天翼云

对象存储

(Object-Oriented Storage,OOS)

Android 开发者指南 V6

天翼云科技有限公司

目录

1 前言

对象存储（Object-Oriented Storage，OOS）为客户提供一种海量、弹性、廉价、高可用的存储服务。OOS 提供了基于 Web 门户和基于 HTTP REST 接口两种访问方式，用户可以在任何地方通过互联网对数据进行管理和访问，也可以通过 OOS 提供的 SDK 来调用 OOS 服务。

说明：本手册适用于对象存储网络。

2 使用条件

2.1 使用前需具备的知识

已经熟悉 OOS 的基本概念，如 Bucket、Object、AccessKey 等。具体介绍可参见《OOS 开发者文档-v6》。

2.2 基础条件

用户需要具备以下条件，然后才能够使用 Android SDK：

- 一个 OOS 账户。
- 开通 OOS 服务

2.3 获取访问凭证

AccessKeyId 和 SecretKey 是用户访问 OOS 的密钥，OOS 会通过它来验证用户的资源请求，请妥善保管。关于 AccessKeyId 和 SecretKey 的介绍，请阅读《OOS 开发者文档-v6》。

2.4 开发环境要求

Android 系统版本：2.3 及以上。

2.5 安装 SDK

- 获取 SDK

可以从官网下载编译好 jar 包或者下载 sdk 源码。

下载地址：http://oos-cn.ctyunapi.cn/sdk/oos/android/oos-android-sdk-6.2.0.zip

对应 MD5：http://oos-cn.ctyunapi.cn/sdk/oos/android/oos-android-sdk-6.2.0-md5.txt

可以直接从官网获取 jar，添加到工程依赖。

- 安装 SDK

也可使用获取到的源码编译 jar 包。

```
# 进入目录
cd /oos-android-sdk-src
# 执行打包脚本
gradlew makeJar
# 生成的 jar 在/oos-android-sdk-src/amazonaws\build\libs
```

2.6 使用 SDK 前设置

- 权限设置

OOS Android SDK 所需 Android 权限如下：

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

3 OOS 服务代码示例

3.1 初始化工作

首先需要创建一个 client 对象，后续 OOS 的所有操作可通过 client 的接口进行访问。

- 准备配置信息

```
//创建 client 配置对象
ClientConfiguration clientConfig = new ClientConfiguration();
clientConfig.setConnectionTimeout(30*1000); //设置连接的超时时间，单位毫秒
clientConfig.setSocketTimeout(30*1000) ;       //设置 socket 超时时间，单位毫秒
//创建选项对象
S3ClientOptions options = new S3ClientOptions();
//可以使用 V4 签名，也可以使用 V2 签名，false 为 V2 签名，使用系统属性来设置
System.setProperty(SDKGlobalConfiguration.ENABLE_S3_SIGV4_SYSTEM_PROPERTY,
"false");
//V4 签名可以用负载参与签名、本例设置负载不参与签名
  options.setPayloadSigningEnabled(false);
```

- 创建 client

```
//使用 AK 和 SK 创建密码以及配置对象创建 client
AmazonS3 oosClient = new AmazonS3Client(
            new PropertiesCredentials(TestConfig.OOS_ACCESS_ID,
TestConfig.OOS_ACCESS_KEY),clientConfig);
//设置 endpoint
oosClient.setEndpoint(TestConfig.OOS_ENDPOINT);
//设置选项
oosClient.setS3ClientOptions(options);
```

3.2 关于 Service 的操作

3.2.1 GET Service(List Bucket)

对于做 Get 请求的服务，返回请求者拥有的所有 Bucket，其中“/”表示根目录。

注意：只有验证用户可以执行该操作，匿名用户不能执行该操作。

示例代码

```
//获取 bucket 列表
List<Bucket> listBuckets = oosClient.listBuckets();
for (Bucket bucketInfo : listBuckets) {
      System.out.println("listBuckets:"
                  + "\t Name:" + bucketInfo.getName()    //bucket 名字
                  + "\t CreationDate:" + bucketInfo.getCreationDate()//创建时间
                  + "\t Owner:"+ bucketInfo.getOwner());//所属用户
}
```

3.2.2 GET Regions

获取资源池中的索引位置和数据位置列表。

示例代码

```
//获取 Region 列表
      CtyunGetRegionsResult region =    oosClient.ctyunGetRegions();
if(region != null ){
      List<String> metadataRegions = region.getMetadataRegions();
      List<String> dataRegions = region.getDataRegions();
      if(metadataRegions!= null){
            // dataRegion 列表
            for (String    metadataRegion : metadataRegions) {
                  System.out.println("getRegions:"
                              + "\t metadataRegion:" + metadataRegion);
            }
      }
      if(dataRegions!= null){
            // metaRegion  列表
            for (String    dataRegion : region.getDataRegions()) {
                  System.out.println("getRegions:"
                              + "\t dataRegion:" + dataRegion);
            }
      }
}
```

