天翼云

对象存储

(Object-Oriented Storage,OOS)

Java SDK 开发者指南 V5

天翼云科技有限公司

目录

1 前言

1.1 对象存储（OOS）

对象存储（Object-Oriented Storage，OOS）是中国电信为客户提供的一种海量、弹性、廉价、高可用的存储服务。客户只需花极少的钱就可以获得一个几乎无限的存储空间，可以随时根据需要调整对资源的占用，并只需为真正使用的资源付费。

1.2 先决条件

您需要先获得以下项，才能使用 OOS SDK Java：

- 注册并开通了一个 OOS 账户
- 已经安装 JDK 8 及以上软件

1.3 获取 OOS 凭证

AccessKeyId 和 SecretKey 是您访问 OOS 的密钥，OOS 会通过它来验证您的资源请求，请妥善保管。关于 AccessKeyId 和 SecretKey 的介绍，请阅读《OOS 开发者文档-V5》。

1.4 OOS 开发者文档

使用 Java SDK 前务必通读《OOS 开发者文档-V5》，以便掌握各请求参数和响应参数的使用方法。

1.5 安装 Java SDK

1. 下载 oos-java-sdk-6.5.4.zip 压缩包,并解压。
2. 将 oos-java-sdk-6.5.4.jar 以及 third-party 文件夹下的所有文件拷贝到项目中。
3. 在 IDE 中导入已添加的 jar 包，以 Eclipse 为例：在 Eclipse 中选择工程，右击选择 Properties > Java Build Path > Add JARs，选中在第 2 步已拷贝的所有 JAR 文件。

2 SDK 代码示例

该部分主要介绍的内容是 OOS 对外开放的 SDK 接口，当用户发送请求给 OOS 时，可以通过签名认证的方式请求，也可以匿名访问。

OOS 的服务端地址参见《OOS 开发者文档-V5》中的 EndPoint 列表。

在使用 Java SDK 前，需要先设置相应的 key 和 EndPoint，然后创建一个 AmazonS3Client，AmazonS3Client 是用来发起请求和接收响应的处理的客户端。

2.1 设置 key 和 endpoint

以下代码示例用于进行 key 和 endpoint 的设置：

```
package cn.ctyun.test;
public class TestConfig {
    //OOS_ACCESS_ID
    public static final String OOS_ACCESS_ID    = "your AccessKey";
    //OOS_ACCESS_KEY
    public static final String OOS_ACCESS_KEY    = "your SecretKey";
    //OOS_ENDPOINT
    public static final String OOS_ENDPOINT    = "your point";
    //OOS_ENDPOINT_ACCESS
    public static final String OOS_ENDPOINT_ACCESS    = "your iam point";
}
```

2.2 创建 AmazonS3Client 对象

以下代码是用来创建 AmazonS3Client 对象：

```
public static AmazonS3 getAmazonS3(){
        ClientConfiguration clientConfig = new ClientConfiguration();
        //设置连接的超时时间，单位毫秒
        clientConfig.setConnectionTimeout(30*1000);
        //设置 socket 超时时间，单位毫秒
```

```
        clientConfig.setSocketTimeout(30*1000) ;
        clientConfig.setProtocol(Protocol.HTTP);          //设置 http

//设置 V4 签名算法中负载是否参与签名，关于签名部分请参看《OOS 开发者文档》
        S3ClientOptions options = new S3ClientOptions();
        options.setPayloadSigningEnabled(true);
        // 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);
        return oosClient;
    }
```

2.3 关于 Service 的操作

2.3.1 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

```
    public static void listBuckets(AmazonS3 oosClient){
        List<Bucket> listBuckets = oosClient.listBuckets();
        for (Bucket bucketInfo : listBuckets) {
            System.out.println("listBuckets:"
```

```
                    + "\t Name:" + bucketInfo.getName()
                    + "\t CreationDate:" + bucketInfo.getCreationDate()
                    + "\t Owner:"+ bucketInfo.getOwner());
        }
    }
```

