天翼云
State Cloud

对象存储（经典版）I 型

(Object-Oriented Storage,OOS)

Java SDK 开发者指南 V6

天翼云科技有限公司

目录

1 前言

对象存储（经典版）Ⅰ型（Object-Oriented Storage，OOS）是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。目前 OOS 提供以下四种服务：对象存储（OOS）、统计分析、操作跟踪、访问控制。

1.1 OOS 服务

OOS 服务为对象存储（经典版）Ⅰ型的核心服务，主要包括 Bucket 操作管理以及 object 操作管理，为 OOS 基础服务。

1.2 统计分析（Statistics API）

统计分析对 OOS 账户内的资源数据进行统计分析，以图、表的方式直观的将账户内的资源情况、变化等信息展示给用户，以便用户及时了解到账户内的资源使用情况及实际业务情况，并采取对应的措施，用户可以通过管理控制台或 SDK 获取统计数据。

1.3 操作跟踪（CloudTrail）

CloudTrail 用于记录 OOS 账户的管理事件，并将产生的日志文件保存到指定的 OOS 存储桶中。记录的信息包括用户的身份，API 调用的开始时间，源 IP 地址，请求参数以及服务返回的响应元素等。

1.4 访问控制（IAM）

IAM（Identity and Access Management）是 OOS 为用户提供的用户身份管理与访问控制服务。账号管理员可以创建、管理子用户账号，并可以控制这些子用户账号对账号内资源具有的操作权限。管理人员可以按需为用户分配最小权限，从而避免与其他用户共享资源使用、访问密钥等，降低账号的信息安全风险。

2 使用条件

2.1 先决条件

用户需要具备以下条件，然后才能够使用 OOS SDK Java 版本：

- 已注册天翼云账户，开通 OOS 服务。
- 安装 JDK 8 及以上版本。

2.2 获取访问凭证

AccessKeyId 和 SecretAccessKey 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 SecretAccessKey 的介绍，请阅读《OOS 开发者文档》。

2.3 开发者文档

完整的通用的开发者文档的下载地址为《OOS 开发者文档》，开发者在 SDK 前请务必首先阅读《OOS 开发者文档》，以便掌握各请求参数和响应参数的使用方法。

2.4 下载及安装

从官方渠道下载 oos-java-sdk-6.5.8.zip 压缩包，放到相应位置后并解压。解压后文件与目录为三个部分，文件【oos-java-sdk-6.5.8.jar】为 sdk 的 jar 包，目录【third-party】的 jar 文件为 sdk 所使用的第三方的 jar 包，目录【example】下的文件为 sdk 的使用示例代码。

用户需要在自己使用的工程中导入已解压的所有 jar 包，才能使用 sdk。如用户使用的是 Eclipse，则在 Eclipse 中选择工程，右击选择 Properties > Java Build Path > Add JARs，选中已解压的所有 JAR 文件。如用户使用的是 Idea,则使用菜单 File>Project Structure > Project Settings > Libraries 添加已解压的所有 JAR 文件，并让 Module 使用该库。

3 SDK 使用设置

3.1 基本设置

使用 sdk 访问 OOS 的服务，需要设置正确的 AccessKeyId、SecretAccessKey 和服务端地址 endpoint，所有的服务可以使用同一 key 凭证来进行访问，但不同的服务需要使用不同的 endpoint 进行访问。OOS 的服务端地址请参见《OOS 开发者文档》。

以下代码示例用于进行 AccessKeyId、SecretAccessKey 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  = "https://oos-cn.ctyunapi.cn";

    //OOS_ENDPOINT_MANAGEMENT
    public static final String OOS_ENDPOINT_MANAGEMENT  = "https://oos-cn-mg.ctyunapi.cn";

    //OOS_ENDPOINT_TRAIL
    public static final String OOS_ENDPOINT_TRAIL  = "https://oos-cn-cloudtrail.ctyunapi.cn";

    //OOS_ENDPOINT_ACCESS
    public static final String OOS_ENDPOINT_ACCESS  = "https://oos-cn-iam.ctyunapi.cn";
}
```

参数说明

参数	描述	是否必须
OOS_ACCESS_ID	AccessKey。	是
OOS_ACCESS_KEY	SecretAccessKey。	是

OOS_ENDPOINT	OOS 域名如 https://oos-cn.ctyunapi.cn。	是
OOS_ENDPOINT_MANAGEMENT	统计域名。	是
OOS_ENDPOINT_TRAIL	操作跟踪域名。	是
OOS_ENDPOINT_ACCESS	IAM 域名。	是
ENABLE_S3_SIGV4_SYSTEM_PROPERTY	使能 S3 V4 签名算法。	是
Protocol	使用的协议，枚举值： ● Protocol.***HTTP*** ● Protocol.***HTTPS***	是

3.2 创建 Client 对象

在使用服务器前，需创建 Client 对象，各个服务创建 Client 对象的代码不同，详细示例见各章节。

4 OOS 服务代码示例

OOS 的服务代码示例是 example 目录的子目录：src\cn\ctyun\test\OOSTest.java。

创建 AmazonS3Client 对象示例：

```
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;
}
```

4.1 关于 Service 的操作

4.1.1 GET Service(List Buckets)

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：只有验证用户可以调用该接口，匿名用户不能调用该接口。

示例代码

```
private static void listBuckets(AmazonS3 oosClient) {
       try {
           List<Bucket> listBuckets = oosClient.listBuckets();
           // Print the Bucket information in the return body to the console
           for (Bucket bucketInfo : listBuckets) {
               System.out.println("listBuckets:"
                       + "\n Name:" + bucketInfo.getName()
                       + "\n CreationDate:" + bucketInfo.getCreationDate()
                       + "\n Owner:" + bucketInfo.getOwner());
           }
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.1.2 GET Regions

获取资源池中的索引位置和数据位置列表。

示例代码

```
private static void getRegions(AmazonS3 oosClient) {
       try {
           CtyunGetRegionsResult region = oosClient.ctyunGetRegions();
           List<String> metadataRegions = region.getMetadataRegions();
           List<String> dataRegions = region.getDataRegions();
           if (metadataRegions != null) {
               for (String metadataRegion : metadataRegions) {
                   System.out.println("getRegions:"
                           + "\n metadataRegion:" + metadataRegion);
               }
           }
           if (dataRegions != null) {
               for (String dataRegion : region.getDataRegions()) {
                   System.out.println("getRegions:"
                           + "\n dataRegion:" + dataRegion);
               }
           }
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2 关于 Bucket 的操作

4.2.1 PUT Bucket

此操作用来创建一个新的 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

```
private static void putBucket(AmazonS3 oosClient) {
       try {
           CreateBucketRequest request = new CreateBucketRequest(BUCKET_NAME);
           // Optional
          // Set Bucket's ACL (Access Control List): private; public-read; public-read-write
           request.setCannedAcl(CannedAccessControlList.Private);
           // Optional
           // Set the data location of the Bucket
           CtyunBucketDataLocation dataLocation = new CtyunBucketDataLocation();
           // Type of data location: Local | Specified
           dataLocation.setType(CtyunBucketDataLocation.CtyunBucketDataType.Specified);
           // Specified data location
           ArrayList<String> dataRegions = new ArrayList<>();
           dataRegions.add("QingDao");
```

```
            dataLocation.setDataRegions(dataRegions);
            request.setDataLocation(dataLocation);
            // Optional
            // Set the index location of the Bucket
            request.setMetadataLocation("QingDao");
            Bucket bucketInfo = oosClient.createBucket(request);
            System.out.println("putBucket: "
                    + "\n BucketName: " + bucketInfo.getName() + " create success!");
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.2.2 GET Bucket location

此操作用来获取 Bucket 的索引位置和数据位置，只有根用户和拥有 GET Bucket Location 权限的子用户才能执行此操作。

示例代码

```
private static void getBucketLocation(AmazonS3 oosClient) {
       try {
           CtyunGetBucketLocationResult locationResult =
oosClient.getBucketLocation(BUCKET_NAME);
           System.out.println("getBucketLocation: "
                  + "\n MetaRegion: " + locationResult.getMetaRegion()
                  + "\n DataLocation: " +
locationResult.getDataLocation().getDataRegions().toString()
           );
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.3 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

```
private static void getBucketAcl(AmazonS3 oosClient) {
       try {
           AccessControlList list = oosClient.getBucketAcl(BUCKET_NAME);
           if (list != null) {
               System.out.println("getBucketAcl: "
                       + "\n Owner: " + list.getOwner().toString()
                       + "\n Grants: " + list.getGrants().toString());
           }
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.4 Get Bucket(List Objects)

此操作用来返回 Bucket 中部分或者全部（每次最多 1000）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

```
private static void listObjects(AmazonS3 oosClient) {
       try {
           ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
           // Optional
           // Set the maximum number of entries to be returned in the response
           listObjectsRequest.setMaxKeys(1000);
           // Set the bucketName
           listObjectsRequest.setBucketName(BUCKET_NAME);
           // Optional
           // Delimiter, a character used to group keywords, only supports "/"
//           listObjectsRequest.setDelimiter("/");
           // Optional
           // Used to restrict the results returned, which must start with this prefix
//           listObjectsRequest.setPrefix("your_prefix");
           ObjectListing list = oosClient.listObjects(listObjectsRequest);
           List<S3ObjectSummary> summaries = list.getObjectSummaries();
           if (summaries.size() == 0) {
               System.out.println("There are no objects");
               return;
           }
           for (S3ObjectSummary summary : summaries) {
               String idString = " ";
               String nameString = " ";
               String etag = summary.getETag();
               if (summary.getOwner() != null) {
                   idString = summary.getOwner().getId();
                   nameString = summary.getOwner().getDisplayName();
               }
```

```
                System.out.printf("etag:%s,key:%s,owner id:%s,display name:%s %n", etag,
summary.getKey(), idString,
                        nameString);
                System.out.println("storage-class:" + summary.getStorageClass());
            }
            // When a delimiter is defined, the return results will include
CommonPrefixes
            List<String> commonPrefixes = list.getCommonPrefixes();
            for (String commonPrefix : commonPrefixes) {
                System.out.println("CommonPrefix: " + commonPrefix);
            }
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.2.5 DELETE Bucket

该操作用来删除 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