3.3 关于 Bucket 的操作

3.3.1 PUT Bucket

Put Bucket 用来创建一个 Bucket。只有根用户和拥有相应权限的子用户才能执行此操作。

Bucket 的命名方式如下：

- Bucket 名称必须全局唯一；
- Bucket 名称长度介于 3 到 63 字节之间；
- Bucket 名称只能由小写字母、数字、短横线（-）和点（.）组成；
- Bucket 名称可以由一个或者多个小节组成，小节之间用点（.）隔开，各个小节需要：
  - 必须以小写字母或者数字开始；
  - 必须以小写字母或者数字结束。
- Bucket 名称不能是 IP 地址形式（如 192.162.0.1）；
- Bucket 名称不能是一组或多组“数字.数字”的组合；
- Bucket 名称中不能包含双点（..）、横线点（-.）和点横线（.-）；
- 不允许使用非法敏感字符，例如暴恐涉政相关信息等。

示例代码

```
// 准备 dataLocation 的 list
List<String> dataRegions = new ArrayList<>();
dataRegions.add(dataLocation1);// 设置第一个 dataLocation
dataRegions.add(dataLocation2);// 设置第二个 dataLocation
// 创建 dataLocaton 配置实例
CtyunBucketDataLocation dataLocation = new
CtyunBucketDataLocation().withDataRegions(dataRegions);
// 是否允许调度
dataLocation.setStragegy(CtyunBucketDataLocation.CtyunBucketDataScheduleStrategy.NotAllowed);
// 创建 request
CreateBucketRequest createBucketRequest = new
CreateBucketRequest(myBucketName,metaLocation,dataLocation);
// 发起创建 bucket 请求
Bucket bucketInfo = oosClient.createBucket(createBucketRequest);
```

```
System.out.println("puteBucket:"
        + "\t Name:" + bucketInfo.getName() // bucketname
        + "\t CreationDate:" + bucketInfo.getCreationDate() //  创建时间
        + "\t Owner:"+ bucketInfo.getOwner()); //  用户
```

3.3.2 GET Bucket location

该操作用来获取 Bucket 的索引位置和数据位置，只有具有该权限的用户才能执行此操作。

示例代码

```
// 获取 Bucket Location 信息
CtyunGetBucketLocationResult locationResult =
oosClient.getBucketLocation(myBucketName);
// dataLocation 结果
for(String dataRegion : locationResult.getDataLocation().getDataRegions()) {
    System.out.println("dataLocation: " + dataRegion);
}
// metaLocation 结果
System.out.println("metaLocation: " + locationResult.getMetaRegion());
```

3.3.3 GET Bucket ACL

此操作用来获取 Bucket ACL 信息，只有拥有 GET Bucket ACL 权限的用户才可以执行此操作

示例代码

```
// 获取 bucket 的 acl 信息
AccessControlList list = oosClient.getBucketAcl(myBucketName);
// 获取到的 displayame 和 用户
System.out.println("owner displayname :" + list.getOwner().getDisplayName() +"owner id:"+
list.getOwner().getId());
// 获取到的 grantee 信息
for (Grant grant : list.getGrantsAsList()){
    System.out.println("grantee:" + grant.getGrantee().getIdentifier() + " " +
grant.getGrantee().getTypeIdentifier());
    System.out.println("permission:" + grant.getPermission() );
}
```

3.3.4 Get Bucket(List Objects)

此操作用来返回 Bucket 中部分或者全部（最多 1000）的 Object 信息。用户可以在请求元素中设置选择条件来获取 Bucket 中的 Object 的子集。

示例代码

```
// 创建 request
ListObjectsRequest listObjectsRequest    = new ListObjectsRequest();
// 设置 Bucket
listObjectsRequest.setBucketName(myBucketName);
// 设置查询数量
listObjectsRequest.setMaxKeys(500);
// 发起请求
ObjectListing list = oosClient.listObjects(listObjectsRequest);
for(S3ObjectSummary object:list.getObjectSummaries()){
      System.out.println();
      System.out.println("key: " + object.getKey() +    // 对象名
                  " size :" + object.getSize() +                 // 对象大小
                  " etage: " + object.getETag());               // 对象 etag
}
```

3.3.5 DELETE Bucket

此操作用来删除 Bucket，但要求被删除 Bucket 中无 Object，即该 Bucket 中的所有 Object 都已被删除。

示例代码

```
// 删除 bucket
oosClient.deleteBucket(bucketName);
```

3.3.6 PUT Bucket Policy

此操作用来在 PUT 操作的 url 中加上 Policy，可以进行添加或修改 Policy 的操作。如果 Bucket 已经存在了 Policy，此操作会替换原有 Policy。

示例代码

```
String buffer = "     {     "
            +"          'Version':'2012-10-17',     "
            +"          'Id':'http referer policy example',     "
            +"                'Statement':[     "
            +"                    {     "
            +"                     'Sid':'Allow get requests referred by www.mysite.com ',     "
            +"                          'Effect':'Allow',     "
            +"                           'Principal':{     "
            +"                                'AWS':[     "
            +"                                  '*'     "
            +"                                      ]     "
            +"                              },     "
            +"                         'Action':'s3:*',     "
            +"                         'Resource':'arn:aws:s3:::" + myBucketName +"/*',     "
            +"                         'Condition':{     "
            +"                              'StringLike':{     "
            +"                                   'aws:Referer':[     "
            +"                                        'http://www.mysite.com/*'    "
            +"                                   ]   "
            +"                          }   "
            +"                    }   "
            +"               }   "
            +"          ]   "
            +"         }   " ;
// 设置 bucket Policy
oosClient.setBucketPolicy(myBucketName,buffer);
```