2.4 关于 Bucket 的操作

2.4.1 PUT Bucket

Put Bucket 用来创建一个 Bucket。只有已注册 OOS 账户才能创建 Bucekt，匿名请求无效，创建 Bucket 的用户将是 Bucket 的拥有者。

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 名称中不能包含双横线（--）、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

```
public static void putBucket(AmazonS3 oosClient){
    myBucketName = "javatest6";
    Bucket bucketInfo = oosClient.createBucket(myBucketName);
     System.out.println("putBucket:"
```

```
                        + "\t Name:" + bucketInfo.getName()
                        + "\t CreationDate:" + bucketInfo.getCreationDate()
                        + "\t Owner:"+ bucketInfo.getOwner())
}
```

2.4.2 GET Bucket ACL

Get Bucket ACL 操作用来获取 Bucket 的 ACL 信息，用户必须对该 Bucket 有读权限才可以执行此操作。Bucket 的 ACL 信息在创建的时候可以设定。

示例代码

```
public static void getBucketAcl(AmazonS3 oosClient){
      AccessControlList list = oosClient.getBucketAcl(myBucketName);
}
```

2.4.3 GET Bucket (List Objects)

Get Bucket（List Objects）操作返回 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。

要执行该操作，需要对操作的 Bucket 拥有读权限。

示例代码

```
public static void listObjects(AmazonS3 oosClient){
      ListObjectsRequest listObjectsRequest    = new ListObjectsRequest();
      listObjectsRequest.setBucketName(myBucketName);
      listObjectsRequest.setMaxKeys(100);
      ObjectListing list = oosClient.listObjects(listObjectsRequest);
 }
```

2.4.4 Delete Bucket

Delete Bucket 操作用来删除 Bucket。执行该操作前，需要确保该 Bucket 中的 Object 已经被全部删除。

示例代码

```
public static void deleteBucket(AmazonS3 oosClient){
        oosClient.deleteBucket(myBucketName);
 }
```

2.4.5 PUT Bucket Policy

在 PUT Bucket 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 Policy，此操作会替换原有 Policy。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

其中的 javatest6 为用户自己的 Bucket 名字。

```
public static void setBucketPolicy(AmazonS3 oosClient){
        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:::javatest6/*',     "
                    +"                      'Condition':{     "
                    +"                          'StringLike':{     "
                    +"
'aws:Referer':[     "
                    +"
'http://www.mysite.com/*'    "
                    +"                                  ]    "
```

```
                                +"                        }  "
                                +"                    }  "
                                +"                }  "
                                 +"            ]  "
                            +"        }  ";

        oosClient.setBucketPolicy(myBucketName,buffer);
    }
```

2.4.6 GET Bucket Policy

在 GET Bucket 操作的 url 中加上 Policy，可以获得指定 Bucket 的 Policy。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。如果 Bucket 没有 Policy，返回 404，NoSuchPolicy 错误。

示例代码

```
public static void getBucketPolicy(AmazonS3 oosClient){
        oosClient.getBucketPolicy(myBucketName);
    }
```

2.4.7 DELETE Bucket Policy

在 Delete Bucket 操作的 url 中加上 policy，可以删除指定 Bucket 的 Policy。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。如果 Bucket 没有 Policy，返回 204 NoContent。

示例代码

```
public static void deleteBucketPolicy(AmazonS3 oosClient){
        oosClient.deleteBucketPolicy(myBucketName);
    }
```

2.4.8 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。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。

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

```
public static void putBucketWebsite(AmazonS3 oosClient){
        BucketWebsiteConfiguration configuration = new
                BucketWebsiteConfiguration();
        configuration.setIndexDocumentSuffix("index.html");
        configuration.setErrorDocument("error.html");

oosClient.setBucketWebsiteConfiguration(myBucketName,configuration);
    }
```