```
private static void deleteBucket(AmazonS3 oosClient) {
       try {
           oosClient.deleteBucket(BUCKET_NAME);
           System.out.println("delete bucket success!");
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.6 PUT Bucket Policy

此操作用来设置存储桶的访问策略。如果 Bucket 已经存在了 Policy，此操作会替换原有 Policy。

注意：如果 Bucket 的属性为私有或者公共读，使用该接口配置允许任何用户可以向该 Bucket 写文件的策略时，请联系天翼云客服评估审核后开通。

示例代码

```
private static void putBucketPolicy(AmazonS3 oosClient) {
       try {
           // For specific setting rules, please refer to the developer documentation
           String buffer = "   {   "
                  + "        'Version':'2012-10-17',   "
                  + "        'Id':'http referer policy example',   "
                  + "              'Statement':[   "
                  + "            {   "
                  + "             'Sid':'Allow get requests referred by www.ctyun.cn or
ctyun.cn',   "
                  + "                'Effect':'Allow',   "
                  + "                 'Principal':{   "
                  + "                      'AWS':[   "
                  + "                        '*'"
                  + "                           ]   "
                  + "                  },   "
                  + "               'Action':'S3:GetObject',   "
                  + "               'Resource':'arn:aws:s3:::example-bucket/*',   "
                  + "               'Condition':{   "
                  + "                   'StringLike':{   "
                  + "                      'aws:Referer':[   "
                  + "                         'https://www.ctyun.cn/*',  "
                  + "                         'https://ctyun.cn/*',  "
                  + "                      ]  "
                  + "                } "
                  + "            } "
                  + "          } "
                  + "      ] "
```

```
                + "      }  ";
            oosClient.setBucketPolicy(BUCKET_NAME, buffer);
            System.out.println("put bucket policy success!");
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.2.7 GET Bucket Policy

此操作用来获取存储桶的访问策略。只有根用户和拥有 GET Bucket Policy 权限的用户才能执行此操作。

示例代码

```
private static void getBucketPolicy(AmazonS3 oosClient) {
       try {
           BucketPolicy bucketPolicy = oosClient.getBucketPolicy(BUCKET_NAME);
           System.out.println("BucketPolicy:" +
                   "\n" + bucketPolicy.getPolicyText());
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.8 DELETE Bucket Policy

此操作用来删除指定存储桶的访问策略。只有根用户和拥有 DELETE Bucket Policy 权限的子用户才能执行此操作。

示例代码

```
private static void deleteBucketPolicy(AmazonS3 oosClient) {
       try {
           oosClient.deleteBucketPolicy(BUCKET_NAME);
           System.out.println("delete bucket policy success!");
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.9 PUT Bucket Website

此操作用来配置网站托管属性。如果 Bucket 已经存在了 website，此操作会替换原有 website。只有根用户和拥有 PUT Bucket Website 权限的子用户才能执行此操作。

注意：

- 如果配置静态网站托管后，当匿名用户直接访问存储桶的域名，会将静态网站文件下载到本地。如果要实现访问静态网站时是预览网站内容，而非下载静态网站文件，静态网站域名须是存储桶绑定的已备案自定义域名，为存储桶绑定自定义域名请联系天翼云客服申请。
- OOS 自有网站托管域名不支持 HTTPS 访问，用户自定义域名支持 HTTPS 访问。如果需要支持 HTTPS 访问，请联系天翼云客服，提供域名证书，证书支持格式：crt+key 或者 PEM，请确保提供的证书在有效期内，建议证书有效期至少 1 年及以上，避免使用免费证书。
- 尽量避免目标存储桶名中带有“.”，否则通过 HTTPS 访问时可能出现客户端校验证书出错。
- 设置 Bucket 的网络配置请求消息体的上限是 10KiB。

网站托管配置步骤如下：

1）创建一个公共读属性的存储桶（Bucket）。

2）向天翼云客服提交工单，申请客户自定义域名添加白名单。

3）在域名管理中添加别名。

- 如果不使用 CDN 加速，将 Bucket 的 CNAME Record Value（*BucketName*.oos-website-cn.oos-*xx*.ctyunapi.cn）作为别名添加到域名管理系统中。
- 如果使用 CDN 加速，将 CDN 厂商提供的别名添加到域名管理系统中，然后在 CDN 回源地址中配置 OOS 侧的 CNAME Record Value，并将回源 host 配置为您的自定义域名（如 your***domain.com）。

说明：创建 Bucket 时显示的 Endpoint 为 oos-cn.ctyunapi.cn，该 Endpoint 是针对整个对象存储网络的域名，该域名在解析时，会根据用户地理位置的不同解析到不同的资源池地址。如果创建 Bucket 时有多个数据位置，系统默认选取创建时第一个有效数据位置作为 CNAME Record Value（*BucketName*.oos-website-cn.oos-

xx.ctyunapi.cn）。CNAME Record Value 可以通过控制台 Bucket 属性中的**网站**查看。如果创建 Bucket 时，只有一个数据位置可用，则在 Bucket 区域中展示的 CNAME Record Value 为 *BucketName*.oos-website-cn.oos-cn.ctyunapi.cn。所以如果使用静态网站托管，建议您根据 Bucket 区域属性中的数据位置，选择您想使用的数据位置的 CNAME Record Value 作为域名管理系统中的别名。例如您创建 Bucket 时有效数据位置为沈阳、兰州、成都、贵阳，则 Bucket 中展示的 CNAME Record Value 为 *BucketName*.oos-website-cn.oos-lnsy.ctyunapi.cn，您可以将 *BucketName*.oos-website-cn.oos-lnsy.ctyunapi.cn 作为别名，也可以将兰州、成都或者贵阳为域名的 CNAME Record Value 作为您的别名。

4) 上传文件

将网站的所有文件（html、CSS、js、图片等）上传到之前创建的 Bucket 中， 注意要保持文件之间的相对路径。

5) 配置 Bucket 网站属性：可以通过控制台或者调用本接口配置。

托管模式为当前容器

示例代码

```
private static void putBucketWebsite(AmazonS3 oosClient) {
       try {
           BucketWebsiteConfiguration config = new BucketWebsiteConfiguration();
           // If a 4XX error occurs, the specified Object will be returned
           config.setErrorDocument("error.html");
           // Set the suffix for requests accessing the website endpoint
           config.setIndexDocumentSuffix("index.html");
           // Container for redirect rules
           RoutingRule rr = new RoutingRule();
           RoutingRuleCondition condition = new RoutingRuleCondition();
           // Set the HTTP status code when redirection takes effect
           condition.setHttpErrorCodeReturnedEquals("404");
           // Set the object name prefix when redirection takes effect
           condition.setKeyPrefixEquals("/abc");
           rr.setCondition(condition);
           // Redirection information container
```

```
            RedirectRule redr = new RedirectRule();
            // The site point used when redirecting requests
            redr.setHostName("oos-cname.ctyunapi.cn");
            redr.setHttpRedirectCode("200");
            // The protocol used when redirecting requests
            redr.setProtocol("http");
            // The object name prefix used when redirecting requests
//            redr.setReplaceKeyPrefixWith("replaceKeyPrefix/abc/");
            // The object name used when redirecting requests
            redr.setReplaceKeyWith("replacekey");
            rr.setRedirect(redr);
            List<RoutingRule> routingRules = new ArrayList<>();
            routingRules.add(rr);
            config.setRoutingRules(routingRules);
            SetBucketWebsiteConfigurationRequest bucketWebsite =
                    new SetBucketWebsiteConfigurationRequest(BUCKET_NAME, config);
            oosClient.setBucketWebsiteConfiguration(bucketWebsite);
            System.out.println("put bucket website success!");
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

托管模式为重定向请求

示例代码

```
private static void putBucketWebsiteAllRequestTo(AmazonS3 oosClient) {
        try {
            BucketWebsiteConfiguration config = new BucketWebsiteConfiguration();
            // config.setErrorDocument("error.html");
            // config.setIndexDocumentSuffix("index.html");
            RedirectRule rr = new RedirectRule();
            rr.setHostName("hostname");
            rr.setProtocol("https");
            config.setRedirectAllRequestsTo(rr);
            SetBucketWebsiteConfigurationRequest bucketWebsite =
```

```
                new SetBucketWebsiteConfigurationRequest(BUCKET_NAME, config);
            oosClient.setBucketWebsiteConfiguration(bucketWebsite);
            System.out.println("put bucket website success!");
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.2.10 GET Bucket Website

此操作用来获取存储桶的网站托管属性。

示例代码