3.3.7 GET Bucket Policy

在 GET 操作的 url 中加上 Policy，获得指定 Bucket 的 Policy。如果 Bucket 没有 Policy，返回 404，NoSuchPolicy 错误

示例代码

```
// 获取 bucket policy
BucketPolicy policy = oosClient.getBucketPolicy(myBucketName);
System.out.println("result: " + policy.getPolicyText());
```

3.3.8 DELETE Bucket Policy

在 DELETE 操作的 url 中加上 Policy，可以删除指定 Bucket 的 Policy。

- 如果 Bucket 配置了 Policy，删除成功，返回 200 0K。
- 如果 Bucket 没有配置 Policy，返回 204 NoContent。

示例代码

```
// 删除 bucket policy
oosClient.deleteBucketPolicy(myBucketName);
```

3.3.9 PUT Bucket Website

在 PUT Bucket 操作的 url 中加上 website，可以设置 website 配置。如果 Bucket 已经存在了 website，此操作会替换原有 website。

WebSite 功能可以让用户将静态网站存放到 OOS 上。对于已经设置了 WebSite 的 Bucket，当用户访问 http://bucketName.oos-website-cn.oos.ctyunapi.cn 时，会跳转到用户指定的主页，当出现 4XX 错误时，会跳转到用户指定的出错页面。如果想通过自有域名的形式（例如 http://yourdomain.com/login.html）而非通过第三方域名的形式（例如 http://yourdomain.com.oos.ctyunapi.cn/login.html）访问，可以创建一个名为“yourdomain.com”的 bucket，并在域名管理系统中将“yourdomain.com”增加一个别名记录“oos.ctyunapi.cn”。

示例代码

```
// 创建配置对象
BucketWebsiteConfiguration configuration = new BucketWebsiteConfiguration();
// 设置 indexhtml
configuration.setIndexDocumentSuffix("new-index.html");
// 设置 errorhtml
configuration.setErrorDocument("new-error.html");
// 发起请求
oosClient.setBucketWebsiteConfiguration(myBucketName,configuration);
```

3.3.10 GET Bucket Website

在 GET Bucket 的 url 中加上 website，获得指定 Bucket 的 website。

示例代码

```
// 获取 bucket website
BucketWebsiteConfiguration result =
oosClient.getBucketWebsiteConfiguration(myBucketName);
// 获取到的 indexhtml 和 errhtml
System.out.println("result: indexDocument:" + result.getIndexDocumentSuffix() +
        "errorDocument :" + result.getErrorDocument());
```

3.3.11 DELETE Bucket Website

在 Delete Bucket 操作的 url 中加上 website，可以删除指定 Bucket 的 website。

如果 Bucket 没有 website，返回 200 OK。

示例代码

```
oosClient.deleteBucketWebsiteConfiguration(myBucketName);
```

3.3.12 List Multipart Uploads

该操作用于列出所有已经通过 Initiate Multipart Upload 请求初始化，但未完成或未终止的分片上传过程。

示例代码

```
// 获取 bucket 下的 multipartUpload 列表
ListMultipartUploadsRequest listMultipartUploadsRequest = new
ListMultipartUploadsRequest(myBucketName);
MultipartUploadListing res = oosClient.listMultipartUploads(listMultipartUploadsRequest);
// 获取到到 multipartUpload 的 uploadId 和 key
for(MultipartUpload one : res.getMultipartUploads()){
    System.out.println("uplaod id : " + one.getUploadId() +
    " uplaod key : " + one.getKey());
}
```

3.3.13 PUT Bucket Logging

在 PUT Bucket 的 url 中加上 logging，可以进行添加/修改/删除 logging 的操作。

如果 Bucket 已经存在了 logging，此操作会替换原有 logging。

示例代码

```
// 设置 bucket logging
BucketLoggingConfiguration loggingConfiguration = new BucketLoggingConfiguration();
// 设置目标日志前缀
loggingConfiguration.setLogFilePrefix("prefix-1");
// 设置目标 bucket
loggingConfiguration.setDestinationBucketName(myBucketName);
// 创建 request
SetBucketLoggingConfigurationRequest setBucketLoggingConfigurationRequest =
        new SetBucketLoggingConfigurationRequest(myBucketName,loggingConfiguration);
// 发起请求
oosClient.setBucketLoggingConfiguration(setBucketLoggingConfigurationRequest);
```

3.3.14 GET Bucket Logging

在 GET Bucket 操作的 url 中加上 logging，可以获得指定 Bucket 的 logging。

示例代码

```
// 获取 bucket logging
BucketLoggingConfiguration logging =
oosClient.getBucketLoggingConfiguration(myBucketName);
// 获取到的目标日志的前缀和目标日志的 bucket
System.out.println("result dest :" + logging.getDestinationBucketName() +
        "prefix :" + logging.getLogFilePrefix() +
        "enabled :" + logging.isLoggingEnabled());
```