2.4.9 GET Bucket WebSite

在 GET Bucket 的 url 中加上 website，可以获得指定 Bucket 的 website。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
public static void getBucketWebsite(AmazonS3 oosClient){
        oosClient.getBucketWebsiteConfiguration(myBucketName);
    }
```

2.4.10 DELETE Bucket WebSite

在 Delete Bucket 操作的 url 中加上 website，可以删除指定 Bucket 的 website。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。如果 Bucket 没有 website，返回 200 OK。

示例代码

```
public static void deleteBucketWebsite(AmazonS3 oosClient){
    oosClient.deleteBucketWebsiteConfiguration(myBucketName);
}
```

2.4.11 List Multipart Uploads

该接口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。

响应中最多返回 1000 个分片上传过程的信息，它既是响应能返回的最大分片上传过程数目，也是请求的默认值。用户也可以通过设置 max-uploads 参数来限制响应中的分片上传过程数目。如果当前的分片上传过程数超出了这个值，则响应中会包含一个值为 true 的 IsTruncated 元素。如果用户要列出多于这个值的分片上传过程信息，则需要继续调用 List Multipart Uploads 请求，并在请求中设置 key-marker 和 upload-id-marker 参数。

在响应体中，分片上传过程的信息通过 key 来排序。如果用户的应用程序中启动了多个使用同一 key 对象开头的分片上传过程，那么响应体中分片上传过程首先是通过 key 来排序，在相同 key 的分片上传内部则是按上传启动的起始时间的升序来进行排列。

示例代码

```
public static void listMultipartUploads(AmazonS3 oosClient){
    ListMultipartUploadsRequest listMultipartUploadsRequest =
        new ListMultipartUploadsRequest(myBucketName);
    oosClient.listMultipartUploads(listMultipartUploadsRequest);
}
```

2.4.12 PUT Bucket Logging

在 PUT Bucket 操作的 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。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
public static void putBacketLogging(AmazonS3 oosClient){
    BucketLoggingConfiguration loggingConfiguration =
        new BucketLoggingConfiguration();

    loggingConfiguration.setLogFilePrefix("prefix-1");
    loggingConfiguration.setDestinationBucketName(myBucketName);

    SetBucketLoggingConfigurationRequest
setBucketLoggingConfigurationRequest
        = new SetBucketLoggingConfigurationRequest(myBucketName,
            loggingConfiguration);

oosClient.setBucketLoggingConfiguration(setBucketLoggingConfigurationRequest);
}
```

2.4.13 GET Bucket Logging

在 GET Bucket 操作的 url 中加上 logging，可以获得指定 Bucket 的 logging。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
public static void getBacketLogging(AmazonS3 oosClient){
    BucketLoggingConfiguration logging =
            oosClient.getBucketLoggingConfiguration(myBucketName);
}
```

2.4.14 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

```
public static void doesBucketExist(AmazonS3 oosClient){
        oosClient.doesBucketExist(myBucketName);
}
```

2.4.15 PUT Bucket Lifecycle

存储在 OOS 中的对象有时需要有生命周期。比如，用户可能上传了一些周期性的日志文件到 bucket 中，一段时间后，用户可能不需要这些日志对象了。可以使用对象到期功能来指定 Bucket 中对象的生命周期，生命周期过后，对象自动被删除。

示例代码

```
public static void putBucketLifecycle(AmazonS3 oosClient){
        BucketLifecycleConfiguration bucketLifecycleConfiguration =
            new BucketLifecycleConfiguration();

        List<Rule> rules = new ArrayList<Rule>();
        Rule rule1 = new Rule();
        rule1.setId("r1");
        rule1.setPrefix("logs");
        rule1.setStatus("Enabled");
        rule1.setExpirationInDays(30);
        rules.add(rule1);

        bucketLifecycleConfiguration.setRules(rules);

        oosClient.setBucketLifecycleConfiguration(
            myBucketName,bucketLifecycleConfiguration);
}
```

2.4.16 GET Bucket Lifecycle

此接口用于返回配置的 bucket 生命周期。

示例代码