```
private static void getBucketWebsite(AmazonS3 oosClient) {
       try {
           BucketWebsiteConfiguration bucketWebsiteConfiguration =
oosClient.getBucketWebsiteConfiguration(BUCKET_NAME);
           if (bucketWebsiteConfiguration == null) {
               System.out.println("There are no bucket website.");
               return;
           }
           System.out.println("GetBucketWebsite: " +
                   "\n" + "IndexDocumentSuffix: " +
bucketWebsiteConfiguration.getIndexDocumentSuffix() +
                   "\n" + "ErrorDocument: " +
bucketWebsiteConfiguration.getErrorDocument());
           for (RoutingRule routingRule : bucketWebsiteConfiguration.getRoutingRules())
{
               if (routingRule.getRedirect() != null) {
                   RedirectRule redirectRule = routingRule.getRedirect();
                   System.out.println("HostName: " + redirectRule.getHostName() +
                           "\n" + "Protocol: " + redirectRule.getprotocol() +
                           "\n" + "ReplaceKeyPrefixWith: " +
redirectRule.getReplaceKeyPrefixWith() +
                           "\n" + "ReplaceKeyWith: " + redirectRule.getReplaceKeyWith() +
                           "\n" + "HttpRedirectCode: " +
redirectRule.getHttpRedirectCode());
               }
               if (routingRule.getCondition() != null) {
                   RoutingRuleCondition condition = routingRule.getCondition();
                   System.out.println("HttpErrorCodeReturnedEquals: " +
condition.getHttpErrorCodeReturnedEquals() +
                           "\n" + "KeyPrefixEquals: " + condition.getKeyPrefixEquals());
               }
```

```
            }
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.2.11 DELETE Bucket Website

此操作用来删除存储桶的网站托管属性。只有根用户和拥有 DELETE Bucket WebSite 权限的子用户才能执行此操作。

示例代码

```
private static void deleteBucketWebsite(AmazonS3 oosClient) {
       try {
           oosClient.deleteBucketWebsiteConfiguration(BUCKET_NAME);
           System.out.println("delete bucket website success!");
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.12 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。

示例代码

```
private static void listMultipartUploads(AmazonS3 oosClient) {
       try {
           ListMultipartUploadsRequest listMultipartUploadsRequest = new
ListMultipartUploadsRequest(BUCKET_NAME);
           // Delimiter, a character used to group keywords, only supports "/"
           listMultipartUploadsRequest.setDelimiter("/");
           // Set the maximum number of returned multipart upload processes, value: [1,
1000]
           listMultipartUploadsRequest.setMaxUploads(100);
           // Used to restrict the results returned, which must start with this prefix
           listMultipartUploadsRequest.setPrefix("");
           MultipartUploadListing listing =
oosClient.listMultipartUploads(listMultipartUploadsRequest);
           if (listing != null) {
               int sz = listing.getMultipartUploads().size();
               System.out.printf("max uploads:%d,sz=%d%n", listing.getMaxUploads(), sz);
               for (int i = 0; i < sz; i++) {
                   MultipartUpload upload = listing.getMultipartUploads().get(i);
                   System.out.printf("upload key:%s, id:%s,
owner:%s,displayName:%s,ownerId:%s%n", upload.getKey(),
                           upload.getUploadId(), upload.getOwner().toString(),
upload.getOwner().getDisplayName(), upload.getOwner().getId());
               }
               // When a delimiter is defined, the return results will include
CommonPrefixes
               for (String commonPrefix : listing.getCommonPrefixes()) {
                   System.out.println("commonPrefix: " + commonPrefix);
               }
           }
       } catch (AmazonServiceException se) {
```

```
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.2.13 PUT Bucket Logging

此操作用来添加/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。只有根用户和拥有 PUT Bucket Logging 权限的子用户才能执行此操作。

示例代码

```
private static void putBucketLogging(AmazonS3 oosClient) {
       try {
           BucketLoggingConfiguration loggingConfiguration = new
BucketLoggingConfiguration();
           // Set the prefix for generating log files
           loggingConfiguration.setLogFilePrefix("/log");
           // Set the target bucket
           loggingConfiguration.setDestinationBucketName(BUCKET_NAME);
           SetBucketLoggingConfigurationRequest setBucketLoggingConfigurationRequest =
                   new SetBucketLoggingConfigurationRequest(BUCKET_NAME,
loggingConfiguration);

oosClient.setBucketLoggingConfiguration(setBucketLoggingConfigurationRequest);
           System.out.println("put bucket logging success!");
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.14 GET Bucket Logging

此操作用来获得指定 Bucket 的 logging。只有根用户和拥有 GET Bucket Logging 权限的子用户才能执行此操作。

示例代码

```
private static void getBucketLogging(AmazonS3 oosClient) {
       try {
           BucketLoggingConfiguration logging =
oosClient.getBucketLoggingConfiguration(BUCKET_NAME);
           System.out.println("GetBucketLogging: " +
                   "\n" + logging.toString());
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.15 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。

示例代码

```
private static void headBucket(AmazonS3 oosClient) {
       try {
           oosClient.headBucket(BUCKET_NAME);
           System.out.println("bucket exist.");
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.16 PUT Bucket Lifecycle

此操作用来设置 Bucket 生命周期规则。只有根用户和具有 PUT Bucket Lifecycle 权限的子用户才能执行此操作。

示例代码

```
private static void putBucketLifecycle(AmazonS3 oosClient) {
       try {
           BucketLifecycleConfiguration bucketLifecycleConfiguration = new
BucketLifecycleConfiguration();
           List<Rule> rules = new ArrayList<>();
           // Configure a lifecycle rule
           Rule rule1 = new Rule();
           // Set the unique identifier for the rule
           rule1.setId("rule1");
           // Set the object prefix for using the lifecycle
           rule1.setPrefix("test");
           // Set whether the lifecycle is effective
           // Enabled | Disabled
           rule1.setStatus("Enabled");
           // Specify after how many days the lifecycle rule becomes effective
           // following the object's last modification or last access time
//           rule1.setExpirationInDays(60);
           // Set the dumping rules for the lifecycle
           BucketLifecycleConfiguration.Transition transition = new
BucketLifecycleConfiguration.Transition();
           // Change standard storage objects to infrequent access storage objects
           transition.setStorageClass(StorageClass.Standard_IA);
           // Specify the effective date of the lifecycle rule
           Calendar calendar = Calendar.getInstance(TimeZone.getTimeZone("GMT"),
Locale.ENGLISH);
           calendar.add(Calendar.DAY_OF_MONTH, 8);
           calendar.set(Calendar.HOUR_OF_DAY, 0);
           calendar.set(Calendar.MINUTE, 0);
           calendar.set(Calendar.SECOND, 0);
           calendar.set(Calendar.MILLISECOND, 0);
```

```
            transition.setDate(calendar.getTime());
            rule1.setTransition(transition);
            rules.add(rule1);
            bucketLifecycleConfiguration.setRules(rules);
            oosClient.setBucketLifecycleConfiguration(BUCKET_NAME,
bucketLifecycleConfiguration);
            System.out.println("put bucket lifecycle success!");
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.2.17 GET Bucket Lifecycle

此操作用于返回配置的 Bucket 生命周期。

示例代码

```
private static void getBucketLifecycle(AmazonS3 oosClient) {
       try {
           BucketLifecycleConfiguration configuration =
oosClient.getBucketLifecycleConfiguration(BUCKET_NAME);
           if (configuration.getRules() != null) {
               System.out.println("GetBucketLifecycle: ");
               for (Rule rule : configuration.getRules()) {
                   System.out.printf("rule id=%s,prefix=%s,status=%s,exp=%d days%n",
                           rule.getId(), rule.getPrefix(), rule.getStatus(),
rule.getExpirationInDays());
                   System.out.println("transition: " +
                           "\nstorageClass: " + rule.getTransition().getStorageClass() +
                           "\ndays: " + rule.getTransition().getDays());
               }
           }
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.18 DELETE Buckete Lifecycle

此操作用于删除配置的 Bucket 生命周期，OOS 将会删除指定 Bucket 的所有生命周期配置规则。用户的文件将永远不会到期，OOS 也不会再自动删除文件。只有根用户和拥有 DELETE Bucket Lifecycle 权限的子用户才能执行此操作。

示例代码

```
private static void deleteBucketLifecycle(AmazonS3 oosClient) {
       try {
           oosClient.deleteBucketLifecycleConfiguration(BUCKET_NAME);
           System.out.println("delete bucket lifecycle success!");
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.19 PUT Bucket CORS

此操作用来设置 Bucket 的跨域资源共享(Cross-Origin Resource Sharing，CORS)。浏览器限制脚本内发起跨源 HTTP 请求，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。通过配置 CORS，可以解决不同域相互访问的问题，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。

示例代码

```
private static void putBucketCORS(AmazonS3 oosClient) {
       try {
           BucketCrossOriginConfiguration bucketCrossOriginConfiguration = new
BucketCrossOriginConfiguration();
           // The sources and methods allowed for user cross-origin requests
           CORSRule rule = new CORSRule();
           // Set the allowed HTTP methods for cross-origin requests
           List<CORSRule.AllowedMethods> allowedMethods = new ArrayList<>();
           allowedMethods.add(CORSRule.AllowedMethods.GET);
           allowedMethods.add(CORSRule.AllowedMethods.POST);
           // Set the allowed origins for cross-origin requests
           List<String> allowedOrigins = new ArrayList<>();
           allowedOrigins.add("https://www.ctyun.cn");
           // Specify the cache duration for the results of preflight requests in the
browser, in seconds
           int maxAgeSeconds = 10;
           // Set the response headers that client applications can access
           List<String> exposedHeaders = new ArrayList<>();
           // Control whether the header specified in the Access-Control-RequestHeaders
header in the precheck OPTIONS request is allowed
           List<String> allowedHeaders = new ArrayList<>();
           exposedHeaders.add("x-test-111");
           exposedHeaders.add("x-test-222");
           allowedHeaders.add("x-oos-111");
           allowedHeaders.add("x-oos-222");
           // Set the unique identifier for the rule
```

```
            rule.setId("myFirstRuleId");
            rule.setAllowedMethods(allowedMethods);
            rule.setAllowedOrigins(allowedOrigins);
            rule.setExposedHeaders(exposedHeaders);
            rule.setAllowedHeaders(allowedHeaders);
            rule.setMaxAgeSeconds(maxAgeSeconds);
            List<CORSRule> rules = new ArrayList<>();
            rules.add(rule);
            bucketCrossOriginConfiguration.setRules(rules);
            oosClient.setBucketCrossOriginConfiguration(BUCKET_NAME,
bucketCrossOriginConfiguration);
            System.out.println("put bucket CORS success!");
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.2.20 GET Bucket CORS

此操作用来返回 Bucket 的跨域配置信息。只有根用户和拥有 GET Bucket CORS 权限的子用户才能执行此操作。

示例代码

