为HLS格式后写入OSS */
        if (last_pos != -1 && cur_pos > last_pos) {
            frame.pts += 90000 / video_frame_rate;
            frame.dts = frame.pts;

            frame.pos = buf_h264 + last_pos;
            frame.end = buf_h264 + cur_pos;

            oss_media_hls_write_frame(&frame, file);
        }

        last_pos = cur_pos;
    }

    /* 关闭文件，释放资源 */
    fclose(file_h264);
    free(buf_h264);
}
```

注：

- 示例代码参考：GitHub
- 如果H.264的数据中缺少Access Unit Delimiter NALs（00 00 00 01 09 xx），需要添加这个NAL，否则无法在ipad，iphone，safari上播放
- H.264的帧是通过0xX0，0x00，0x00，0x01分隔的；AAC的帧是通过0xFF，0x0X分隔的；
- 当前帧为关键帧时，frame.key需要设置为1

写M3U8文件

```
/**
 *  @brief  写M3U8文件的头部数据
 *  @param[in]  max_duration  TS文件最长持续时间
 *  @param[in]  sequence      TS文件起始编号
 *  @param[out] file          m3u8 file
 *  @return:
 *      返回0时表示成功
 *      否则, 返回-1时表示出现了错误
 */
void oss_media_hls_begin_m3u8(int32_t max_duration,
                              int32_t sequence,
                              oss_media_hls_file_t *file);

/**
 *  @brief  写M3U8文件数据
 *  @param[in]  size          m3u8 item个数
 *  @param[in]  m3u8          m3u8 item的详细数据
 *  @param[out] file          m3u8 file
 *  @return:
 *      返回0时表示成功
 *      否则, 返回-1时表示出现了错误
 */
int oss_media_hls_write_m3u8(int size,
                             oss_media_hls_m3u8_info_t m3u8[],
                             oss_media_hls_file_t *file);

/**
 *  @brief  写M3U8文件的结束符等数据
 *  @param[out] file          m3u8 file
 */
void oss_media_hls_end_m3u8(oss_media_hls_file_t *file);
```

示例程序：

```
static void write_m3u8() {
    char *bucket_name;
    char *key;
    oss_media_hls_file_t *file;

    bucket_name = "<your bucket name>";
    key = "<your m3u8 file name>";

    /* 打开一个HLS文件用来写M3U8格式的数据，文件名必须以.m3u8结尾 */
    file = oss_media_hls_open(bucket_name, key, auth_func);
    if (file == NULL) {
        printf("open m3u8 file[%s] failed.", key);
        return;
    }

    /* 构造3个ts格式文件的信息 */
    oss_media_hls_m3u8_info_t m3u8[3];
    m3u8[0].duration = 9;
    memcpy(m3u8[0].url, "video-0.ts", strlen("video-0.ts"));
    m3u8[1].duration = 10;
```

```
    memcpy(m3u8[1].url, "video-1.ts", strlen("video-1.ts"));

    /* 写入M3U8文件
    oss_media_hls_begin_m3u8(10, 0, file);
    oss_media_hls_write_m3u8(2, m3u8, file);
    oss_media_hls_end_m3u8(file);

    /* 关闭HLS文件 */
    oss_media_hls_close(file);

    printf("write m3u8 to oss file succeeded\n");
}
```

注：

- 目前使用的M3U8版本是3
- 如果是录播，需要在结束的时候调用oss_media_hls_end_m3u8(file)接口写入结束符，否则可能无法播放；如果是直播，则不能调用此接口
- 示例代码参考：GitHub
- 可以通过示例程序观看效果
- Windows平台可以通过VLC播放器观看，iPhone，iPad，Mac等可以直接使用Safari观看。

HLS封装接口

OSS MEDIA C SDK 客户端部分支持将接收到的H.264、AAC封装为TS、M3U8格式后写入OSS，除了基础接口外，还提供封装好的录播、直播接口。

接口

HLS相关封装接口都位于oss_media_hls_stream.h中，目前提供的接口有：

- oss_media_hls_stream_open
- oss_media_hls_stream_write
- oss_media_hls_stream_close

下面详细介绍各个接口的功能和注意事项

基础结构体介绍

```
/**
* OSS MEDIA HLS STREAM OPTIONS的描述信息
*/
```

```
typedef struct oss_media_hls_stream_options_s {
int8_t is_live;
char *bucket_name;
char *ts_name_prefix;
char *m3u8_name;
int32_t video_frame_rate;
int32_t audio_sample_rate;
int32_t hls_time;
int32_t hls_list_size;
} oss_media_hls_stream_options_t;

/**
* OSS MEDIA HLS STREAM的描述信息
*/
typedef struct oss_media_hls_stream_s {
const oss_media_hls_stream_options_t *options;
oss_media_hls_file_t *ts_file;
oss_media_hls_file_t *m3u8_file;
oss_media_hls_frame_t *video_frame;
oss_media_hls_frame_t *audio_frame;
oss_media_hls_m3u8_info_t *m3u8_infos;
int32_t ts_file_index;
int64_t current_file_begin_pts;
int32_t has_aud;
aos_pool_t *pool;
} oss_media_hls_stream_t;
```

注：

- is_live，是否是直播模式，直播模式的时候M3U8文件里面只最新的几个ts文件信息
- bucket_name，存储HLS文件的存储空间名称
- ts_name_prefix，TS文件名称的前缀
- m3u8_name，M3U8文件名称
- video_frame_rate，视频数据的帧率
- audio_sample_rate，音频数据的采样率
- hls_time，每个ts文件最大持续时间
- hls_list_size，直播模式时在M3U8文件中最多保留的ts文件个数

打开HLS stream文件

```
/**
* @brief 打开一个oss hls文件
* @param[in] auth_func 授权函数，设置access_key_id/access_key_secret等
* @param[in] options 配置信息
* @return:
* 返回非NULL时成功，否则失败
*/
oss_media_hls_stream_t* oss_media_hls_stream_open(auth_fn_t auth_func,
const oss_media_hls_stream_options_t *options);
```

注：

- 示例代码参考：GitHub

关闭HLS stream文件

```
/**
* @brief 关闭HLS stream文件
*/
int oss_media_hls_stream_close(oss_media_hls_stream_t *stream);
```

注：

- 示例代码参考：GitHub

写HLS stream文件