```
public static void getBucketLifecycle(AmazonS3 oosClient){
    oosClient.getBucketLifecycleConfiguration(myBucketName);
}
```

2.4.17 DELETE Bucket Lifecycle

此接口用于删除配置的 Bucket 生命周期，OOS 将会删除指定 Bucket 的所有生命周期配置规则。删除 Bucket 的生命规则之后，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

```
public static void deleteBucketLifecycle(AmazonS3 oosClient){
    oosClient.deleteBucketLifecycleConfiguration(myBucketName);
}
```

2.4.18 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

```
public static void putBucketCors(AmazonS3 oosClient){
    BucketCrossOriginConfiguration bucketCrossOriginConfiguration =
            new BucketCrossOriginConfiguration();
    CORSRule rule = new CORSRule();
    List<CORSRule.AllowedMethods> allowedMethods = new
```

```
            ArrayList<CORSRule.AllowedMethods>();
        allowedMethods.add(CORSRule.AllowedMethods.GET);
        allowedMethods.add(CORSRule.AllowedMethods.POST);
        List<String> allowedOrigins = new ArrayList();
        allowedOrigins.add("a");
        allowedOrigins.add("b");
        int maxAgeSeconds = 1000;
        List<String> exposedHeaders = new ArrayList();
        List<String> allowedHeaders = new ArrayList();
        exposedHeaders.add("e1");
        exposedHeaders.add("e2");
        allowedHeaders.add("a1");
        allowedHeaders.add("a2");
        rule.setId("myfirstRuleId");
        rule.setAllowedMethods(allowedMethods);
        rule.setAllowedOrigins(allowedOrigins);
        rule.setExposedHeaders(exposedHeaders);
        rule.setAllowedHeaders(allowedHeaders);
        rule.setMaxAgeSeconds(maxAgeSeconds);
        List<CORSRule> rules = new ArrayList<CORSRule>();
        rules.add(rule);
        bucketCrossOriginConfiguration.setRules(rules);

        oosClient.setBucketCrossOriginConfiguration(
                myBucketName,bucketCrossOriginConfiguration);
    }
```

2.4.19 GET Bucket cors

Get Bucket cors 返回 Bucket 的跨域配置信息。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
    public static void getBucketCors(AmazonS3 oosClient){
        BucketCrossOriginConfiguration cors =
            oosClient.getBucketCrossOriginConfiguration(myBucketName);
    }
```

2.4.20 DELETE Bucket cors

Delete Bucket cors 操作可以删除 Bucket 的跨域配置信息。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
public static void deleteBucketCors(AmazonS3 oosClient){
    oosClient.deleteBucketCrossOriginConfiguration(myBucketName);
}
```

2.4.21 PUT Bucket Trigger

在 PUT Bucket 操作的 url 中加上 Trigger，可以进行添加 Trigger 的操作，即添加一个向异地资源池同步的触发器。当客户端向本地资源池的 Bucket 上传对象时，OOS 可以根据配置的策略，自动将对象同步到异地资源池中。一个 Bucket 可以配置多个触发器，但只能有一个是默认的触发器。如果客户端要使用非默认的触发器上传对象，需要在 PUT Object 时，加上请求头 x-ctyun-trigger，值是指定的 *TriggerName*。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
public static void putBucketTrigger(AmazonS3 oosClient){
    CtyunTriggerMetadata triggerMetadata = new CtyunTriggerMetadata();
      triggerMetadata.setTriggerName("theTriggerNameForTwoRemoteSite");
      List<CtyunRemoteSite> remoteSites = new ArrayList<>();
      CtyunRemoteSite remoteSite1 = new CtyunRemoteSite();

      remoteSite1.setRemoteAK("your remoteAK1");
      remoteSite1.setRemoteSK("your remoteSK1");
      remoteSite1.setRemoteBucketName("bucket1");
      remoteSite1.setRemoteEndPoint("http://oos-js.ctyunapi.cn");
      remoteSites.add(remoteSite1);

      CtyunRemoteSite remoteSite2 = new CtyunRemoteSite();
      remoteSite2.setRemoteAK("your remoteAK2");
```

```
    remoteSite2.setRemoteSK("your remoteSK2");
    remoteSite2.setRemoteBucketName("bucket2");
    remoteSite2.setRemoteEndPoint("http://oos-nm2.ctyunapi.cn");
    remoteSites.add(remoteSite2);

  triggerMetadata.setDefault(true);
  triggerMetadata.setRemoteSites(remoteSites);

  List<CtyunTriggerMetadata> triggers = new ArrayList<>();
  triggers.add(triggerMetadata);

  CtyunBucketTriggerConfiguration triggerConfiguration =
        new CtyunBucketTriggerConfiguration();
    triggerConfiguration.setTriggers(triggers);

  CtyunSetBucketTriggerConfigurationRequest
      triggerConfigurationRequest =        new
        CtyunSetBucketTriggerConfigurationRequest(myBucketName,
        triggerConfiguration);

        oosClient.ctyunSetBucketTrigger(triggerConfigurationRequest);
   }
```