3.3.15 HEAD Bucket

此操作用于判断 Bucket 是否存在，而且用户是否有权限访问。如果 Bucket 存在，而且用户有权限访问时，此操作返回 200 OK。否则，返回 404 不存在，或者 403 没有权限。

示例代码

```
// head Bucket
HeadBucketResult result = oosClient.headBucket(new HeadBucketRequest(myBucketName));
```

3.3.16 PUT Bucket Lifecycle

此操作用来设置 Bucket 生命周期规则。通过设置存储桶的生命周期规则，可以：删除与生命周期规则匹配的对象。当对象的生命周期到期时，OOS 会异步删除它们。生命周期中配置的到期时间和实际删除时间之间可能会有一段延迟。但对象到期被删除后，用户将不需要再为到期的对象付费。OOS 删除到期对象后，会在 Bucket log 中记录一条日志，操作项是"OOS.EXPIRE.OBJECT"。

示例代码

```
// 设置 bucket lifecycle
BucketLifecycleConfiguration bucketLifecycleConfiguration = new
BucketLifecycleConfiguration();
// 准备 rule 列表
List<Rule> rules = new ArrayList<Rule>();
Rule rule1 = new Rule();
// 设置 ruleID
rule1.setId("r1");
// 设置前缀
rule1.setPrefix("logs");
// 设置是否启用
rule1.setStatus("Enabled");
// 设置超期时间
rule1.setExpirationInDays(30);
// 指定固定日期设置超期时间
java.util.Date expireDate = new java.util.Date();
rule1.setExpirationDate(expireDate);
rules.add(rule1);
bucketLifecycleConfiguration.setRules(rules);
// 发起请求
oosClient.setBucketLifecycleConfiguration(myBucketName,bucketLifecycleConfiguration);
```

3.3.17 GET Bucket Lifecycle

此接口用于返回配置的 Bucket 生命周期规则。

示例代码

```
// 获取 bucket lifecycle
BucketLifecycleConfiguration result =
oosClient.getBucketLifecycleConfiguration(myBucketName);
// 获取到 rule 信息
for(Rule rule : result.getRules()){
    System.out.println("rule id:" + rule.getId() +
            " status:" + rule.getStatus() +
            "prefix :" + rule.getPrefix() +
            "expir" + rule.getExpirationInDays());
}
```

3.3.18 DELETE Buckete Lifecycle

此操作用来删除指定 Bucket 所有生命周期。

说明：删除 Bucket 的生命周期后，用户的对象将永远不会到期，OOS 也不会再自动删除对象。

示例代码

```
// 删除 bucket lifecycle
oosClient.deleteBucketLifecycleConfiguration(myBucketName);
```

3.3.19 PUT Bucket cors

跨域资源共享(Cross-Origin Resource Sharing，CORS)定义了客户端 Web 应用程序在一个域中与另一个域中的资源进行交互的方式，是浏览器出于安全考虑而设置的一个限制，即同源策略。例如，当来自于 A 网站的页面中的 JavaScript 代码希望访问 B 网站的时候，浏览器会拒绝该访问，因为 A、B 两个网站是属于不同的域。

通过 CORS，客户可以构建丰富的客户端 Web 应用程序，同时可以选择性地允许跨域访问 OOS 资源。

示例代码

```
// 设置 bucket cors
// 创建 cors 配置对象
BucketCrossOriginConfiguration bucketCrossOriginConfiguration = new
BucketCrossOriginConfiguration();
// 创建 rule 对象
CORSRule rule = new CORSRule();
// 设置允许的 method
List<CORSRule.AllowedMethods> allowedMethods = new
ArrayList<CORSRule.AllowedMethods>();
allowedMethods.add(CORSRule.AllowedMethods.GET);
allowedMethods.add(CORSRule.AllowedMethods.POST);
// 设置允许的 origins
List<String> allowedOrigins = new ArrayList();
allowedOrigins.add("a");
allowedOrigins.add("b");
// 设置 headers
int maxAgeSeconds = 1000;
List<String> exposedHeaders = new ArrayList();
List<String> allowedHeaders = new ArrayList();
// 设置 exposedheaders
exposedHeaders.add("e1");
exposedHeaders.add("e2");
// 设置允许的 headers
allowedHeaders.add("a1");
allowedHeaders.add("a2");
rule.setId("myfirstRuleId");
```

```
rule.setAllowedMethods(allowedMethods);
rule.setAllowedOrigins(allowedOrigins);
rule.setExposedHeaders(exposedHeaders);
// 指定浏览器预检缓存时间
rule.setMaxAgeSeconds(3600);
rule.setAllowedHeaders(allowedHeaders);
// 创建 corsRule 列表
List<CORSRule> rules = new ArrayList<CORSRule>();
rules.add(rule);
// 设置 rules 列表到配置对象
bucketCrossOriginConfiguration.setRules(rules);
// 发起请求
oosClient.setBucketCrossOriginConfiguration(myBucketName,bucketCrossOriginConfiguratio
n);
```