```
/**
* @brief 写入视频和音频数据
* @param[in] video_buf 视频数据
* @param[in] video_len 视频数据的长度，可以为0
* @param[in] audio_buf 音频数据
* @param[in] audio_len 音频数据的长度，可以为0
* @param[in] stream HLS stream
* @return:
* 返回0时表示成功
* 否则, 表示出现了错误
*/
int oss_media_hls_stream_write(uint8_t *video_buf,
uint64_t video_len,
uint8_t *audio_buf,
uint64_t audio_len,
oss_media_hls_stream_t *stream);
```

示例程序：

```
static void write_video_audio_vod() {
int ret;
int max_size = 10 * 1024 * 1024;
FILE *h264_file, *aac_file;
uint8_t *h264_buf, *aac_buf;
int h264_len, aac_len;

oss_media_hls_stream_options_t options;
oss_media_hls_stream_t *stream;

/* 设置HLS stream的参数值 */
options.is_live = 0;
```

```
options.bucket_name = "<your buckete name>";
options.ts_name_prefix = "vod/video_audio/test";
options.m3u8_name = "vod/video_audio/vod.m3u8";
options.video_frame_rate = 30;
options.audio_sample_rate = 24000;
options.hls_time = 5;

/* 打开HLS stream */
stream = oss_media_hls_stream_open(auth_func, &options);
if (stream == NULL) {
printf("open hls stream failed.\n");
return;
}

/* 创建两个buffer用来存储音频和视频数据 */
h264_buf = malloc(max_size);
aac_buf = malloc(max_size);

/* 读取一段视频数据和音频数据，然后调用接口写入OSS */
{
h264_file = fopen("/path/to/video/1.h264", "r");
h264_len = fread(h264_buf, 1, max_size, h264_file);
fclose(h264_file);

aac_file = fopen("/path/to/audio/1.aac", "r");
aac_len = fread(aac_buf, 1, max_size, aac_file);
fclose(aac_file);

ret = oss_media_hls_stream_write(h264_buf, h264_len,
aac_buf, aac_len, stream);
if (ret != 0) {
printf("write vod stream failed.\n");
return;
}
}

/* 再读取一段视频数据和音频数据，然后调用接口写入OSS */
{
h264_file = fopen("/path/to/video/2.h264", "r");
h264_len = fread(h264_buf, 1, max_size, h264_file);
fclose(h264_file);

aac_file = fopen("/path/to/audio/1.aac", "r");
aac_len = fread(aac_buf, 1, max_size, aac_file);
fclose(aac_file);

ret = oss_media_hls_stream_write(h264_buf, h264_len,
aac_buf, aac_len, stream);
if (ret != 0) {
printf("write vod stream failed.\n");
return;
}
}

/* 写完数据后，关闭HLS stream */
ret = oss_media_hls_stream_close(stream);
```

```
if (ret != 0) {
printf("close vod stream failed.\n");
return;
}

/* 释放资源 */
free(h264_buf);
free(aac_buf);

printf("convert H.264 and aac to HLS vod succeeded\n");
}
```

注：

- 目前的录播、直播接口都支持只有视频，只有音频，同时有音视频等。
- 示例代码参考：GitHub
- 目前的录播、直播接口比较初级，用户如果有高级需求，可以模拟这两个接口，使用基础接口自助实现高级定制功能。
- 可以通过示例程序观看效果

使用场景

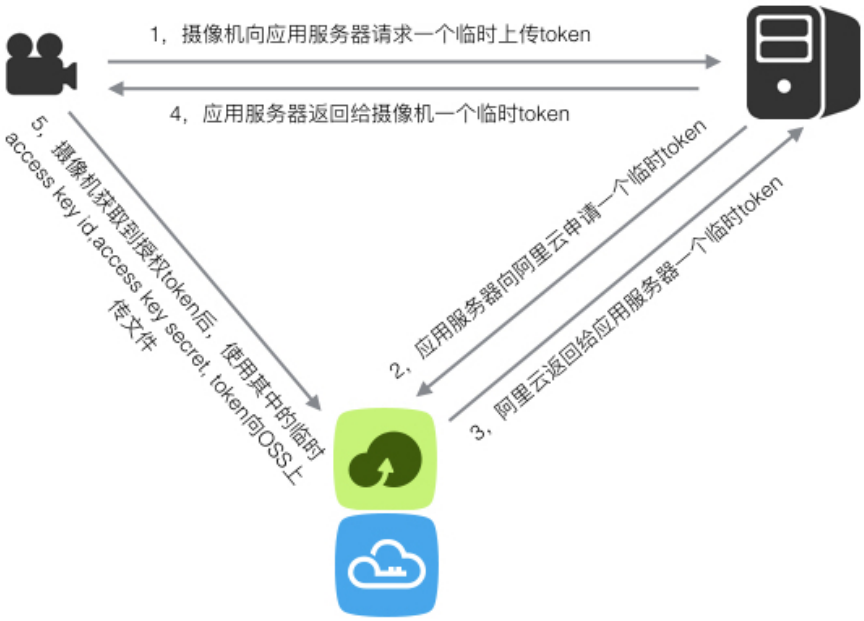
场景

在上两节分别介绍了客户端和服务端的相关操作，接下来我们介绍如何将客户端和服务端连接起来使用，如果您还没有阅读前两节，强烈建议先阅读前两节，然后再阅读本节。

视频监控

在前言里面，介绍了可以方便的用于网络摄像头等设备，这里，会详细介绍一下如何使用。

流程



角色包括三个，网络摄像机，应用服务器，阿里云对象存储服务（OSS），网络摄像机内部使用OSS MEDIA C SDK的client部分，应用服务器内部使用OSS MEDIA C SDK的server部分。

当网络摄像机拍摄了一段视频，需要上传到OSS。流程如下：

1. 首先，网络摄像机向应用服务器发送网络请求：要求获取一个上传视频到OSS的授权。
2. 应用服务器收到请求后，通过检查觉得可以给网络摄像机上传的权限，就通过OSS MEDIA C SDK的get_token接口，向阿里云请求一个在特定有效期有效的，只有上传权限的token。
3. 阿里云接收到应用服务器的获取token请求后，通过检查用户的配置，如果允许，就颁发一个临时token（包括临时access key id，临时access key secret和临时sts token）：只有上传OSS的权限，且在特定时间内有效，然后发送给应用服务器。
4. 应用服务器收到临时token后，转发给刚才要token的网络摄像机。
5. 网络摄像机获取到token后，就可以通过OSS MEDIA C SDK client部分的oss_media_write接 口上传视频文件到OSS了。
6. 还可以在应用服务器上使用C SDK的server部分或者JAVA, C#，Go，Python，Php，Ruby等SDK实现一个HTTP服务，这样其他人就可以在网页上查看，管理各个视频文件。

示例代码

下面是一个简单模拟客户端和服务端操作的示例程序：