2.4.22 GET Bucket Trigger

在 GET Bucket 操作的 url 中加上 Trigger，可以查询指定 Bucket 中配置的所有 Trigger。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
  public static void getBucketTrigger(AmazonS3 oosClient){
        oosClient.ctyunGetBucketTrigger(myBucketName);
   }
```

2.4.23 DELETE Bucket Trigger

在 Delete Bucket 操作的 url 中加上 Trigger，可以删除 Bucket 中的指定 Trigger。只有 Bucket 的拥有者才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
public static void getBucketTrigger(AmazonS3 oosClient){
    oosClient.ctyunDeleteBucketTrigger(myBucketName,
    "theTriggerNameForTwoRemoteSite");
}
```

2.5 关于 Object 的操作

2.5.1 PUT Object

Put Object 操作用来向指定 Bucket 中添加一个对象，要求发送请求者对该 Bucket 拥有写权限，用户必须添加完整的对象。

示例代码

```
public static void putObject(AmazonS3 oosClient){
        String objectInBucketKey = "objectInBucketKey1";
        File testFile = new File("D:\\Temp\\aaa.txt");
        PutObjectRequest request = new
              PutObjectRequest(myBucketName,objectInBucketKey,testFile);

        ObjectMetadata metadata = new ObjectMetadata();

        CtyunBucketDataLocation dataLocation =
           new CtyunBucketDataLocation();
        dataLocation.setType(
                CtyunBucketDataLocation.CtyunBucketDataType.Specified);

        List<String> dataRegions = new ArrayList<String>();
        dataRegions.add("QingDao");
        dataRegions.add("ShenZhen");

        dataLocation.setDataRegions(dataRegions);

        metadata.setDataLocation(dataLocation);

        request.setMetadata(metadata);
        oosClient.putObject(request);
    }
```

2.5.2 GET Object

GET Object 操作用来检索在 OOS 中的对象信息，用户必须对 Object 所在 Bucket 拥有读权限才能执行此操作。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

```
public static void getObject(AmazonS3 oosClient){
    String objectInBucketKey = "objectInBucketKey1";

    S3Object object =
        oosClient.getObject(myBucketName,objectInBucketKey);
    object.getObjectMetadata();
}
```

2.5.3 DELETE Object

Delete Object 操作用来移除指定的对象。对 Object 所在的 Bucket 拥有写权限的用户才能执行此操作。

示例代码

```
public static void deleteObject(AmazonS3 oosClient){
    String objectInBucketKey = "objectInBucketKey1";
    oosClient.deleteObject(myBucketName,objectInBucketKey);
}
```