```
private static void getBucketCORS(AmazonS3 oosClient) {
       try {
           BucketCrossOriginConfiguration cors =
oosClient.getBucketCrossOriginConfiguration(BUCKET_NAME);
           if (cors != null) {
               List<CORSRule> rules = cors.getRules();
               if (rules != null) {
                   System.out.println("GetBucketCORS:");
                   for (CORSRule rule : rules) {
                       System.out.println(
                               "id: " + rule.getId() + "\n" +
                                       "MaxAgeSeconds: " + rule.getMaxAgeSeconds()
                       );
                       for (String allowedHeader : rule.getAllowedHeaders()) {
                           System.out.println("AllowHeader: " + allowedHeader);
                       }
                       for (CORSRule.AllowedMethods allowedMethod :
rule.getAllowedMethods()) {
                           System.out.println("AllowMethod: " +
allowedMethod.toString());
                       }
                       for (String allowedOrigin : rule.getAllowedOrigins()) {
                           System.out.println("AllowOrigin: " + allowedOrigin);
                       }
                       for (String exposedHeader : rule.getExposedHeaders()) {
                           System.out.println("ExposeHeader: " + exposedHeader);
                       }
                   }
               }
           }
```

```
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.2.21 DELETE Bucket CORS

此操作用来删除 Bucket 的跨域配置信息。只有根用户和拥有 DELETE Bucket CORS 权限的子用户才能执行此操作。

示例代码

```
private static void deleteBucketCORS(AmazonS3 oosClient) {
       try {
           oosClient.deleteBucketCrossOriginConfiguration(BUCKET_NAME);
           System.out.println("delete bucket CORS success!");
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.22 PUT Bucket Object Lock

使用此操作可以开启合规保留功能，开启后将对 Bucket 中所有文件生效。只有根用户和有权限的子用户才可以进行此操作，匿名用户不能进行此操作。

开启 Bucket 合规保留功能后，任何用户（包括根用户）都不能对此 Bucket 内处于合规保留期的文件进行修改和删除。

可以重复调用此接口：

- 如果已经开启合规保留策略：设置合规保留时长大于或等于上次设置的时长，才能生效。如果使用 Years 和 Days 两种方式设置合规保留时长，年与天的换算关系为：1 年等于 365 天。
- 如果未开启合规保留策略：设置合规保留时长可以大于、等于或小于上次设置的时长。

注意：

- 合规保留一旦开启，不能关闭，不能缩短合规保留时长，但可以延长合规保留时长。
- 合规保留的时间精确到秒，例如对 Bucket A 设置合规保留时长为 10 天，文件 A 属于 Bucket A，A1 的最后更新时间为 2019-3-1 12:00:00，该文件会在 2019-3-11 12:00:01 过合规保留期。
- 任何用户（包括根用户）都不能修改、覆盖、删除处于合规保留期的文件。
- 处于合规保留期的文件，无法通过调用 API、控制台修改文件的存储类型，只能通过生命周期修改存储类型。
- 处于合规保留期的文件，如果设置了生命周期规则，则修改存储类型的生命周期规则可以生效，设置删除操作的生命周期规则待文件过了合规保留期后才能生效。

示例代码

```
private static void putBucketObjectLock(AmazonS3 oosClient) {
       try {
           ObjectLockConfiguration lockConf = new ObjectLockConfiguration();
           // Enable compliance retention feature
           lockConf.setLockEnable();
           // Set the number of days for compliance retention, during which the object
cannot be deleted
           lockConf.setDays(1);
```

```
            oosClient.setObjectLockConfiguration(BUCKET_NAME, lockConf);
            System.out.println("put bucket object lock success!");
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.2.23 GET Bucket Object Lock

使用此操作可以获取 Bucket 合规保留的配置信息。只有根用户和有权限的子用户才可以进行此操作。

示例代码

```
private static void getBucketObjectLock(AmazonS3 oosClient) {
       try {
           ObjectLockConfiguration lockConf =
oosClient.getObjectLockConfiguration(BUCKET_NAME);
           System.out.println("GetBucketObjectLock: " +
                   "\n" + lockConf.toString());
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.24 DELETE Bucket Object Lock

使用此操作可以删除未启用的合规保留配置信息。只有根用户和有权限的子用户才可以进行此操作。

示例代码

```
private static void deleteBucketObjectLock(AmazonS3 oosClient) {
       try {
           oosClient.deleteObjectLockConfiguration(BUCKET_NAME);
           System.out.println("delete bucket object lock success!");
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.2.25 PUT Bucket Inventory Configuration

此操作用来为 Bucket 配置 Bucket 清单规则。

通过 OOS Bucket 清单功能可以获取 Bucket 中指定文件（Object）的大小、存储类型等信息。相对于 GET Bucket (List Objects)接口，Bucket 清单结果文件可以按天或者按周以 CSV 的形式输出指定文件的相关信息，且不会影响 Bucket 的请求速率。在需要列举海量 Object 的场景中，推荐使用 Bucket 清单功能。

说明：每个 Bucket 最多可以配置 10 条 Bucket 清单规则。

示例代码

```
private static void putBucketInventoryConfiguration(AmazonS3 oosClient) {
       try {
           SetBucketInventoryConfigurationRequest request = new
SetBucketInventoryConfigurationRequest();
           BucketInventoryConfiguration configuration = new
BucketInventoryConfiguration();
           List<BucketInventoryDestination> destinations = new ArrayList<>();
           BucketInventoryDestination destination = new BucketInventoryDestination();
           List<OOSBucketDestination> bucketDestinations = new ArrayList<>();
           OOSBucketDestination bucketDestination = new OOSBucketDestination();
           List<BucketInventoryFilter> filters = new ArrayList<>();
           BucketInventoryFilter filter = new BucketInventoryFilter();
           List<BucketInventoryOptionalFields> optionalFields = new ArrayList<>();
           BucketInventoryOptionalFields optionalField = new
BucketInventoryOptionalFields();
           List<BucketInventorySchedule> schedules = new ArrayList<>();
           BucketInventorySchedule schedule = new BucketInventorySchedule();

           // Configure the request parameter ID
           // ID must be set!
           // The ID can only be a combination of lowercase letters, numbers, hyphens
(-), and underscores (_),
           // with a length of 1 to 64 characters
           String id = "id1";
           request.setId(id);
```

```
            // The logging bucket name must be set
            request.setBucketName(BUCKET_NAME);

            // The container for storing inventory results must be configured
            // The target bucket must be set and can be different from the logging bucket
            // The prefix must be 'arn:ctyun:oos:::' or 'arn:aws:s3:::'
            bucketDestination.setBucket("arn:ctyun:oos:::" + "destinationBucketName");
            // The format must be set and must be 'CSV'
            bucketDestination.setFormat("CSV");
            // The directory prefix for saving inventory files must be between 0 and 512
characters in length
            bucketDestination.setPrefix("");
            bucketDestinations.add(bucketDestination);
            destination.setOOSBucketDestinations(bucketDestinations);
            destinations.add(destination);
            configuration.setDestinations(destinations);

            // Configure whether the specified inventory feature is enabled; this must be
set
            configuration.setEnabled(true);

            // Optional
            // Configure the prefix for inventory filtering, with a length of 0 to 1024
characters
            filter.setPrefix("");
            filters.add(filter);
            configuration.setFilters(filters);

            // Configure the specified inventory name,
            // which must be set and must exactly match the ID specified in the request
parameters
            configuration.setId(id);

            // Optional
            // Configure the container for inventory result configuration items
            // For specific configurable content, please refer to the developer
documentation
```

```
            optionalField.setField("Size");
            optionalFields.add(optionalField);
            configuration.setOptionalFields(optionalFields);

            // Configure the export frequency of the inventory result files
            // this must be set and can only be configured as 'Daily' or 'Weekly'
            schedule.setFrequency("Daily");
            schedules.add(schedule);
            configuration.setSchedules(schedules);

            request.setBucketInventoryConfiguration(configuration);
            oosClient.setBucketInventoryConfiguration(request);
            System.out.println("put bucket inventory configuration success!");
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.2.26 GET Bucket Inventory Configuration

此操作用来查看某 Bucket 中指定清单配置。

示例代码