```
char* global_temp_access_key_id = NULL;
char* global_temp_access_key_secret = NULL;
char* global_temp_token = NULL;

/* 授权函数 */
static void auth_func(oss_media_file_t *file) {
file->endpoint = "your endpoint";
file->is_cname = 0;
file->access_key_id = global_temp_access_key_id;
```

```
file->access_key_secret = global_temp_access_key_secret;
file->token = global_temp_token;

/* 本次授权的有效时间 */
file->expiration = time(NULL) + 300;
}

/* 模拟服务端发送token给客户端 */
static void send_token_to_client(oss_media_token_t token) {
global_temp_access_key_id = token.tmpAccessKeyId;
global_temp_access_key_secret = token.tmpAccessKeySecret;
global_temp_token = token.securityToken;
}

void get_and_use_token() {
oss_media_token_t token;

/* 服务端逻辑：从阿里云获取到临时token后发送给客户端 */
{
int ret;
char *policy = NULL;
oss_media_config_t config;

policy = "{\n"
"\"Statement\": [\n"
"{"
"\"Action\": \"oss:*\",\n"
"\"Effect\": \"Allow\",\n"
"\"Resource\": \"*\"\n"
"}\n"
"],\n"
"\"Version\": \"1\"\n"
"}\n";

init_media_config(&config);

/* 从阿里云请求一个临时授权token */
ret = oss_media_get_token_from_policy(&config, policy,
17 * 60, &token);

if (ret != 0) {
printf ("Get token failed.");
return;
}

/* 模拟将临时token发送给客户端 */
send_token_to_client(token);
}

/* 客户端逻辑：从服务端获取到临时token后，使用临时token操作文件 */
{
int ret;
int64_t write_size = 0;
oss_media_file_t *file = NULL;
char *content = NULL;
char *bucket_name;
```

```
char *object_key;
oss_media_file_stat_t stat;

content = "hello oss media file\n";
bucket_name = "<your bucket name>";
object_key = "key";

/* 打开文件 */
file = oss_media_file_open(bucket_name, object_key, "w", auth_func);
if (file != NULL) {
printf ("open file failed.");
return;
}

/* 写文件 */
write_size = oss_media_file_write(file, content, strlen(content));
if (write_size != strlen(content)) {
printf ("write file failed.");
return;
}