2.5.4 PUT Object - Copy

PUT Object - Copy 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

```
public static void copyObject(AmazonS3 oosClient){
    String sourceBucketName = myBucketName;
    String sourceKey = "objectInBucketKey1";
    String destinationBucketName = myBucketName;
    String destinationKey = "objectInBucketKey2";
    CopyObjectRequest request = new CopyObjectRequest(
      sourceBucketName,sourceKey,destinationBucketName,destinationKey);
```

```
        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);

        request.setNewObjectMetadata(metadata);

        oosClient.copyObject(request);
}
```

2.5.5 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（见 Upload Part）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

```
    public static void initiateMultipartUpload(AmazonS3 oosClient){
        String objectInBucketKey2 = "objectInBucketKey2";

        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);

        InitiateMultipartUploadRequest request = new
                InitiateMultipartUploadRequest(myBucketName,
                objectInBucketKey2, metadata);
        InitiateMultipartUploadResult res =
            oosClient.initiateMultipartUpload(request);
        String uploadID = res.getUploadId();

        System.out.println("=========uploadID:\t :" + uploadID);
    }
```

2.5.6 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。

```
public static void uploadPart(AmazonS3 oosClient){
    String objectInBucketKey2 = "objectInBucketKey2";

    UploadPartRequest request = new
            UploadPartRequest().withUploadId("1552285359321501497");
    File testFile = new File("D:\\Tmp\\A3.txt");
    request.setFile(testFile);
    request.setBucketName(myBucketName);
    request.setKey(objectInBucketKey2);
    request.setPartNumber(3);
    request.setPartSize(testFile.length());

    UploadPartResult res = oosClient.uploadPart(request);
    String eTag = res.getETag();

    System.out.println("=========eTag:\t :" + eTag);
}
```

2.5.7 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

```
public static void completeMultipartUpload(AmazonS3 oosClient){
    String objectInBucketKey2 = "objectInBucketKey2";

    List<PartETag> partETags = new ArrayList<PartETag>();
    partETags.add(new PartETag(1,"f6a6263167c92de8644ac998b3c4e4d1"));
    partETags.add(new PartETag(2,"830fea339a3aeee9913560b751d3afeb"));
    //partETags.add(new PartETag(3,"fa930e80bba6c56c19cc2c03975719c0"));

    CompleteMultipartUploadRequest request = new
      CompleteMultipartUploadRequest(                    myBucketName,obje
            ctInBucketKey2,"1552285359321501497"
      ,partETags);

    CompleteMultipartUploadResult result =
          oosClient.completeMultipartUpload(request);
}
```

2.5.8 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

```
public static void abortMultipartUpload(AmazonS3 oosClient){
    String objectInBucketKey2 = "objectInBucketKey2";

    AbortMultipartUploadRequest request =new AbortMultipartUploadRequest(
            myBucketName,objectInBucketKey2,"1552285359321501497");

    oosClient.abortMultipartUpload(request);
}
```

2.5.9 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

```
public static void listParts(AmazonS3 oosClient){
    String objectInBucketKey2 = "objectInBucketKey2";

    ListPartsRequest request = new ListPartsRequest(
            myBucketName,objectInBucketKey2,"1552285359321501497");

    PartListing res = oosClient.listParts(request);
    int size = res.getParts().size();
```

```
        System.out.println("=========size:\t :" + size);
    }
```

2.5.10 Copy Part

Copy Part 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

```
    public static void copyPart(AmazonS3 oosClient){
        CopyPartRequest request = new CopyPartRequest();
        request.setSourceBucketName(myBucketName);
        request.setSourceKey("objectInBucketKey1");

        request.setDestinationBucketName( "destinationbacket");
        request.setDestinationKey("objectInBucketKey2");

//设置 Copy Part 范围（如果要拷贝整个 object，不需要设置）。
//如果设置范围，必须同时设置 request.setFirstByte 和 request.setLastByte。
        request.setFirstByte(0L);   //Copy Part 起始字段。
        request.setLastByte(100L);   //Copy Part 结束字段。

        request.setPartNumber(2);

        request.setUploadId("15522853593215014 97");

        CopyPartResult res = oosClient.copyPart(request);
    }
```