3.3.20 GET Bucket cors

此操作用来返回 Bucket 的跨域配置信息。

示例代码

```
// 获取 bucket cors 信息
BucketCrossOriginConfiguration cors =
oosClient.getBucketCrossOriginConfiguration(myBucketName);
// 打印获取结果信息
for(CORSRule rule : cors.getRules()){
    System.out.println("rule id:" + rule.getId() +
            " allowOrigins :" + rule.getAllowedOrigins() +
            " allowHeaders:" + rule.getAllowedHeaders() +
            " allowMethod" + rule.getAllowedMethods() +
            "exposeHeaders" + rule.getExposedHeaders());
}
```

3.3.21 DELETE Bucket cors

删除 Bucket 的跨域配置信息。

- 如果 Bucket 配置了 CORS，删除成功，返回 200 0K。
- 如果 Bucket 没有配置 CORS，返回 204 NoContent。

示例代码

```
// 删除 bucket cors 信息
oosClient.deleteBucketCrossOriginConfiguration(myBucketName);
```

3.4 关于 Object 的操作

3.4.1 PUT Object

此操作用来向指定 Bucket 中添加一个对象。

示例代码

```
// 上传文件对象
File testFile = new File(docPath,fileName);
// 创建上传对象请求
PutObjectRequest request = new PutObjectRequest(myBucketName,objectKey,testFile);
// 设置请求携带元数据对象
ObjectMetadata metadata = new ObjectMetadata();
// 构造 dataLocation 对象
CtyunBucketDataLocation dataLocation = new CtyunBucketDataLocation();
// 设置 dataLocation 的 type
dataLocation.setType(CtyunBucketDataLocation.CtyunBucketDataType.Specified);
// 构造 dataLocation 列表
List<String> dataRegions = new ArrayList<String>();
dataRegions.add("QingDao");
dataRegions.add("ShenZhen");
dataLocation.setDataRegions(dataRegions);
// 设置 dataLocation 到元数据对象
metadata.setDataLocation(dataLocation);
// 设置请求元数据
request.setMetadata(metadata);
// 发送请求
oosClient.putObject(request);
```

3.4.2 GET Object

此操作用来检索在 OOS 中的对象信息，执行此操作，用户必须对 Object 所在的 Bucket 拥有读权限。如果 Bucket 是 public read 的权限，匿名用户也可以通过非授权的方式进行读操作。

示例代码

```
// 准备存储文件路径
String sdCardPath = Environment.getExternalStorageDirectory().getAbsolutePath();
String sFilePath = sdCardPath + File.separator + "test.txt";
// 创建下载请求
GetObjectRequest request = new GetObjectRequest(myBucketName, objectInBucketKey);
// 创建下载文件对象
File fRecv = new File(sFilePath);
// 发起下载请求
ObjectMetadata meta = oosClient.getObject(request, fRecv);
System.out.println("get object size: " + meta.getContentLength());
```

3.4.3 DELETE Object

此操作用来删除指定的对象，要求用户要对对象所在的 Bucket 拥有写权限。

示例代码

```
// 删除对象
String objectInBucketKey = "myobject";
oosClient.deleteObject(myBucketName,objectInBucketKey);
```

3.4.4 PUT Object - Copy

此操作用来创建一个存储在 OOS 里对象的拷贝。PUT Object - Copy 操作类似于执行一个 GET Object，然后再执行一次 PUT Object。用户对源对象有读权限，对目标 Bucket 有写权限，才能执行 PUT Object - Copy 操作。

示例代码

```
// 拷贝对象
String objectKeySource = "source.txt";
String objectKeyDest = "dest.txt";
// 创建 copy 请求
CopyObjectRequest request = new CopyObjectRequest(
        myBucketName, objectKeySource,myBucketName, objectKeyDest);
// 设置 location
ObjectMetadata metadata = new ObjectMetadata();
CtyunBucketDataLocation dataLocation = new CtyunBucketDataLocation();
dataLocation.setType(CtyunBucketDataLocation.CtyunBucketDataType.Specified);
List<String> dataRegions = new ArrayList<String>();
dataRegions.add("ShenZhen");
dataRegions.add("QingDao");
dataLocation.setDataRegions(dataRegions);
metadata.setDataLocation(dataLocation);
// 设置元数据到请求
request.setNewObjectMetadata(metadata);
// 发起请求
oosClient.copyObject(request);
```

3.4.5 Initial Multipart Upload

本接口初始化一个分片上传（Multipart Upload）操作，并返回一个上传 ID，此 ID 用来将此次分片上传操作中上传的所有片段合并成一个对象。用户在执行每一次子上传请求时都应该指定该 ID。用户也可以在表示整个分片上传完成的最后一个请求中指定该 ID。或者在用户放弃该分片上传操作时指定该 ID。

示例代码

```
// 初始化分片上传
String objectInBucketKey2 = "objectInBucketKey2";
// 创建元数据对象
ObjectMetadata metadata = new ObjectMetadata();
// 创建 dataLocation
CtyunBucketDataLocation dataLocation = new CtyunBucketDataLocation();
dataLocation.setType(CtyunBucketDataLocation.CtyunBucketDataType.Specified);
List<String> dataRegions = new ArrayList<String>();
dataRegions.add("ShenZhen");
dataRegions.add("QingDao");
dataLocation.setDataRegions(dataRegions);
// 设置 dataLocation 到元数据对象
metadata.setDataLocation(dataLocation);
// 创建分片上传请求
InitiateMultipartUploadRequest request = new
InitiateMultipartUploadRequest(myBucketName,objectInBucketKey2, metadata);
// 发起分片上传请求
InitiateMultipartUploadResult res = oosClient.initiateMultipartUpload(request);
// 创建成功后，获取到分片上传的 ID
String uploadID = res.getUploadId();
System.out.println("==========uploadID:\t :" + uploadID);
```