/* 关闭文件释放资源 */
oss_media_file_close(file);
}
}
```

提示：

- Policy可以从阿里云访问控制RAM > 角色管理 > 点击某个角色 > 基本信息下面的方框中获取。
- 如果不需要精准控制权限，可以使用更简单的oss_media_get_token接口，其中path参数可以是”/*”，mode参数可以是”rwa”。
- 更详细的RAM、STS使用可以参考RAM和STS指南。

常见问题

OSS MEDIA C SDK和OSS C SDK是啥关系？

- OSS MEDIA C SDK依赖于OSS C SDK，OSS MEDIA C SDK中的上传和下载等功能是通过调用OSS C SDK的接口实现的。

OSS MEDIA C SDK是否支持windows？

- 目前还不支持。

是否支持追加写文件？

- 支持，调用oss_media_file_open时使用“a”模式，然后可以通过多次调用oss_media_file_write接口实现追加写。

什么是role arn？如何获取role arn？

- role arn表示的是需要扮演角色的id，由阿里云访问控制RAM提供。可以前往**访问控制RAM** > **角色管理** > 点击已经创建的角色名称 > **基本信息** > **Arn**，值类似于:acs:ram::xxxx:role/yyyyy。如果还没有已创建的角色，需要在角色管理页面创建一个新的用户角色，并赋予AliyunSTSAssumeRoleAccess和其他相应角色，更详细的介绍可以参考：RAM的文档。

如何运行sample？

- 修改sample/config.c文件，增加自己的access key id，access key secret，bucket等值，然后编译后，在bin目录下就会出现sample的可执行文件

报错：error:a timeout was reached

- 检查一下host的值，是否是类似于oss-cn-hangzhou.aliyuncs.com的值。这个是C SDK的一个已知问题，会在后期版本修复。

运行sample时报错：error:Couldn’t resolve host name 和[code=-990, message=HttpIoError]

- 修改sample/config.c文件，配置您自己的参数值，然后重新编译即可。测试也一样。

客户端和服务端的access key id，access key secret，token配置有啥不同和注意点？

- 服务端只需要配置access key id和access key secret，这两个值有两种来源：第一个是主账号的AccessKeys，第二个是主账号生成的子账号的AccessKeys。
- 客户端有两种配置方式，第一种是和服务端一样，只配置主账号或者子账号的access key id，access key secret，第二种是配置access key id，access key secret和token三个值，但这三个值都是服务端通过oss_media_get_token或者oss_media_get_token_from_policy获取到的临时AccessKey和token，有时间期限，超过有效期后，就不能再次使用。

执行sample获取token的时候出现以下错误：http_code=500, error_code=GetSTSTokenError, error_message=Internal Error

- 原因是安装的libcurl不支持HTTPS协议，导致无法访问sts服务。具体过程是机器上没有安装openssl-devel等ssl的开发包，在编译libcurl的时候找不到ssl，libcurl就自动禁止了HTTPS协议，导致编译出来的libcurl库不支持HTTPS，最终访问STS失败。
- 解决办法是先安装openssl-devel等ssl开发包，然后重新安装libcurl。在安装libcurl时，当执行完./configure后，检查最后一行的Protocols里包含了HTTPS，如果包含了，就说明正确了。

历史版本SDK下载

OSS SDK开发包

OSS SDK 开发包

语言	下载详情
Java SDK	点击这里
Python SDK	点击这里
Android SDK	点击这里
iOS SDK	点击这里
.NET SDK	点击这里
JavaScript SDK	点击这里
Node.js SDK	点击这里
PHP SDK	点击这里
C SDK	点击这里
Ruby-SDK	点击这里
Go-SDK	点击这里
Media C SDK	点击这里

Java SDK开发包

Java SDK 开发包

Java SDK最新版本

点击查看

Java SDK开发包(2017-02-22) 版本号 2.5.0

Java SDK下载地址 ： aliyun_java_sdk_20170222.zip

更新日志：

- 添加：支持get bucket stat
- 添加：CName支持IsPurgeCdnCache
- 添加：强制关闭OSSObject的功能
- 修复：断点续传下载不成功不修改本地同名文件
- 修复：BucketInfo添加解析ExtranetEndpoint/IntranetEndpoint
- 修复：修复key为”“时抛IndexOutOfBoundException的异常
- 修复：initiateMultipartUpload特定情况下抛NullPointerException的异常
- 优化：AccessControlList中添加权限直接描述
- 优化：统一UDF/上传回调后返回的Response
- 优化：InvalidResponse异常的描述信息中添加上下文信息

Java SDK开发包(2017-01-09) 版本号 2.4.1

Java SDK下载地址 ： aliyun_java_sdk_20170109.zip

更新日志：

- 修复：GeneratePresignedUrlRequest支持callback
- 修复：GetObjectToFile支持CRC校验
- 修复：PutObject With URL支持CRC校验

Java SDK开发包(2016-11-21) 版本号 2.4.0

Java SDK下载地址 ： aliyun_java_sdk_20161121.zip

更新日志：

- 添加：支持符号链接；
- 添加：镜像支持querystring，支持RangGet,支持slash；
- 添加：跨区域复制支持Prefix和Action；
- 修复：图片服务格式，从setparams改成setProcess，两个格式都能使用；

Java SDK开发包(2016-08-11) 版本号 2.3.0

Java SDK下载地址 ： aliyun_java_sdk_20160811.zip

更新日志：

- 支持RTMP功能；
- 支持CRC64校验功能；
- 支持OSS和IMG合并；
- 支持代理Proxy访问OSS；
- 修复OSSClient.shutdown抛异常的bug；
- 添加分片上传提交时自动对分片列表排序；
- 优化合并测试cases；
- 支持请求级别的超时。

Java SDK开发包(2016-05-10) 版本号 2.2.3

Java SDK下载地址 ：aliyun_java_sdk_20160510.zip

更新日志：

- 支持正常数据操作请求返回ReqestId；
- listMultipartUploads/listParts支持url编码；
- 修复getSimplifiedObjectMeta没有关闭连接的bug；
- 修复doesObjectExist返回InvalidResponse的bug；
- 支持请求超时设置；
- 添加UploadPartRequest/UploadPartCopyRequest的易用性构造函数。

Java SDK开发包(2016-03-01) 版本号 2.2.1

Java SDK下载地址 ：aliyun_java_sdk_20160301.zip

更新日志：

- 支持断点续传并发的上传/下载；
- 支持上传回调(callback)功能；
- 条件重定向添加AliCDN类型。

Java SDK开发包(2016-02-01) 版本号 2.2.0

Java SDK下载地址 ：aliyun_java_sdk_20160201.zip

更新日志：

- 添加cname功能支持；
- 添加跨集群复制功能支持；
- 添加镜像回写配置功能支持；
- 添加设置bucket storage功能支持；
- 添加GetObjectMeta接口获取object的元数据；
- Website功能增强，支持重定向功能；
- 添加GetBucketInfo接口获取bucket的相关信息；

- Lifecycle功能增强，支持Object按照创建时间过期，支持Multipart过期。

Java SDK开发包(2016-01-30) 版本号 2.1.2

Java SDK下载地址 ：aliyun_java_sdk_20160130.zip

更新日志：

- 添加从连接池中获取连接超时接口；