```
private static void getBucketInventoryConfiguration(AmazonS3 oosClient) {
       try {
           GetBucketInventoryConfigurationRequest request = new
GetBucketInventoryConfigurationRequest();
           // ID must be set!
   // Only lowercase letters, numbers, hyphens (-), and underscores (_) are allowed
           // It cannot start or end with a hyphen (-) or an underscore (_)
           // Length must be between 1 and 64 characters.
           request.setId("id1");
           // bucketName must be set
           request.setBucketName(BUCKET_NAME);
           GetBucketInventoryConfigurationResult result =
oosClient.getBucketInventoryConfiguration(request);
           BucketInventoryConfiguration inventoryConfiguration =
result.getInventoryConfiguration();
           if (inventoryConfiguration != null) {
               System.out.println("GetBucketInventoryConfiguration: " +
                       "\n" + "id: " + inventoryConfiguration.getId() +
                       "\n" + "isEnabled: " + inventoryConfiguration.isEnabled()
               );
               if (inventoryConfiguration.getDestinations() != null) {
                   System.out.println("BucketInventoryDestination/OOSBucketDestination");
                   for (BucketInventoryDestination destination :
inventoryConfiguration.getDestinations()) {
                       if (destination.getOOSBucketDestinations() != null) {
                           for (OOSBucketDestination bucketDestination :
destination.getOOSBucketDestinations()) {
                               System.out.println("bucket: " +
bucketDestination.getBucket() +
                                       "\n" + "format: " + bucketDestination.getFormat() +
```

```
                                    "\n" + "OOSBucketDestinationPrefix: " +
bucketDestination.getPrefix()
                                );
                            }
                        }
                    }
                }
                if (inventoryConfiguration.getFilters() != null) {
                    System.out.println("Filter ");
                    for (BucketInventoryFilter filter :
inventoryConfiguration.getFilters()) {
                        System.out.println("FilterPrefix: " + filter.getPrefix());
                    }
                }
                if (inventoryConfiguration.getOptionalFields() != null) {
                    System.out.println("OptionalFields ");
                    for (BucketInventoryOptionalFields optionalFields :
inventoryConfiguration.getOptionalFields()) {
                        System.out.println("Field: " + optionalFields.getField());
                    }
                }
                if (inventoryConfiguration.getSchedules() != null) {
                    System.out.println("Schedule ");
                    for (BucketInventorySchedule schedule :
inventoryConfiguration.getSchedules()) {
                        System.out.println("Frequency: " + schedule.getFrequency());
                    }
                }
            } else {
                System.out.println("BucketInventoryConfiguration not set.");
            }
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.2.27 List Bucket Inventory Configuration

此操作用来查看某 Bucket 的所有 Bucket 清单的配置。

示例代码

```
private static void listBucketInventoryConfiguration(AmazonS3 oosClient) {
       try {
           ListBucketInventoryConfigurationRequest request = new
ListBucketInventoryConfigurationRequest();
           // bucketName must be set
           request.setBucketName(BUCKET_NAME);
           // Optional
           // Note that if the previous list request did not return all inventory
configurations,
           // the continuationToken here must be set to the value of
NextContinuationToken returned by the last list request
           request.setContinuationToken("");
           ListBucketInventoryConfigurationResult result =
oosClient.listBucketInventoryConfiguration(request);
           if (result != null) {
               System.out.println(
                       "ListBucketInventoryConfiguration: " +
                               "\n" + "isTruncated: " + result.isTruncated() +
                               "\n" + "continuationToken: " +
result.getContinuationToken() +
                               "\n" + "nextContinuationToken: " +
result.getNextContinuationToken()
               );
               if (result.getBucketInventoryConfigurations() != null) {
                   for (BucketInventoryConfiguration configuration :
result.getBucketInventoryConfigurations()) {
                       System.out.println("id: " + configuration.getId() +
                               "\n" + "isEnabled: " + configuration.isEnabled()
                       );
                       if (configuration.getDestinations() != null) {
```

```
                        for (BucketInventoryDestination destination :
configuration.getDestinations()) {
                            System.out.println("BucketInventoryDestination ");
                            if (destination.getOOSBucketDestinations() != null) {
                                System.out.println("OOSBucketDestination ");
                                for (OOSBucketDestination bucketDestination :
destination.getOOSBucketDestinations()) {
                                    System.out.println("bucket: " +
bucketDestination.getBucket() +
                                            "\n" + "format: " +
bucketDestination.getFormat() +
                                            "\n" + "OOSBucketDestinationPrefix: " +
bucketDestination.getPrefix()
                                    );
                                }
                            }
                        }
                        if (configuration.getFilters() != null) {
                            System.out.println("Filter ");
                            for (BucketInventoryFilter filter :
configuration.getFilters()) {
                                System.out.println("FilterPrefix: " + filter.getPrefix());
                            }
                        }
                        if (configuration.getOptionalFields() != null) {
                            System.out.println("OptionalFields ");
                            for (BucketInventoryOptionalFields optionalFields :
configuration.getOptionalFields()) {
                                System.out.println("Field: " + optionalFields.getField());
                            }
                        }
                        if (configuration.getSchedules() != null) {
                            System.out.println("Schedule ");
                            for (BucketInventorySchedule schedule :
configuration.getSchedules()) {
```

```
                            System.out.println("Frequency: " +
schedule.getFrequency());
                        }
                    }
                }
            }
        } else {
            System.out.println("BucketInventoryConfiguration not set.");
        }
    } catch (AmazonServiceException se) {
        System.out.println(se.toString());
    } catch (AmazonClientException ce) {
        System.out.println(ce.getMessage());
    }
}
```

4.2.28 DELETE Bucket Inventory Configuration

此操作用来删除某 Bucket 的指定 Bucket 清单配置。

示例代码

```
private static void deleteBucketInventoryConfiguration(AmazonS3 oosClient) {
       try {
           DeleteBucketInventoryConfigurationRequest request = new
DeleteBucketInventoryConfigurationRequest();
           // bucketName must be set
           request.setBucketName(BUCKET_NAME);
           // ID must be set!
           // The ID can only be a combination of lowercase letters, numbers, hyphens
(-), and underscores (_),
           // with a length of 1 to 64 characters
           request.setId("id1");
           oosClient.deleteBucketInventoryConfiguration(request);
           System.out.println("delete bucket inventory configuration success!");
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.3 关于 Object 的操作

4.3.1 PUT Object

此操作用来向指定 Bucket 中添加一个文件，要求发送请求者对该 Bucket 有写权限，用户必须添加完整的文件。

说明：文件名称不能包含 ASCII 码为 0 的字符（NUL）。

示例代码

```
private static void putObject(AmazonS3 oosClient) {
       try {
           // Uploading an object requires the configuration of three parameters:
           // BucketName, ObjectName, and the file to upload.
           // All other parameters can be left unset and commented out
           PutObjectRequest request = new PutObjectRequest(BUCKET_NAME, OBJECT_NAME,
UPLOAD_FILE);
           // Optional
           // Set the metadata for the object
           ObjectMetadata objectMetadata = new ObjectMetadata();
           // Optional
           // Set the MD5, it can be calculated automatically
//         objectMetadata.setContentMD5("");
           // Optional
           // Set the caching behavior
//         objectMetadata.setCacheControl("");
           // Optional
           // Set the expiration time of the object in the browser
//         objectMetadata.setExpirationTime(new Date(new Date().getTime() + 1000 * 3600 * 2));
           // Set the storage location. refer to the developer documentation for details.
           CtyunBucketDataLocation location = new CtyunBucketDataLocation();
           ArrayList<String> dataRegions = new ArrayList<>();
           dataRegions.add("QingDao");
           location.setDataRegions(dataRegions);
           objectMetadata.setDataLocation(location);

           request.setMetadata(objectMetadata);
```

```
            // Optional
            // Forbid overwrite, default value is false
            // Note: This configuration is only supported by some resource pools; please refer to
the user manual
//            request.setForbidOverwrite(false);

            // Optional
            // Set the upload rate limit for the object, in KiB/s
            // Note: If the value is between 0 and 128, the rate is set at 128 KiB/s
//        request.setLimit("256");
            PutObjectResult result = oosClient.putObject(request);
            System.out.println("put object success!"  +
                    "\n" + OBJECT_NAME + ",md5: " + result.getContentMd5());
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.3.2 GET Object

此操作用来检索在 OOS 中的文件信息，执行此操作，用户必须对 Object 所在的 Bucket 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

```
private static void getObject(AmazonS3 oosClient) {
       try {
           // Set the bucketName and objectName
           GetObjectRequest request = new GetObjectRequest(BUCKET_NAME, OBJECT_NAME);
           // Set the start position for retrieving the object
           long start = 0;
           // Set the end position for retrieving the object
           long end = 0;
           // Optional
           // Set the range for retrieving the object
//           request.setRange(start, end);
           // Optional
           // Set the download rate for the object
           // Note that the 'limit' and 'limitrate' parameters are mutually exclusive
and only one can be chosen
//           request.setLimitrate("");
           // Set the download rate for the object
           // Note that the 'limit' and 'limitrate' parameters can only be chosen one at
a time
//           request.setLimit("");
           // Optional
           // Return objects modified after this timestamp
//           request.setModifiedSinceConstraint(new Date());
           // Optional
           // Return objects that have not been modified after this timestamp
//           request.setUnmodifiedSinceConstraint(new Date());

//           ArrayList<String> eTagsMatch = new ArrayList<>();
//           eTagsMatch.add("eTag1");
           // Optional
```

```
            // Return the object when the target object's ETag matches the specified
value
//            request.setMatchingETagConstraints(eTagsMatch);

//            ArrayList<String> eTagsNonMatch = new ArrayList<>();
//            eTagsNonMatch.add("eTag2");
            // Optional
            // Do not return the object when the target object's ETag matches the
specified value
//            request.setNonmatchingETagConstraints(eTagsNonMatch);

            S3Object s3Object = oosClient.getObject(request);
            if (s3Object != null) {
                S3ObjectInputStream input = s3Object.getObjectContent();
                FileOutputStream fos = null;
                try {
                    int bufferSize = 2 * 1024 * 1024;
                    fos = new FileOutputStream(DOWNLOAD_FILE);
                    byte[] buffer = new byte[bufferSize];
                    int bytesRead = -1;
                    while ((bytesRead = input.read(buffer)) > -1) {
                        fos.write(buffer, 0, bytesRead);
                    }
                    fos.flush();
                } catch (IOException e) {
                    try {
                        input.abort();
                    } catch (IOException abortException) {
                        abortException.printStackTrace();
                    }
                } finally {
                    if (fos != null) {
                        try {
                            fos.close();
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
```

```
                }
                if (input != null) {
                    try {
                        input.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
            System.out.println("get object: '" + OBJECT_NAME + "' success!");
        } else {
            System.out.println("There's no object you want to get here.");
        }
    } catch (AmazonServiceException se) {
        System.out.println(se.toString());
    } catch (AmazonClientException ce) {
        System.out.println(ce.getMessage());
    }
}
```