2.5.11 Delete Multiple Objects

Delete Multiple Objects 操作用来批量删除 Object。

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，你需要指定要删除的 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

```
public static void deleteObjects(AmazonS3 oosClient){
        String destBacketName = "destinationbacket";
        String objectInBucketKey1 = "objectInBucketKey1";
        String objectInBucketKey2 = "objectInBucketKey2";

        DeleteObjectsRequest request =
                new DeleteObjectsRequest(myBucketName);
        DeleteObjectsRequest.KeyVersion key1 = new
                DeleteObjectsRequest.KeyVersion(objectInBucketKey1,null);
        DeleteObjectsRequest.KeyVersion key2 = new
                DeleteObjectsRequest.KeyVersion(objectInBucketKey2,null);
        List<DeleteObjectsRequest.KeyVersion> keys = new
                ArrayList<DeleteObjectsRequest.KeyVersion>();
        keys.add(key1);
        keys.add(key2);
        request.withKeys(keys);

        //返回的响应信息全面信息还是简明信息,设置 true 为简明信息
        request.setQuiet(true);

        DeleteObjectsResult res = oosClient.deleteObjects(request);
}
```

2.5.12 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

```
public static void generatePresignedUrl(AmazonS3 oosClient){
        String objectInBucketKey1 = "objectInBucketKey1";
        GeneratePresignedUrlRequest request = new
           GeneratePresignedUrlRequest(myBucketName,objectInBucketKey1);

        //2019/03/10
        java.util.Date expireDate = new java.util.Date(119,02,10);
        request.setExpiration(expireDate);
        URL url = oosClient.generatePresignedUrl(request);
        System.out.println(url.toString());
}
```

2.5.13 HEAD Object

Head 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

```
public static void headObject(AmazonS3 oosClient){
        String objectInBucketKey1 = "objectInBucketKey1";

        ObjectMetadata    metadata =
              oosClient.getObjectMetadata(myBucketName,objectInBucketKey1);
}
```

2.6 关于 AccessKey 的操作

需要设置 IAM 的 Endpoint,示例如下：

```
oosClient.setEndpoint(TestConfig.OOS_ENDPOINT_ACCESS);
```

各地域的 IAM Endpoint 如下：

- 北京2：oos-bj2-iam.ctyunapi.cn
- 内蒙：oos-nm2-iam.ctyunapi.cn
- 长沙：oos-hncs-iam.ctyunapi.cn
- 西安：oos-snxa-iam.ctyunapi.cn
- 杭州：oos-hz-iam.ctyunapi.cn
- 江苏：oos-js-iam.ctyunapi.cn
- 广州：oos-gz-iam.ctyunapi.cn
- 北京：oos-hq-bj-iam.ctyunapi.cn
- 上海：oos-hq-sh-iam.ctyunapi.cn

2.6.1 CreateAccessKey

创建一对普通的 AccessKey 和 SecretKey，默认的状态是 Active。只有主 Key 才能执行此操作。

为保证账户的安全，SecretKey 只在创建的时候会被显示。需要将 Key 保存起来。如果 SecretKey 丢失了，可以删除 AccessKey，并创建一对新的 Key。默认情况下，每个账号最多创建 10 个 AccessKey。

示例代码

```
public static void createAccessKey(AmazonS3 oosClient){
    CreateAccessKeyResult result = oosClient.ctyunCreateAccessKey();
}
```

2.6.2 DeleteAccessKey

删除一对普通的 AccessKey 和 SecretKey。只有主 Key 才能执行此操作。

示例代码

```
public static void deleteAccessKey(AmazonS3 oosClient){
    DeleteAccessKeyRequest request = new DeleteAccessKeyRequest();
    request.setAccessKeyId("ebqa339c9aa8bdb3a575");
    oosClient.ctyunDeleteAccessKey(request);
```

```
    }
```