- 修复使用过期连接的bug；
- 修复重试时缓存不足的bug；
- 添加自动识别ip格式的Endpoint的功能。

Java SDK开发包(2015-12-23) 版本号 2.1.0

Java SDK下载地址 ：aliyun_java_sdk_20151223.zip

更新日志：

- 去掉log4j.properties配置文件；
- 添加Put/Get/Delete Bucket Tagging接口。

Java SDK开发包(2015-11-24) 版本号 2.0.7

Java SDK下载地址 ：aliyun_java_sdk_20151124.zip

更新日志:

- 修复SetObjectAcl存在的bug
- 去掉log4j依赖，采用commons-logging

Java SDK开发包(2015-09-22) 版本号 2.0.6

更新日志:

- 添加setObjectAcl/getObjectAcl接口，用于设置、获取指定Object的ACL；
- 添加ClientConfiguration.setLogLevel接口设置全局日志级别，默认将404错误码日志级别设置为INFO；
- 添加ClientConfiguration.setSLDEnabled接口设置是否开启二级域名的访问方式；
- 添加ClientConfiguration.setSupportCname接口设置是否支持Cname作为Endpoint；
- 输出Invalid Response详细错误信息、完善OssException/ClientException错误信息；
- 修复设置Bucket默认ACL为private的bug；
- 修复upload part支持chunked编码的bug，并将输入流包装为可重试输入流；
- 添加webp至mime types列表。

Java SDK开发包(2015-07-10) 版本号 2.0.5

Java SDK下载地址 ：aliyun_java_sdk_20150710.zip

更新日志:

- 添加Append Object支持；
- 添加HTTPS支持；
- DeleteObject、ListObjects接口添加encoding-type参数，允许指定Object名称的编码方式；
- 去除Expires响应头日期格式的强制检查，修复无法解析非GMT日期格式的Expires响应头。

Java SDK开发包(2015-05-29) 版本号 2.0.4

Java SDK下载地址 ：aliyun_java_sdk_20150529.zip

更新日志:

- 添加STS支持；
- 新增测试Demo Jar，具体使用方式可参考demo文件夹下的Readme.txt；
- 修改CreateBucket逻辑，允许创建Bucket同时指定Bucket ACL；
- 修改bucket名称检查规则，允许操作带有下划线且已存在的Bucket，但不允许创建带有下划线的Bucket；
- 优化HTTP连接池，提升并发处理能力。

Java SDK开发包(2015-04-24) 版本号 2.0.3

Java SDK下载地址 ：aliyun_java_sdk_20150424.zip

更新日志:

- 新增deleteObjects接口，批量删除Object列表；
- 新增doesObjectExist接口，判断指定Bucket下的Object是否存在；
- 新增switchCredentials接口，可用于更换已有OSSClient实例的Credentials；
- 新增带有CredentialsProvider（Credentials提供者）的OSSClient构造函数，可对CredentialsProvider进行扩展定制；
- 修复copyObject接口中Source Key包含特殊字符加号时出现的bug；
- 调整OSSException/ClientException异常信息的显示格式；

Java SDK开发包(2015-03-23) 版本号 2.0.2

Java SDK下载地址 ：aliyun_java_sdk_20150323.zip

更新日志:

- 新增GeneratePostPolicy/calculatePostSignature接口，用于生成Post请求的Policy字符串及Post签

名
- 支持Bucket Lifecycle
- 新增基于URL签名的Put/GetObject重载接口
- 支持PutObject、UploadPart以Chunked编码的方式上传数据
- 修复若干Bug

Java SDK开发包(2015-01-15) 版本号 2.0.1

Java SDK下载地址 ：aliyun_java_sdk_20150115.zip

更新日志:

- 支持Cname，允许用户指定哪些是保留域名
- 生成URI时支持用户指定ContentType及ContentMD5
- CopyObject请求不支持Server端加密的问题
- 更改UserAgent的格式
- 扩充Location常量 - 增加部分Samples

Java SDK开发包(2014-11-13)

Java SDK下载地址 ：aliyun_java_sdk_20141113.zip

新增内容： Upload Part Copy 功能 OSS Java SDK源码 示例代码

重要提示： 2.0.0 版本OSS Java SDK移除了2.0.0之前版本中OTS相关代码，调整了包结构，加入了OSS SDK源码和示例代码。 使用2.0.0之前版本开发的程序在使用2.0.0版本需要修改引用的包名称。 包名称 com.aliyun.openservices.* 与 com.aliyun.openservices.oss.* 更换为com.aliyun.oss.*。

Python SDK开发包

Python SDK最新版本

点击查看

Python SDK开发包(2016-04-19) 版本 0.4.6

更新日志:

- oss_api修复某些接口不支持sts_token的bug

Python SDK开发包下载地址:OSS_Python_API_20160419.zip

Python cmd工具使用说明：点击查看

Python SDK开发包(2016-04-13) 版本 0.4.5

更新日志:

- osscmd增加增加object acl相关的接口

Python SDK开发包下载地址:OSS_Python_API_20160413.zip

Python cmd工具使用说明：点击查看

Python SDK开发包(2016-02-23) 版本 0.4.4

更新日志:

- osscmd增加对特殊控制字符的listparts和cancel支持，使用encoding_type选项

Python SDK开发包下载地址:OSS_Python_API_20160223.zip

Python cmd工具使用说明：点击查看

Python SDK开发包(2015-11-20) 版本 0.4.2

更新日志:

- osscmd增加appendfromfile接口支持自动追加文件内容到指定object
- API中添加apengd(追加上传相关接口)

Python SDK开发包下载地址:OSS_Python_API_20151120.zip

Python cmd工具使用说明：点击查看

Python SDK开发包(2015-08-11) 版本 0.4.1

更新日志:

- osscmd支持通过对响应的xml编码接受对包括控制字符的list和delete object。

Python SDK开发包下载地址:OSS_Python_API_20150811.zip

Python cmd工具使用说明：点击查看

Python SDK开发包(2015-07-07) 版本 0.4.0

更新日志:

- osscmd中支持STS功能

Python SDK开发包下载地址:OSS_Python_API_20150707.zip

Python cmd工具使用说明：点击查看

Python SDK开发包(2015-06-24) 版本 0.3.9

更新日志:

- osscmd中添加copylargefile命令，支持大文件复制

Python SDK开发包下载地址:OSS_Python_API_20150624.zip

Python cmd工具使用说明：点击查看

Python SDK开发包(2015-04-13) 版本 0.3.8

更新日志:

- osscmd中修复multiupload指定max_part_num不生效的问题
- oss_api增加对upload_part指定part的md5校验

Python SDK开发包下载地址:oss_python_sdk_20150413.zip

Python cmd工具使用说明：点击查看

Python SDK开发包(2015-01-29) 版本 0.3.7

更新日志:

- oss_api中增加了lifecycle和referer相关的接口。
- osscmd中增加了lifecycle和referer相关的命令。
- osscmd中修复了指定upload_id不生效的问题。