4.3.3 DELETE Object

此操作用来删除指定的文件，要求用户要对文件所在的 Bucket 拥有写权限。

示例代码

```
private static void deleteObject(AmazonS3 oosClient) {
       try {
           oosClient.deleteObject(BUCKET_NAME, OBJECT_NAME);
           System.out.println("delete object: '" + OBJECT_NAME + "' success!");
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.3.4 PUT Object - Copy

此操作用来创建一个存储在 OOS 里的文件拷贝。类似于执行一个 GET，然后再执行一次 PUT。要执行拷贝请求，用户需要对源文件有读权限，对目标 Bucket 有写权限。

示例代码

```
private static void copyObject(AmazonS3 oosClient) {
       try {
           String copyKey = "testCopy";
           // Source BucketName, Source ObjectName, Target BucketName, and Target
ObjectName must be set
           CopyObjectRequest request = new CopyObjectRequest(BUCKET_NAME, OBJECT_NAME,
BUCKET_NAME, copyKey);
           // Optional
           // Set the storage type for the copied object,the default is standard storage
//           request.setStorageClass(StorageClass.Standard);
//           ObjectMetadata objectMetadata = new ObjectMetadata();
           // Optional
           // Set new metadata attributes for the copied object, such as setting the
storage location
//           CtyunBucketDataLocation dataLocation = new CtyunBucketDataLocation();
//           ArrayList<String> location = new ArrayList<>();
//           location.add("QingDao");
//           dataLocation.setDataRegions(location);
           // Optional
           // Set the scheduling policy: 'Allow' to permit scheduling or 'NotAllowed' to
deny scheduling
//
dataLocation.setStragegy(CtyunBucketDataLocation.CtyunBucketDataScheduleStrategy.Allowed
);
//           objectMetadata.setDataLocation(dataLocation);
           // Optional
           // If NewObjectMetaData is set, then x-amz-metadata-directive is of the
'replace' type, otherwise, it is 'copy'
//           request.setNewObjectMetadata(objectMetadata);
           // Optional
```

```
            // Set to only perform the object copy operation when the source object's
Etag matches the given Etag
//            ArrayList<String> eTagMatch = new ArrayList<>();
//            eTagMatch.add("eTag1");
//            request.setMatchingETagConstraints(eTagMatch);

            // Optional
            // Set to only perform the object copy operation when the source object's
Etag does not match the given Etag
//            ArrayList<String> eTagNonMatch = new ArrayList<>();
//            eTagNonMatch.add("eTag2");
//            request.setNonmatchingETagConstraints(eTagNonMatch);

            // Optional
            // Set to only perform the object copy operation if the source object has not
been modified since the specified time
//            request.setUnmodifiedSinceConstraint(new Date());

            // Optional
            // Set to only perform the object copy operation if the source object has
been modified after the specified time
//            request.setModifiedSinceConstraint(new Date());

            CopyObjectResult result = oosClient.copyObject(request);
            if (result != null) {
                System.out.printf("copyObject success!" + System.lineSeparator() +
"lastmodify:%s,etag:%s%n", result.getLastModifiedDate(), result.getETag());
            } else {
                System.out.println("copyObject fail.please check your configuration.");
            }
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.3.5 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文件。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

```
private static String initialMultipartUpload(AmazonS3 oosClient) {
       try {
           // Set the BucketName and ObjectName
           InitiateMultipartUploadRequest request = new
InitiateMultipartUploadRequest(BUCKET_NAME, OBJECT_NAME);
           // Optional
           // Set the storage type, the default is standard storage
           request.setStorageClass(StorageClass.Standard);
           // Optional
           // Set metadata information, such as the data's storage location
           ObjectMetadata objectMetadata = new ObjectMetadata();
           CtyunBucketDataLocation dataLocation = new CtyunBucketDataLocation();
           ArrayList<String> location = new ArrayList<>();
           // Set the data archiving location to Qingdao
           location.add("QingDao");
           dataLocation.setDataRegions(location);
           // Optional
           // Allowed for permitting scheduling | NotAllowed for denying scheduling

dataLocation.setStragegy(CtyunBucketDataLocation.CtyunBucketDataScheduleStrategy.Allowed);
           objectMetadata.setDataLocation(dataLocation);
           request.setObjectMetadata(objectMetadata);
           InitiateMultipartUploadResult res = oosClient.initiateMultipartUpload(request);
           String uploadID = res.getUploadId();
           System.out.println("initialMultipartUpload success!" +
                  "\n" + "uploadID: " + uploadID);
           return uploadID;
       } catch (AmazonServiceException se) {
```

```
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
        return null;
    }
```

4.3.6 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文件中的位置，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。

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。

```
private static void uploadPart(AmazonS3 oosClient, String uploadId, int
currentPartNumber, long currentPartSize,
                                List<PartETag> parts, long partNum, long offset) {
       try {
           boolean isLastPart = currentPartNumber == partNum;
           UploadPartRequest request = new UploadPartRequest().withUploadId(uploadId);
           // Set the file to be uploaded in parts
           // Note that there is no need to manually split the file, simply set the file
to be uploaded in parts
           request.setFile(UPLOAD_FILE);
           // Set the BucketName
           request.setBucketName(BUCKET_NAME);
           // Set the ObjectName
           request.setKey(OBJECT_NAME);
           // Indicate the current part number being uploaded
           request.setPartNumber(currentPartNumber);
           // Set the size of the upload part
           request.setPartSize(currentPartSize);
           // If it is not the first part, set the offset of the part
           if (currentPartNumber != 1) {
```

```
                request.setFileOffset(offset);
            }
            // Set whether it is the last part
            request.setLastPart(isLastPart);
            UploadPartResult res = oosClient.uploadPart(request);
            System.out.println("uploadPart " + currentPartNumber + " success!");
            String eTag = res.getPartETag().getETag();
            System.out.println("eTag: " + eTag);
            PartETag e = new PartETag(currentPartNumber, eTag);
            parts.add(e);
            System.out.println("The " + currentPartNumber + " part-> size: " +
request.getPartSize() + " offset: " + request.getFileOffset());
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.3.7 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文件。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。

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。

示例代码

```
private static void completeMultipartUpload(AmazonS3 oosClient, String uploadId,
List<PartETag> parts) {
       try {
           // Set the BucketName, ObjectName, the uploadId for initializing multipart
upload,
           // and the eTag values returned for the parts that have been successfully
uploaded
           CompleteMultipartUploadRequest request = new
CompleteMultipartUploadRequest(BUCKET_NAME, OBJECT_NAME, uploadId, parts);
           CompleteMultipartUploadResult result =
oosClient.completeMultipartUpload(request);
           System.out.println("complete multipart upload success!" +
                  "\n" + "location: " + result.getLocation());
           System.out.printf("bucket:%s,obj:%s,etag:%s%n", result.getBucketName(),
result.getKey(), result.getETag());
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
```

```
            System.out.println(ce.getMessage());
        }
    }
```

一次完整的上传分片请求

示例代码

```
private static void completeMultipartUploadWithUploadPart(AmazonS3 oosClient) {
        try {
            // When merging parts, each part must be at least 5 MiB in size
            long partSize = 5 * 1024 * 1024; // 5MiB
            String uploadId = initialMultipartUpload(oosClient);
            // Calculate the total number of parts needed for the upload
            long partNum = (UPLOAD_FILE.length() % partSize) == 0 ?
UPLOAD_FILE.length() / partSize : (UPLOAD_FILE.length() / partSize) + 1;
            // Collect the eTag values returned from the successfully uploaded parts
            List<PartETag> parts = new ArrayList<>();
            for (int i = 1; i <= partNum; i++) {
                // Calculate the offset for each part, in Byte
                long offset = (i - 1) * partSize;
                // Calculate the size of each part, in Byte
                long currentPartSize = (i == partNum ? UPLOAD_FILE.length() - (partSize
* (i - 1)) : partSize);
                uploadPart(oosClient, uploadId, i, currentPartSize, parts, partNum,
offset);
            }
            completeMultipartUpload(oosClient, uploadId, parts);

        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.3.8 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

```
private static void abortMultipartUpload(AmazonS3 oosClient, String uploadId) {
       try {
           // Set the BucketName, ObjectName, and the UploadId obtained during the
initialization of the multipart upload
           AbortMultipartUploadRequest request = new AbortMultipartUploadRequest(
                   BUCKET_NAME, OBJECT_NAME, uploadId);
           oosClient.abortMultipartUpload(request);
           System.out.println("abort multipart upload success!,uploadId: " + uploadId);
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.3.9 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