3.4.6 Upload Part

该接口用于实现分片上传操作中片段的上传。

在上传任何一个分片之前，必须执行 Initial Multipart Upload 操作来初始化分片上传操作，初始化成功后，OOS 会返回一个上传 ID，这是一个唯一的标识，用户必须在调用 Upload Part 接口时加入该 ID。

分片号 PartNumber 可以唯一标识一个片段并且定义该分片在对象中的位置，范围从 1 到 10000。如果用户用之前上传过的片段的分片号来上传新的分片，之前的分片将会被覆盖。

除了最后一个分片外，所有分片的大小都应该不小于 5M，最后一个分片的大小不受限制。

为了确保数据不会由于网络传输而毁坏，需要在每个分片上传请求中指定 Content-MD5 头，OOS 通过提供的 Content-MD5 值来检查数据的完整性，如果不匹配，则会返回一个错误信息。

示例代码

响应中包含 Etag 头，用户需要在最后发送完成分片上传过程请求的时候包含该 Etag 值。

```
// 上传分片
// 准备上传的文件
String sdCardPath = Environment.getExternalStorageDirectory().getAbsolutePath();
String sFilePath = sdCardPath + File.separator + "1.part01.rar";
String objectInBucketKey2 = "objectInBucketKey2";
// 准备上传的文件对象
File file = new File(sFilePath);
// 创建请求通过分片上传 ID
UploadPartRequest request = new
UploadPartRequest().withUploadId("1591258704156956376");
// 设置上传的 bucket
request.setBucketName(myBucketName);
// 设置上传对象的 key
request.setKey(objectInBucketKey2);
// 设置分片序号
request.setPartNumber(1);
```

```
// 设置分片大小
request.setPartSize(file.length());
// 设置上传文件
request.setFile(file);
// 上传分片
UploadPartResult res = oosClient.uploadPart(request);
System.out.println("=========第一个分片 res:\t :" + res.toString());
```

3.4.7 Complete Multipart Upload

该接口通过合并之前的上传片段来完成一次分片上传过程。

用户首先初始化分片上传过程，然后通过 Upload Part 接口上传所有分片。在成功将一次分片上传过程的所有相关片段上传之后，调用这个接口来结束分片上传过程。当收到这个请求的时候，OOS 会以分片号升序排列的方式将所有片段依次拼接来创建一个新的对象。在这个 Complete Multipart Upload 请求中，用户需要提供一个片段列表。同时，必须确保这个片段列表中的所有片段必须是已经上传完成的，Complete Multipart Upload 操作会将片段列表中提供的片段拼接起来。对片段列表中的每个片段，需要提供该片段上传完成时返回的 ETag 头的值和对应的分片号。

处理一次 Complete Multipart Upload 请求可能需要花费几分钟时间。OOS 在处理这个请求之前会发送一个值为 200 响应头。在处理这个请求的过程中，OOS 每隔一段时间就会发送一个空格字符来防止连接超时。因为一个请求在初始的 200 响应已经发出之后仍可能失败，用户需要检查响应体内容以判断请求是否成功。

由于 Complete Multipart Upload 请求可能需要花费几分钟时间，所以 OOS 提供了不合并片段也可以读取 Object 内容的功能。在没有调用 Complete Multipart Upload 接口合并片段时，也可以通过调用 Get Object 接口来获取文件内容，OOS 会根据最近一次创建的 uploadId，以分片号升序的方式顺序读取片段内容，返回给客户端。但此时不能返回整个文件的 ETag 值。

示例代码

```
// 合并分片
// 合并对象的 key
String objectInBucketKey2 = "objectInBucketKey2";
// 合并的分片的 etag 列表
List<PartETag> partETags = new ArrayList<PartETag>();
partETags.add(new PartETag(1,"65fb4e39dd0b2d121a67f649f518b856"));
partETags.add(new PartETag(2,"2b81b72b96a5c233e3c1d81b7406eaf5"));
partETags.add(new PartETag(3,"4d215a782ba7d32d07a0662e2a608d75"));
// 创建合并分片请求
CompleteMultipartUploadRequest request = new CompleteMultipartUploadRequest(
```

```
        myBucketName,objectInBucketKey2,"1591258704156956376",partETags);
// 发送请求
CompleteMultipartUploadResult result = oosClient.completeMultipartUpload(request);
```

3.4.8 Abort Multipart Upload

该接口用于终止一次分片上传操作。分片上传操作被终止后，用户不能再通过上传 ID 上传其它片段，之前已上传完成的片段所占用的存储空间将被释放。如果此时任何片段正在上传，该上传过程可能会也可能不会成功。所以，为了释放所有片段所占用的存储空间，可能需要多次终止分片上传操作。