Python SDK开发包下载地址:oss_python_sdk_20150129.zip

Python cmd工具使用说明：点击查看

Python SDK开发包(2014-12-31) 版本 0.3.6

更新日志:

- osscmd的uploadfromdir命令增加了check_point功能，使用—check_point选项设置。
- osscmd的deleteallobject命令增加—force功能，强制删除所有文件。

- osscmd的multipart命令和uploadfromdir、downloadtodir命令增加—thread_num选项，可以调整线程数
- oss_api中增加了根据文件名生成Content-Type的功能。
- osscmd的downloadtodir命令增加—temp_dir选项，支持将下载的文件临时存在指定目录下。
- osscmd增加—check_md5选项，能对上传的文件进行md5检查。

Python SDK开发包下载地址:oss_python_sdk_20141231.zip

快速入门可参考SDK中的README文件

Python SDK 开发包（2014-05-09）

更新日志:

- 修复oss_util中logger初始化错误的bug。
- 优化在某些情况下oss_api中multi_upload_file上传接口，减少网络异常情况下重传的次数。

Python SDK开发包下载地址：oss_python_sdk_20140509.zip

Android SDK开发包

Android SDK 开发包

SDK 版本历史参见链接

iOS SDK开发包

iOS SDK 开发包

iOS SDK版本历史参见链接

PHP SDK开发包

PHP SDK 开发包

PHP SDK最新版本

点击查看

v2.0.5

- aliyun-oss-php-sdk-2.0.5.zip
- aliyun-oss-php-sdk-2.0.5.phar

v2.0.4

- aliyun-oss-php-sdk-2.0.4.zip
- aliyun-oss-php-sdk-2.0.4.phar

PHP SDK 开发包(2015-08-19)

下载地址 ：oss_php_sdk_20150819

更新日志

- 修复了在有response-content-disposition等HTTP Header的时候，下载签名不对的问题。
- 新增了转换响应body的设置，目前支持xml,array,json三种格式。默认为xml。
- 新增copy_upload_part方法。
- 支持sts。
- 调整签名url中$options参数的位置。
- fix read_dir循环遍历的问题。
- 增加了referer和lifecycle相关的接口。增加了upload by file和multipart upload时候的content-md5检查的选项。
- 增加了init_multipart_upload 直接获取string类型的upload。
- 调整了batch_upload_file函数的返回值，从原来的空，变成boolean，成功为true，失败为false。
- 调整了sdk.class.php 中工具函数的位置，放置在util/oss_util.class.php中，如果需要引用，需要增加OSSUtil::， 并引用该文件。

修复的Bug：

- 修复Copy object的过程中无法修改header的问题。
- 修复upload part时候自定义upload的语法错误。
- 修复上传的时候，office2007文件的mimetype无法设置正确的问题。
- 修复batch_upload_file时候，遇到空目录会超时退出的问题。

PHP SDK 开发包(2014-06-25)

下载地址 ：oss_php_sdk_20140625新增功能：

- 加入设置 CORS 功能

PHP SDK V1 开发包（2013-06-25）

PHP SDK开发包（2012-10-10）

此版本主要按新API发布内容，修改生成域名规则。

PHP SDK开发包（2012-08-17）

此版本主要解决 get_sign_url无法设置Expires的问题。

PHP SDK开发包（2012-06-12）

更新日志:

1. 修复了设置hostname的bug。
2. 优化了内部的Exception处理。
3. 支持三级域名，如bucket.storage.aliyun.com
4. 优化了demo程序，使得看起来更加的简洁。

C SDK开发包

C SDK 开发包
OSS C SDK最新版本

点击查看

OSS C SDK开发包（2017-02-22）版本3.4.0

下载地址：

- Linux: aliyun_oss_c_sdk_v3.4.0.tar.gz
- Windows: aliyun_oss_c_sdk_v3.4.0.zip

更新日志：

- 添加：支持并发断点续传上传oss_resumable_upload_file
- 修复：oss_gen_signed_url支持临时用户签名
- 修复：初始化默认不打开fd 2，退出时不关闭fd 2
- 修复：解决key为xxx/./yyy/，./async_test/test报SignatureDoesNotMatch的问题

OSS C SDK开发包（2016-12-22）版本3.3.0

下载地址：

- Linux: aliyun_oss_c_sdk_v3.3.0.tar.gz
- Windows: aliyun_oss_c_sdk_v3.3.0.zip

更新日志：

- 添加：支持代理*Proxy*
- 修复：oss_get_object_to_file先下载到本地临时文件，成功后修改文件名称
- 修复：去除*Visual Studio*编译警告aos_util.c(512) C4146
- 修复：URL上传下载添加CRC校验

OSS C SDK开发包（2016-11-22）版本3.2.1

下载地址：

- Linux: aliyun_oss_c_sdk_v3.2.1.tar.gz
- Windows: aliyun_oss_c_sdk_v3.2.1.zip

更新日志：

- 修复：解决oss_copy_object源文件名不能有URL特殊字符的问题

OSS C SDK开发包（2016-11-14）版本3.2.0

下载地址：

- Linux: aliyun_oss_c_sdk_v3.2.0.tar.gz
- Windows: aliyun_oss_c_sdk_v3.2.0.zip

更新日志：

- 添加：支持上传、下载CRC检验
- 添加：支持上传回调功能
- 添加：支持进度条功能

OSS C SDK开发包（2016-08-11）版本3.1.0

下载地址：

- Linux: aliyun_oss_c_sdk_v3.1.0.tar.gz
- Windows: aliyun_oss_c_sdk_v3.1.0.zip

更新日志：

- 添加：支持RTMP功能
- 修复：支持OSS和IMG合并

OSS C SDK开发包（2016-05-30）版本3.0.0

下载地址：

- Linux: aliyun_oss_c_sdk_v3.0.0.tar.gz
- Windows: aliyun_oss_c_sdk_v3.0.0.zip

更新日志：

- 合并Linux、Windows代码
- 修复分片上传时分片过多会异常的问题
- 提供Windows示例工程
- 优化Linux编译安装方法

OSS C SDK开发包（2016-03-28）版本2.1.0

下载地址：

- Linux: aliyun_oss_c_sdk_linux_v2.1.0.tar.gz
- Windows: aliyun_oss_c_sdk_windows_v2.1.0.zip

更新日志：

- 完善示例程序
- header长度由限制为1K升级为最长8K
- 解决部分单词拼写错误

OSS C SDK开发包（2016-03-06）版本2.0.0

下载地址：

- Linux: aliyun_oss_c_sdk_linux_v2.0.0.tar.gz
- Windows: aliyun_oss_c_sdk_windows_v2.0.0.zip

更新日志：

- complete multipart接口支持修改原有header
- 重构示例程序和组织方式
- 开放params参数，允许用户自定义设置
- 允许params和headers参数为空，简化用户使用和减少用户代码量
- 支持https
- 支持ip
- 新增部分测试
- 新增oss_put_bucket_acl接口
- 新增目录相关示例
- 新增signed url相关示例
- 完善接口注释
- 删除无用的port配置参数
- 调整oss_init_multipart_upload接口参数顺序
- 优化配置参数名称，使其与官方网站保持一致
- 解决endpoint不能含有http等前缀的问题
- 解决用户无法设置content-type的问题
- 解决无法自动根据file name和key设置content-type的问题
- 解决list upload parts为空时coredump的问题