```
private static void listParts(AmazonS3 oosClient, String uploadId) {
       try {
           ListPartsRequest request = new ListPartsRequest(
                   BUCKET_NAME, OBJECT_NAME, uploadId);
           // Optional
           // Set the maximum number of parts to be returned in the response body,
ranging from 1 to 1000, with a default of 1000
           request.setMaxParts(1000);
           PartListing result = oosClient.listParts(request);
           System.out.println("listPart: " +
                   "\n" + "bucketName: " + result.getBucketName() +
                   "\n" + "objectKey: " + result.getKey() +
                   "\n" + "uploadId: " + result.getUploadId() +
                   "\n" + "maxParts: " + result.getMaxParts()
           );
           if (result.getParts() != null) {
               for (PartSummary part : result.getParts()) {
                   System.out.println("partNumber: " + part.getPartNumber() +
                           "\n" + "eTag: " + part.getETag() +
                           "\n" + "size: " + part.getSize() +
                           "\n" + "lastModified: " + part.getLastModified()
                   );
               }
           }
```

```
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.3.10 Copy Part

此接口用来将已经存在的 Object 作为分段上传的片段，拷贝生成一个新的片段。

示例代码

```
private static void copyPart(AmazonS3 oosClient, String uploadId) {
       try {
           CopyPartRequest request = new CopyPartRequest();
           // Optional
           // When copying a part of the source object, you need to set the start and
stop positions, in Byte
           // If not set, the entire object will be copied
           long firstByte = 0;
           long lastByte = 100;
//           request.setFirstByte(firstByte);
//           request.setLastByte(lastByte);
           // Set the BucketName where the source object is located
           request.setSourceBucketName(BUCKET_NAME);
           // Set the source object key
           request.setSourceKey("sourceKey");
           // Set the target BucketName
           request.setDestinationBucketName(BUCKET_NAME);
           // Set the target object key
           request.setDestinationKey(OBJECT_NAME);
           // Set the copied part object as which part of the multipart upload
           request.setPartNumber(1);
           // Set the uploadId returned during the initialization of the multipart
upload
           request.setUploadId(uploadId);
           // Optional
           // Set to only perform the copy operation if the object has been modified
after the specified time
//           request.setModifiedSinceConstraint(new Date());
           // Optional
           // Set to only perform the copy operation if the object has not been modified
after the specified time
```

```
//            request.setUnmodifiedSinceConstraint(new Date());
//            ArrayList<String> etagMatch = new ArrayList<>();
//            etagMatch.add("etag1");
            // Optional
            // Set to only perform the copy operation when the object's actual etag
matches the given etag
//            request.setMatchingETagConstraints(etagMatch);
//            ArrayList<String> etagNonMatch = new ArrayList<>();
//            etagNonMatch.add("etag2");
            // Optional
            // Set to only perform the copy operation when the object's actual etag does
not match the given etag
//            request.setNonmatchingETagConstraints(etagNonMatch);

            CopyPartResult result = oosClient.copyPart(request);
            System.out.println("CopyPart success! " +
                    "\n" + "etag: " + result.getETag() +
                    "\n" + "partNumber: " + result.getPartNumber() +
                    "\n" + "versionId: " + result.getVersionId());
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.3.11 Delete Multiple Objects

批量删除 Object 功能支持用一个 HTTP 请求删除一个 Bucket 中的多个 Object。如果你知道你想删除的 Object 名字，此功能可以批量删除这些 Object，而不用发送多个单独的删除请求。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。对于每个 Object，OOS 都会返回删除的结果，成功或者失败。

示例代码

```
private static void deleteObjects(AmazonS3 oosClient, List<String> objectNames) {
       try {
           List<DeleteObjectsRequest.KeyVersion> keys = new ArrayList<>();
           DeleteObjectsRequest request = new DeleteObjectsRequest(BUCKET_NAME);
           for (String name : objectNames) {
               DeleteObjectsRequest.KeyVersion key = new
DeleteObjectsRequest.KeyVersion(name, null);
               keys.add(key);
           }
           request.setKeys(keys);
           DeleteObjectsResult result = oosClient.deleteObjects(request);
           System.out.println("DeleteObjects: ");
           if (result.getDeletedObjects() != null) {
               for (DeleteObjectsResult.DeletedObject deletedObject :
result.getDeletedObjects()) {
                   System.out.println("key: " + deletedObject.getKey() + " delete
success!");
               }
           }
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.3.12 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

```
private static void multiUpload(AmazonS3 oosClient) {
       try {
           String filePath = "";
           // Set multiple parameters in UploadFileRequest
           UploadFileRequest request = new UploadFileRequest(BUCKET_NAME, OBJECT_NAME);
           // The local file to be uploaded
           request.setUploadFile(filePath);
           // Concurrent threads for multipart upload, default is 1, with a maximum of
1000
           request.setTaskNum(5);
           // The size of each part, with a default of 5 MiB.
           // If partSize is less than 5 MiB, it will be adjusted to 5 MiB for all parts
except the last one
           request.setPartSize(10 * 1024 * 1024);
           // Enable the resumable upload feature; it is disabled by default
           request.setEnableCheckpoint(true);
           // A file to record the results of local part uploads
           // This parameter needs to be set when enabling the resumable upload feature
           // Progress information during the upload process will be saved in this file
           // If a part upload fails, the upload will resume from the recorded point in
the file upon retry
           // After the upload is complete, this file will be deleted
           // By default, it is in the same directory as the local file to be uploaded
           MultipleUpload multiUpload = new MultipleUpload(request, oosClient);
           // Resumable upload
           CompleteMultipartUploadResult result = multiUpload.upload();
```

```
            System.out.println("MultiUpload success!" +
                    "\n" + "objectName: " + result.getKey() +
                    "\n" + "bucketName: " + result.getBucketName() +
                    "\n" + "location: " + result.getLocation() +
                    "\n" + "etag: " + result.getETag());
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.3.13 POST Object

此接口用来使用 HTML 表单将文件上传到指定的 Bucket。POST 是另一种形式的 PUT 操作，POST 可以让使用者通过 Browser-based 的方式，将文件上传到指定 Bucket 中。

需要依赖 sdk 包中的 Post Object。

示例代码

```
private static void postObject() {
        PostObject po = new PostObject();
        // Set the contentType of the request header
        ContentType contentType = ContentType.create("image/jpeg",
StandardCharsets.UTF_8);
        try {
            po.postObjectDemo(BUCKET_NAME, OBJECT_NAME, contentType);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

4.3.14 OPTIONS Object

浏览器可以向 OOS 发送预检请求，来判断其是否可以发送特定源、HTTP 方法和头的实际请求。当浏览器发送预检请求时，OOS 根据 Bucket 的跨域配置来返回响应信息。

示例代码

```
private static void optionsObject(AmazonS3 oosClient) {
        try {
            OptionObjectRequest request = new OptionObjectRequest();
            // Set the objectName
            request.setObjectName(OBJECT_NAME);
            // Set the bucketName
            request.setBucketName(BUCKET_NAME);
            // Set the origin
            request.setOrigin("https://www.ctyun.cn");
            // Set the HTTP method to be used in the actual request
            request.setAccessControlRequestMethod("PUT");
            // Optional
            // Set the HTTP request headers to be sent in the actual request, separated
by commas(,)
            request.setAccessControlRequestHeaders("GET, DELETE");
            oosClient.optionsObject(request);
            System.out.println("options object success!");
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.3.15 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

```
private static void generatePresignedUrl(AmazonS3 oosClient) {
       try {
           GeneratePresignedUrlRequest shareUrlRequest = new
GeneratePresignedUrlRequest(
                   BUCKET_NAME, OBJECT_NAME);
           Date now = new Date();
           // Set the expiration time to one day later
           Date expire = new Date(now.getTime() + 1000 * 3600 * 24);
           // Optional
           // Set the connection timeout, with a default of 15 minutes
           shareUrlRequest.setExpiration(expire);
           // Optional
           // Set the download speed limit in KiB/s
           //shareUrlRequest.addRequestParameter("x-amz-limitrate", "2048");
           URL url = oosClient.generatePresignedUrl(shareUrlRequest);
           System.out.println("generatePresignedUrl success!" +
                   "\n" + url.toString());
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

4.3.16 分段上传高级别 API

分段上传高级别 API 用于简化分段上传。用户可以从文件或流上传数据，也可以设置高级选项，例如，用于分段上传的片段大小，或在并发上传时使用的线程数。用户可以使用 PutObjectRequest 和 TransferManagerConfiguration 类来设置这些高级选项。Java API 的 TransferManager 类提供了高级别 API，用户可以使用它来上传数据。TransferManager 会尝试使用多个线程来一次性上传单个文件的多个分段片段。当文件大小很大，而且带宽也很大时，此操作可以大幅度地增加吞吐量。

高级别 API 文件上传过程如下：

1. 创建 TransferManager 类的实例。
2. 用户可以选择从文件还是从流上传数据，并执行 TransferManager.upload 方法。

以下 Java 示例代码演示了上述任务。

```
private static void testTransferManagerUpload(AmazonS3 oosClient) {
        ThreadFactory threadFactory = new ThreadFactory() {
            private int threadCount = 1;

            public Thread newThread(Runnable r) {
                Thread thread = new Thread(r);
                thread.setName("s3-transfer-manager-worker-" + threadCount++);
                return thread;
            }
        };
        ThreadPoolExecutor threadPoolExecutor = (ThreadPoolExecutor)
Executors.newFixedThreadPool(10, threadFactory);
        TransferManager transferManager = new TransferManager(oosClient,
threadPoolExecutor);
        System.out.println("before upload");
        Upload upload = transferManager.upload(BUCKET_NAME, OBJECT_NAME, UPLOAD_FILE);
        System.out.println("in upload");
        try {
            upload.waitForUploadResult();
            System.out.println("Upload complete.");
        } catch (AmazonServiceException | InterruptedException se) {
            System.out.println(se.toString());
```

```
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

TransferManager 类提供了 abortMultipartUploads 方法，用于终止正在上传的分段片段。用户可以在调用 API 时指定日期，删除所有在指定日期之前被初始化，且还在上传的分段片段。

高级别 API 终止分段片段过程如下：

1. 创建 TransferManager 类的实例。
2. 执行 TransferManager. abortMultipartUploads 方法，指定 Bucket 名称和日期。

以下Java示例代码演示了上述任务。