示例代码

```
// 终止分片上传
String objectInBucketKey2 = "objectInBucketKey2";
// 创建终止分片请求
AbortMultipartUploadRequest request = new AbortMultipartUploadRequest(
        myBucketName,objectInBucketKey2,"1591258962174932754");
// 发起请求
oosClient.abortMultipartUpload(request);
```

3.4.9 List Part

该操作用于列出一次分片上传过程中已经上传完成的所有片段。

该操作必须包含一个通过 Initial Multipart Upload 操作获取的上传 ID。该请求最多返回 1000 个上传片段信息，默认返回的片段数是 1000。用户可以通过指定 max-parts 参数来指定一次请求返回的片段数。如果用户的分片上传过程超过 1000 个片段，响应中的 IsTruncated 字段的值则被设置成 true，并且指定一个 NextPartNumberMarker 元素。用户可以在下一个连续的 List Part 请求中加入 part-number-marker 参数，并把它的值设置成上一个请求返回的 NextPartNumberMarker 值。

示例代码

```
// 获取分片列表
String objectInBucketKey2 = "objectInBucketKey2";
// 通过分片 ID 构造获取分片列表请求
ListPartsRequest request = new ListPartsRequest(
            myBucketName,objectInBucketKey2,"1591258704156956376");
// 发起请求
PartListing result = oosClient.listParts(request);
int size = result.getParts().size();
// 打印获取分片列表
System.out.println("=========size:\r\n :" + size);
for(PartSummary part :result.getParts()) {
      System.out.printf("=========partNum:    :" + part.getPartNumber() +
                  " size:" + part.getSize() +
                  " etag: " + part.getETag());
      System.out.println();
}
```

3.4.10 Copy Part

此操作将已经存在的 Object 作为分段上传的片段，拷贝生成一个新的片段。

示例代码

```
// 拷贝对象作为分片
String destBacketName = "destBucket";
String objectkeySource = "sourceObject ";
String objectKeyDest = "destObejct";
// 创建 copypart 请求
CopyPartRequest request = new CopyPartRequest();
request.setSourceBucketName(myBucketName);
request.setSourceKey(objectkeySource);
// 设置目的 bucket
request.setDestinationBucketName(destBacketName);
// 设置目标对象 key
request.setDestinationKey(objectKeyDest);
// 设置作为目标对象第几个分片
request.setPartNumber(3);
// 设置分片上传的 ID
request.setUploadId("1591258704156956376");
// 发起请求
CopyPartResult result = oosClient.copyPart(request);
```

3.4.11 Delete Multiple Objects

批量删除请求包含一个不超过 1000 个 Object 的 XML 列表。在这个 xml 中，需要指定要删除的 Object 的名字。对于要删除的每个 Object，OOS 都会返回删除的结果，成功或者失败。

注意：如果请求中的 Object 不存在，那么 OOS 也会返回删除成功。

批量删除功能支持两种格式的响应，全面信息和简明信息。默认情况下，OOS 在响应中会显示全面信息，即包含每个 Object 的删除结果。在简明信息模式下，OOS 只返回删除出错的 object 的结果。对于成功删除的 Object，在响应中将不返回任何信息。

最后，批量删除功能必须使用 Content-MD5 请求头，OOS 使用此头来保证请求体在传输过程中没有被修改。

示例代码

```
// 删除多个对象
String objectInBucketKey1 = "1.txt";
String objectInBucketKey2 = "2.txt";
// 创建删除多对象请求
DeleteObjectsRequest request = new DeleteObjectsRequest(myBucketName);
DeleteObjectsRequest.KeyVersion key1 = new
DeleteObjectsRequest.KeyVersion(objectInBucketKey1,null);
DeleteObjectsRequest.KeyVersion key2 = new
DeleteObjectsRequest.KeyVersion(objectInBucketKey2,null);
// 创建删除列表
List<DeleteObjectsRequest.KeyVersion> keys = new
ArrayList<DeleteObjectsRequest.KeyVersion>();
keys.add(key1);
keys.add(key2);
// 添加删除列表到请求
request.withKeys(keys);
// 设置是否静默模式
request.setQuiet(false);
// 发起请求
DeleteObjectsResult result = oosClient.deleteObjects(request);
```

3.4.12 生成共享链接

对于私有或只读 Bucket，可以通过生成 Object 的共享链接的方式，将 Object 分享给其他用户，同时可以在链接中设置限速以对下载速度进行控制。

示例代码

```
// 生成共享链接
String objectInBucketKey1 = "test 中文.txt";
// 创建生成共享链接请求
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(
        myBucketName,objectInBucketKey1);
// 设置超期时间
//2019/03/10
java.util.Date expireDate = new java.util.Date();
expireDate.setDate(expireDate.getDate() + 1);
request.setExpiration(expireDate);
//可选 设置限速 rate 单位 KBps 并发数 concurrency 单位个数
request.addRequestParameter("x-amz-limit","rate=2048, concurrency=100");
//可选 单独设置限速
//request.addRequestParameter("x-amz-limit","rate=2048");
// 生成共享链接
URL url = oosClient.generatePresignedUrl(request);
System.out.println(url.toString());
```

3.4.13 HEAD Object

此操作用于获取对象的元数据信息，而不返回数据本身。当只希望获取对象的属性信息时，可以使用此操作。

示例代码

```
// headObject
String objectInBucketKey1 = "myObject";
//  发起请求
ObjectMetadata    metadata = oosClient.getObjectMetadata(myBucketName,objectInBucketKey1);
```