- 解决oss_upload_file接口在断点续传时可能会coredump的问题
- 解决部分单词拼写错误
- 解决所有警告
- 解决部分头文件宏保护无效的问题
- 解决oss_head_object_by_url接口不生效的问题
- 提高易用性，降低用户使用门槛
- 支持Visual C++ 2008

OSS C SDK开发包（2015-12-17）版本1.0.0

下载地址：

- Linux: aliyun_oss_c_sdk_linux_v1.0.0.tar.gz
- Windows: aliyun_oss_c_sdk_windows_v1.0.0.zip

更新日志：

1. 调整OSS C SDK依赖的XML第三方库，使用minixml替换libxml减小OSS C SDK的大小
2. 修改编译方式为CMAKE，方便用户使用SDK
3. 新增oss_upload_file接口，封装multipart upload相关的接口，使用multipart方式上传文件
4. 新增oss_delete_objects_by_prefix接口，删除指定prefix的object
5. 新增OSS C SDK根据object name或者filename自动添加content_type

OSS C SDK开发包（2015-11-12）版本0.0.7

下载地址：

- Linux: aliyun_OSS_C_SDK_v0.0.7.tar.gz
- Windows: oss_c_sdk_windows_v0.0.7.zip

更新日志：

1. OSS C SDK修复sts_token超过http header最大限制的问题

OSS C SDK开发包（2015-10-29）版本0.0.6

下载地址：

- Linux: aliyun_OSS_C_SDK_v0.0.6.tar.gz
- Windows: oss_c_sdk_windows_v0.0.6.zip

更新日志：

1. OSS C SDK签名时请求头支持x-oss-date，允许用户指定签名时间，解决系统时间偏差导致签名出错的问题
2. OSS C SDK支持CNAME方式访问OSS，CNAME方式请求时指定is_oss_domain值为0
3. 新增OSS C SDK demo,提供简单的接口调用示例，方便用户快速入门
4. OSS C SDK sample示例中去除对utf8第三方库的依赖

OSS C SDK开发包（2015-09-14）版本0.0.5

下载地址：

- Linux: aliyun_OSS_C_SDK_v0.0.5.tar.gz
- Windows: aliyun_OSS_C_SDK_windows_v0.0.5.rar

更新日志：

1. 调整签名时获取GMT时间的方法
2. 调整req_id的处理方式，改为从aos_status_t放回状态中直接获取

OSS C SDK开发包（2015-08-17）版本0.0.4

下载地址：

- Linux: aliyun_OSS_C_SDK_v0.0.4.tar.gz
- Windows: aliyun_OSS_C_SDK_windows_v0.0.4.rar

更新日志：

1. 支持keeplive长连接

2. 支持lifecycle设置

OSS C SDK开发包（2015-07-08）版本0.0.3

下载地址：

- Linux: aliyun_OSS_C_SDK_v0.0.3.tar.gz
- Windows: aliyun_OSS_C_SDK_windows_v0.0.3.rar

更新日志：

1. 增加oss_append_object_from_buffer接口，支持追加上传buffer中的内容到object
2. 增加oss_append_object_from_file接口，支持追加上传文件中的内容到object

OSS C SDK开发包（2015-06-10）版本0.0.2

更新日志：

1. 增加oss_upload_part_copy接口，支持Upload Part Copy方式拷贝
2. 增加使用sts服务临时授权方式访问OSS

OSS C SDK开发包（2015-05-28）版本0.0.1

更新日志：

1. 增加oss_create_bucket接口，创建oss bucket
2. 增加oss_delete_bucket接口，删除oss bucket
3. 增加oss_get_bucket_acl接口，获取oss bucket的acl
4. 增加oss_list_object接口，列出oss bucket中的object
5. 增加oss_put_object_from_buffer接口，上传buffer中的内容到object
6. 增加oss_put_object_from_file接口，上传文件中的内容到object
7. 增加oss_get_object_to_buffer接口，获取object的内容到buffer
8. 增加oss_get_object_to_file接口，获取object的内容到文件
9. 增加oss_head_object接口，获取object的user meta信息
10. 增加oss_delete_object接口，删除object
11. 增加oss_copy_object接口，拷贝objet
12. 增加oss_init_multipart_upload接口，初始化multipart upload
13. 增加oss_upload_part_from_buffer接口，上传buffer中的内容到块中
14. 增加oss_upload_part_from_file接口，上传文件中的内容到块
15. 增加oss_list_upload_part接口，获取所有已上传的块信息
16. 增加oss_complete_multipart_upload接口，完成分块上传
17. 增加oss_abort_multipart_upload接口，取消分块上传事件
18. 增加oss_list_multipart_upload接口，获取bucket内所有分块上传事件
19. 增加oss_gen_signed_url接口，生成一个签名的URL
20. 增加oss_put_object_from_buffer_by_url接口，使用url签名的方式上传buffer中的内容到object
21. 增加oss_put_object_from_file_by_url接口，使用url签名的方式上传文件中的内容到object

22. 增加oss_get_object_to_buffer_by_ur接口，使用url签名的方式获取object的内容到buffer
23. 增加oss_get_object_to_file_by_url接口，使用url签名的方式获取object的内容到文件中
24. 增加oss_head_object_by_url接口，使用url签名的方式获取object的user meta信息

Media C SDK开发包

MEDIA C SDK 开发包

OSS MEDIA C SDK最新版本

点击查看

OSS MEDIA C SDK开发包（2016-03-06）版本1.0.0

下载地址：

- Linux: aliyun_oss_media_c_sdk_1.0.0.tar.gz

更新日志：

- 新增H.264,aac转HLS基础接口
- 新增H.264,aac转HLS的录播、直播封装接口
- 优化client，server端接口，提高易用性
- 解决用户无法设置日志级别的问题

OSS C SDK开发包（2016-01-16）版本0.2.0

下载地址：

- Linux: aliyun_oss_media_c_sdk_0.2.0.tar.gz

更新日志：

- 解决头文件多次引用保护无效的问题
- 解决日志开关设置有误的问题
- 解决sts相关sample有误的问题
- 解决部分内存泄露、数组越界等问题
- 优化编译，项目结构等
- 优化、简化示例程序

- 合并client和server两个项目，支持独立编译
- 增加测试用例

.NET SDK开发包

.NET SDK 最新开发包

环境要求：

Windows

- .NET Framework 2.0及以上版本
- 必须注册有Aliyun.com用户账户

Linux / Mac

- Mono 3.12及以上版本

程序集：Aliyun.OSS.dll

最新版本号：2.4.0

文档：点击查看

包结构：

- bin
  - Aliyun.OSS.dll .NET程序集文件
  - Aliyun.OSS.pdb 调试和项目状态信息文件
  - Aliyun.OSS.xml 程序集注释文档
- doc
  - Aliyun.OSS.chm 帮助文档
- src
  - SDK源代码
- sample
  - Sample源代码

.NET SDK 开发包（2016-12-14）

下载地址：aliyun_dotnet_sdk_2.4.0

更新日志:

- 增加：SetObjectAcl/GetObjectAcl接口
- 增加：图片处理功能
- 增加：OssClient的构造函数，方便使用