2.6.3 UpdateAccessKey

UpdateAccessKey 操作用来更新普通的 AccessKey 的状态，或将普通 Key 设置成为主 Key，或者将主 Key 设置为普通 Key。只有主 key 才能执行此操作。

示例代码

```
public static void updateAccessKey(AmazonS3 oosClient){
    UpdateAccessKeyRequest updateAccessKeyRequest = new
        UpdateAccessKeyRequest();
    updateAccessKeyRequest.setAccessKeyId("eb2a339c9aa8bdq3a575");
    updateAccessKeyRequest.setStatus("Active");
    oosClient.ctyunUpdateAccessKey(updateAccessKeyRequest);
}
```

2.6.4 ListAccessKey

ListAccessKey 操作用来列出账号下的主 Key 和普通 Key。只有主 Key 才能执行此操作。可以通过 MaxItems 参数指定返回的结果数量，默认返回 100 个 Key。可以通过 Marker 参数设置返回的起始位置，该参数可以从前一次请求的响应体中获得。为保证账号安全，ListAccessKey 操作时，不会返回 SecretKey。

示例代码

```
public static void listAccessKey(AmazonS3 oosClient){
    ListAccessKeysRequest listAccessKeysRequest = new
        ListAccessKeysRequest();
    ListAccessKeysResult result =
        oosClient.ctyunListAccessKeys(listAccessKeysRequest);
}
```

2.7 错误处理

2.7.1 错误响应

当请求出错时，OOS 返回以下格式的错误响应信息。

```
<?xml version="1.0" encoding="UTF-8"?>
<Error>
    <Code>NoSuchKey</Code>
    <Message>The resource you requested does not exist</Message>
    <Resource>/mybucket/myfoto.jpg</Resource>
    <RequestId>4442587FB7D0A2F9</RequestId>
</Error>
```

响应元素：

名称	描述
Code	错误码是描述出错信息的字符串。详细的错误码信息可以参见错误码列表。
Message	错误信息的简单描述。
RequestId	请求的 ID。
Resource	出错相关的 Bucket 或 Object 信息。

2.7.2 错误码列表

错误码	描述	HTTP 状态码
NotVerify	用户未激活。	403 Forbidden
InvalidArgument	无效的请求参数。	400
BucketAlreadExists	请求的 Bucket name 已经存在，请选择一个不同的名字。	409 Conflict
BucketNotEmpty	试图删除的 Bucket 非空。	409 Conflict
InvalidAccessKeyId	AccessKetId 不存在	403 Forbidden
InvalidBucketName	指定的 Bucket 无效。	400 Bad Request
InvalidURI	无法解析指定的 URI。	400 Bad Request
MethodNotAllowed	指定的方法不允许操作该资源。	405 Method Not Allowed
MissingContentLength	在 HTTP 头中必须提供 ContentLength。	411 length required
RequestTimeTooSkewed	请求的时间和服务器时间相差 15 分钟以上。	403 Bad Request
NoSuchBucket	指定的 Bucket 不存在。	404 Not Found
NoSuchKey	指定对象的 Key 不存在。	404 Not Found

SignatureDoesNotMatch	计算出的请求中的签名与用户提供的签名不一致，需检查 OOS 账户安全。	403 Forbidden
TooManyBuckets	Bucket 个数超出限制。	400 Bad Request
PutObjectTooFast	并发写同一个对象或分段片段时冲突。	400 Bad Request
MalformedXML	XML 格式不合法。	400 Bad Request
CanNotModifyMetadataLocation	索引位置不允许修改。	400 Bad Request
InvalidLocationConstraint	用户提供的配置索引位置和数据位置的 xml 格式不合法。	400 Bad Request
InvalidRequest	IP 白名单不合法。	400 Bad Request
InvalidRequest	IP 白名单个数超过限制。	400 Bad Request
Internal Error	OOS 服务错误，请重试。	500 InternalServerError
SlowDown	OOS 服务繁忙，请重试。	503 Service Unavailable