3.4.14 断点续传

通过 MultipleUpload 类以及文件上传请求 UploadFileRequest 类，实现基于分段上传的断点续传的功能。

用 checkpoint 文件来记录所有分片的状态。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。checkpoint 文件默认与待上传的本地文件同目录，为 uploadFile.ucp。

示例代码 1(不使用暂停功能)

```
// 分段上传
// 创建分片上传请求
UploadFileRequest request = new UploadFileRequest(myBucketName,"test/123456.txt");
String sdardPath = Environment.getExternalStorageDirectory().getAbsolutePath();
String filePath = sdardPath + "/uploadFile";
// 设置分段上传的记录文件位置，也可以不设置
//request.setCheckpointFile(sdardPath + "/storage/emulated/0/a.txt");
// 设置上传的并发数目
request.setTaskNum(3);
// 设置需要分段上传的文件
request.setUploadFile(filePath);
// 是否启动断点续传
request.setEnableCheckpoint(true);
// 发起请求
oosClient.asyncMultipleUpload(request);
```

示例代码 2(使用暂停功能)

```
// 分段上传
// 创建分片上传请求
UploadFileRequest request = new UploadFileRequest(myBucketName,"test/123456.txt ");
String sdardPath = Environment.getExternalStorageDirectory().getAbsolutePath();
String filePath = sdardPath + "/uploadFile";
// 设置分段上传的记录文件位置，也可以不设置
//request.setCheckpointFile(sdardPath + "/storage/emulated/0/a.txt");
// 设置上传的并发数目
request.setTaskNum(3);
```

```
// 设置需要分段上传的文件
request.setUploadFile(filePath);
// 是否启动断点续传
request.setEnableCheckpoint(true);
// 发起请求
MultipleUpload upload = new MultipleUpload(request, oosClient);
//此方法会阻塞等待上传结束，所以需要单开另一线程调用 pause 和 proceed 方法才能实现暂停和继续功能。
upload.upload();
//暂停上传
upload.pause();
//结束暂停，继续上传
upload.proceed();
```

3.5 关于 AccessKey 的操作

3.5.1 CreateAccessKey

此操作用来为指定的 IAM 用户创建新的 AccessKey。

说明：

- 新密钥的默认状态是 Active。
- 如果未指明 IAM 用户名，则为请求者创建新的 AccessKey。

示例代码

```
// 创建 accesskey
CreateAccessKeyResult result = oosClient.ctyunCreateAccessKey();
// 打印创建成功的 accesskey
System.out.println(result.getAccessKey());
```

3.5.2 DeleteAccessKey

此操作用来删除指定 IAM 用户关联的 AccessKey。

说明：

- 如果未指定用户名，IAM 将根据签名请求的 OOS AccessKeyID 确定用户名。
- 此操作也可以用来删除根用户的 AccessKey。

示例代码

```
// 删除 accesskey
// 创建删除 accesskey 请求
DeleteAccessKeyRequest request = new DeleteAccessKeyRequest();
// 设置要删除的 accesskey
request.setAccessKeyId("8606fe75d8b9590f93ce");
// 发起请求
oosClient.ctyunDeleteAccessKey(request);
```

3.5.3 UpdateAccessKey

此操作用来更新指定访问密钥（AK）的状态，从 Active 到 Inactive，或者从 Inactive 到 Active。

说明：

- 如果请求中未携带 IAM 用户名，则更新请求者的密钥状态。
- 此操作可以管理根用户的密钥。

示例代码

```
// 更新 accesskey 状态
// 创建更新 acesskey 请求
UpdateAccessKeyRequest updateAccessKeyRequest = new UpdateAccessKeyRequest();
// 设置需要更新的 accesskeyID
updateAccessKeyRequest.setAccessKeyId("ecc0817914ca4e6bbd71");
// 设置是否启用
updateAccessKeyRequest.setStatus("Inactive");
// 发起请求
oosClient.ctyunUpdateAccessKey(updateAccessKeyRequest);
```

3.5.4 ListAccessKey

此操作用来返回指定 IAM 用户的 AK 的详细信息。

说明：

- 如果指定用户没有 AK，则操作返回空列表。
- 如果未指定 IAM 用户，则返回请求者的 AK。

示例代码

```
// 获取 acesskey 列表
// 创建 listacesskey 列表请求
ListAccessKeysRequest listAccessKeysRequest = new ListAccessKeysRequest();
// 发起请求
ListAccessKeysResult result = oosClient.ctyunListAccessKeys(listAccessKeysRequest);
// 返回结果是否翻页
System.out.println("result Marker:" + result.getMarker() +
            " isTruncated: "+ result.isTruncated());
// 打印返回的 accesskey 和用户名、状态
for(AccessKeyMetadata data : result.getAccessKeyMetadata()){
      System.out.println("key: " + data.getAccessKeyId() +
                  " UserName:" + data.getUserName() +
                  " status:" + data.getStatus() +
                  " createDate:" + data.getCreateDate());
}
System.out.println("size: " + result.getAccessKeyMetadata().size());
```