- 修复：GetObject中使用MemoryStream占用大量内存的问题
- 修复：ObjectMetadata设置ContentLength时5G限制的问题
- 修复：使用Proxy时个别请求没有Proxy-Authorization头的问题

.NET SDK 开发包（2016-03-28）

下载地址：aliyun_dotnet_sdk_2.3.0

更新日志:

- ObjectMetadata新增ContentMd5属性，支持上传文件时验证MD5
- 拷贝时验证目标bucket和object名称合法性
- 解决无法从Metadata获取Expires的问题
- 解决重试机制失效的问题
- 解决设置Content-Encoding等值为null时抛异常的问题
- 解决Endpoint头尾含空字符报错的问题

.NET SDK 开发包（2015-12-12）

下载地址：aliyun_dotnet_sdk_2.2.0

更新日志:

- 支持Mono 3.12.0及其以上版本
- 新增追加文件接口：AppendObject
- 新增断点续传上传接口：ResumableUploadObject
- 新增断点续传拷贝接口：ResumableCopyObject
- 新增部分示例程序
- 移除对System.Web库的依赖

.NET SDK 开发包（2015-11-28）

下载地址：aliyun_dotnet_sdk_2.1.0

更新日志:

- .NET Framework 2.0和.NET Framework 3.5支持
- 大文件拷贝接口：CopyBigObject
- 大文件上传接口：UploadBigObject
- 文件meta修改接口：ModifyObjectMeta
- 提升SDK健壮性
- ContentType支持大多数MIME种类（226种）
- 补齐SDK API速查文档中的缺失注释
- 删除某些类中已经废弃的ObjectMetaData属性，请使用类中对应的的ObjectMetadata属性

.NET SDK 开发包（2015-11-18）

更新日志:

- 自动根据对象key和上传文件名后缀判断ContentType
- ListObjects，ListMultipartUploads，DeleteObjects接口默认增加EncodingType参数
- UploadPart接口新增内容md5校验
- 新增部分示例程序
- 精简命名空间
- 合并重复目录
- 统一目录名称
- 删除重复的测试项目
- 修改CNAME支持形式
- 当存储空间或者文件不存在时，DoesObjectExist不再抛出异常，而是返回false

.NET SDK 开发包（2015-05-28）

下载地址：aliyun_dotnet_sdk_20150528

更新日志:

- 添加Bucket Lifecycle支持，可添加、删除Lifecycle规则；
- 添加DoesBucketExist、DoesObjectExist接口，用于判断Bucket/Object的存在性；
- 添加SwitchCredentials，可在运行期更换用户账号信息；
- 添加ICredentialsProvider接口类，通过实现该类提供自定义的Credentials生成策略。
- 添加GeneratePostPolicy接口，用于生成Post Policy；
- 添加异步化接口（支持Put/Get/List/Copy/PartCopy等异步操作）；
- 添加STS支持；
- 添加自定义时间校准功能，可通过Client配置项SetCustomEpochTicks接口进行设置；
- 添加Chunked编码传输支持，上传时可以不指定Content-Length项；
- 修复设置Bucket CORS的Expose Header属性，但取出的结果为空的bug；
- 修复ListObjects请求返回的CommonPrefixs包含多个Prefix时，SDK只能取到第一个Prefix的bug；
- 修复当出现OSS相关异常时，解析出的RequestId、HostId为null的bug；
- 修复CopyObject/CopyPart接口中source key包含中文，出现编码错误的bug;

.NET SDK 开发包（2015-01-15）

下载地址：aliyun_dotnet_sdk_20150115

更新日志:

- 移除OTS分支，程序集命名更改为Aliyun.OSS.dll
- .NET Framework版本升至4.0及以上
- OSS: 添加Copy Part、Delete Objects、Bucket Referer List等接口
- OSS: 添加ListBuckets分页功能
- OSS: 添加CNAME支持
- OSS: 修复Put/GetObject流中断问题

- OSS: 增加Samples

.NET SDK 开发包（2014-06-26）

下载地址：aliyun_dotnet_sdk_20140626

Open Services SDK for.NET 包含了OSS和OTS的SDK。.NETSDK采用了与 Java SDK统一的接口设计，并结合C#语言特点适当地改进。（最新版本已支持OSS的分块上传操作）

更新日志：

- 加入cors功能。

2013/09/02- OSS:

- 修复了某些情况下无法抛出正确的异常的Bug。
- 优化了SDK的性能。

2013/06/04- OSS:

- 将默认OSS服务访问方式修改为三级域名方式。

2013/05/20- OTS:

- 将默认的OTS服务地址更新为：ots.aliyuncs.com
- 新加入对Mono的支持。
- 修复了SDK中的几处Bug，使其运行更稳定。

2013/04/10- OSS：

- 添加了Object分块上传（Multipart Upload）功能。
- 添加了Copy Object功能。
- 添加了生成预签名URL的功能。
- 分离出IOss接口，并由OssClient继承此接口。

2012/10/10- OSS:

- 将默认的OSS服务地址更新为：oss.aliyuncs.com

2012/09/05- OSS:

- 解决ListObjects时Prefix等参数无效的问题。

2012/06/15- OSS：

- 首次加入对OSS的支持。包含了OSS Bucket、ACL、Object的创建、修改、读取、删除等基本操作。
- OTS：
- OTSClient.GetRowsByOffset支持反向读取。
- 加入对特定请求错误的自动处理机制。
- 增加HTML格式的帮助文件。

2012/05/16- OTS

- Client.GetRowsByRange支持反向读取。

2012/03/16- OTS

- 访问接口，包括对表、表组的创建、修改和删除等操作，对数据的插入、修改、删除和查询等操作。
- 访问的客户端设置，如果代理设置、HTTP连接属性设置等。
- 统一的结构化异常处理。

OssImport版本

版本号：2.3.0 日期：2017-12-15

变更内容

- 增加：支持使用高速通道迁移数据
- 修复：集团开发规约格式化代码

版本号：2.2.4 日期：2017-10-23

变更内容

- 增加：版本化管理，支持version命令，方便问题排除，详见#OSS-2793
- 修复：进程内存控制，防止占用过大内存被杀，详见#OSS-2794
- 修复：本地大文件上传优化，从单线程上传优化成多线程并发上传
- 修复：通过System.getProperty配置项变成在配置文件中配置，详见#OSS-2756
- 修复：把异常信息记录到日志

版本号：2.2.3 日期：2017-10-10

变更内容

- 增加：又拍云支持从CDN下载文件
- 增加：支持配置文件设置网络超时时间
- 修复：又拍createSingleThreadTask报NPE，详见#OSS-27870
- 修复：dispatcherThreadNum的默认值修改为1
- 修复：Linux单机版在非工作目录下启动无法正常运行

版本号：2.2.2 日期：2017-09-11

变更内容

- 修复：配置文件动态加载
- 修复：重试级别优化，从Task到File
- 修复：Job配置文件支持isSkipExistFile

版本号：2.2.1 日期：2017-07-29

变更内容

- 修复：http等迁移数据时流控无效
- 修复：Windows脚本改成英文
- 增加：README的内容

版本号：2.2.0 日期：2017-07-12

变更内容。

- 增加：支持Azure Blob迁移到OSS
- 增加：单机版本，增加Linux命令行console.sh、一键执行脚本import.sh
- 增加：单机版本，增加Windows命令行console.sh、一键执行脚本import.bat，去除其它bat
- 增加：分布式版本，增加Linux命令行console.sh
- 修复：命名优化，slave->worker, dispatcher->tasktracker, 命名空间ossimport2->ossimport, Jar名称变更
- 修复：awk在Ubuntu的目录是/usr/bin/awk而不是/bin/awk，awk等命令去除目录