```
private static void abortTransferManagerUpload(AmazonS3 oosClient) {
        TransferManager transferManager = new TransferManager(oosClient);
        Date oneWeekAgo = new Date(System.currentTimeMillis() - 1000 * 60 * 60 * 24 * 7);
        try {
            transferManager.abortMultipartUploads(BUCKET_NAME, oneWeekAgo);
        } catch (AmazonServiceException se) {
            System.out.println(se.toString());
        } catch (AmazonClientException ce) {
            System.out.println(ce.getMessage());
        }
    }
```

4.3.17 HEAD Object

此接口用于获取文件的元数据信息，而不返回数据本身。当只希望获取文件的属性信息时，可以调用此接口。

示例代码

```
private static void headObject(AmazonS3 oosClient) {
       try {
           ObjectMetadata metadata = oosClient.getObjectMetadata(BUCKET_NAME, OBJECT_NAME);
           if (metadata.getUserMetadata() != null && metadata.getUserMetadata().size() > 0)
{
               System.out.println("UserMetaData");
               for (String userMetaDataKey : metadata.getUserMetadata().keySet()) {
                   System.out.println(userMetaDataKey + ": " +
metadata.getUserMetadata().get(userMetaDataKey));
               }
           }
           if (metadata.getRawMetadata() != null && metadata.getRawMetadata().size() > 0) {
               System.out.println("RawMetaData");
               for (String rawMetaData : metadata.getRawMetadata().keySet()) {
                   System.out.println(rawMetaData + ": " +
metadata.getRawMetadata().get(rawMetaData));
               }
           }
           System.out.println("========================");
           System.out.println("etag: " + metadata.getETag() +
                   "\n" + "contentMD5: " + metadata.getContentMD5() +
                   "\n" + "expirationTime: " + metadata.getExpirationTime());
       } catch (AmazonServiceException se) {
           System.out.println(se.toString());
       } catch (AmazonClientException ce) {
           System.out.println(ce.getMessage());
       }
   }
```

5 统计分析服务代码示例

统计分析（Statistics API）服务代码示例在 example 的子目录：

\src\cn\ctyun\test\ManagementTest.java。

创建统计分析 Client 对象示例：

```
private static AmazonS3 getAmazonS3() {
        ClientConfiguration configuration = new ClientConfiguration();
        configuration.setSocketTimeout(30 * 1000);
        configuration.setConnectionTimeout(30 * 1000);
        configuration.setProtocol(Protocol.HTTP);
        // 使用 V4 签名
        System.setProperty(SDKGlobalConfiguration.ENABLE_S3_SIGV4_SYSTEM_PROPERTY, "true");
        // 负载是否参与签名，设置参与
        S3ClientOptions options = new S3ClientOptions();
        options.setPayloadSigningEnabled(true);
        AmazonS3 client = new AmazonS3Client(new
PropertiesCredentials(TestConfig.OOS_ACCESS_KEY, TestConfig.OOS_SECRET_KEY), configuration);
        client.setEndpoint(TestConfig.OOS_MANAGEMENT_ENDPOINT);
        client.setS3ClientOptions(options);
        return client;
    }
```

5.1 GetCapacity

此操作用来查询用户的容量。

示例代码

```
private static void getCapacity(AmazonS3 client) {
        CtyunGetCapacityRequest request = new CtyunGetCapacityRequest();
        Date date = new Date();
        // Specify the start time for querying capacity
        request.setBeginDate(new Date(date.getTime() - (1000 * 3600 * 24)));
        // Specify the end time for querying capacity
        request.setEndDate(date);
        request.setBucket(BUCKET_NAME);
        // Specify the storage type to query: STANDARD/STANDARD_IA/all, with the default
being STANDARD
        request.setStorageClass(StorageClass.All);
        // Specify the data location for querying capacity
        // If no data location name is specified, the total capacity of all data
locations under the account name will be queried
        request.setRegion("QingDao");
        // Set the sampling granularity of the statistics data: byHour | by5min | byDay
        request.setFreq("byHour");

        try {
            CtyunGetCapacityResult result = client.ctyunGetCapacity(request);
            for (CtyunCapacityStatistics statistics : result.getCapacityStatistics()) {
                if (statistics.getStatisticsData() != null) {
                    System.out.println(
                            "Date: " + statistics.getStatisticsDate() + "\n" +
                                    "MaxCapacity: " +
statistics.getStatisticsData().getMaxCapacity() + "\n" +
                                    "AverageCapacity: " +
statistics.getStatisticsData().getAverageCapacity() + "\n" +
                                    "SampleCapacity: " +
statistics.getStatisticsData().getSampleCapacity() + "\n"
                    );
```

```
            }
            if (statistics.getStatisticsData_ia() != null) {
                System.out.println(
                        "Date: " + statistics.getStatisticsDate() + "\n" +
                                "MaxCapacity_ia: " +
statistics.getStatisticsData_ia().getMaxCapacity() + "\n" +
                                "AverageCapacity_ia: " +
statistics.getStatisticsData_ia().getAverageCapacity() + "\n" +
                                "SampleCapacity_ia: " +
statistics.getStatisticsData_ia().getSampleCapacity() + "\n"
                );
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

5.2 GetBilledStorageUsage

此操作用来查询出账相关的峰值容量信息。用户可以根据需求，查询指定存储桶、指定数据位置、指定存储类型的容量。

示例代码

```
private static void getBilledStorageUsage(AmazonS3 client) {
        CtyunGetBilledStorageUsageRequest request = new
CtyunGetBilledStorageUsageRequest();
        Date date = new Date();
        request.setBeginDate(new Date(date.getTime() - 1000 * 3600 * 24));
        request.setEndDate(date);
        // Specify the storage type to query: STANDARD/STANDARD_IA/all, with the default
being all
        request.setStorageClass(StorageClass.All);
        request.setBucket(BUCKET_NAME);
        // Specify the data location for querying capacity
        // If no data location name is specified, the total capacity of all data
locations under the account name will be queried
        request.setRegion("QingDao");
        // Set the sampling granularity of the statistics data: byHour | by5min, with the
default being byHour
        request.setFreq(Freq.BY_HOUR);
        try {
            CtyunGetBilledStorageUsageResult billedStorageUsage =
client.ctyunGetBilledStorageUsage(request);
            for (CtyunGetBilledStorageUsageStatistics statics :
billedStorageUsage.getCtyunGetBilledStorageUsageStatistics()) {
                if (statics.getStandard() != null) {
                    System.out.println(
                            "Date: " + statics.getStatisticsDate() + "\n" +
                                    "BilledStorageUsage: " +
statics.getStandard().getBilledStorageUsage() + "\n" + // 单位 Byte
                                    "RemainderChargeStorageUsage: " +
statics.getStandard().getRemainderChargeStorageUsage() + "\n" + // 值为 0
```

```
                                    "RemainderChargeOfDuration: " +
statics.getStandard().getRemainderChargeOfDuration() + "\n" + // 值为 0
                                    "RemainderChargeOfSize: " +
statics.getStandard().getRemainderChargeOfSize() + "\n"  // 值为 0
                    );
                }
                if (statics.getStandard_ia() != null) {
                    System.out.println(
                            "Date: " + statics.getStatisticsDate() + "\n" +
                                    "BilledStorageUsage_ia: " +
statics.getStandard_ia().getBilledStorageUsage() + "\n" + // 单位 Byte
                                    "RemainderChargeStorageUsage_ia: " +
statics.getStandard_ia().getRemainderChargeStorageUsage() + "\n" + // 单位 Byte
                                    "RemainderChargeOfDuration_ia: " +
statics.getStandard_ia().getRemainderChargeOfDuration() + "\n" + // 单位 Byte
                                    "RemainderChargeOfSize: " +
statics.getStandard_ia().getRemainderChargeOfSize() + "\n" // 单位 Byte


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

5.3 GetRestoreCapacity

此操作用来查询数据取回量。用户可以根据需求，查询指定存储桶、指定存储类型的数据取回量。

示例代码

```
private static void getRestoreCapacity(AmazonS3 client) {
        CtyunGetRestoreCapacityRequest request = new CtyunGetRestoreCapacityRequest();
        Date date = new Date();
        request.setBeginDate(date);
        request.setEndDate(date);
        request.setBucket(BUCKET_NAME);
        // Set the sampling granularity of the statistics data: byHour | by5min | byDay,
with the default being byHour
        request.setFreq(Freq.BY_HOUR);
        // Specify the storage type to query: STANDARD_IA/all, with the default being all
        request.setStorageClass(StorageClass.All);
        // Specify the data location for querying capacity
        // If no data location name is specified, the total capacity of all data
locations under the account name will be queried
        request.setRegion("QingDao");
        try {
            CtyunGetRestoreCapacityResult result =
client.ctyunGetRestoreCapacity(request);
            for (CtyunGetRestoreCapacityStatistics statistics :
result.getCtyunGetRestoreCapacityStatistics()) {
                if (statistics.getStandard_iaCapacity() != null) {
                    System.out.println(
                            "Date: " + statistics.getStatisticsDate() + "\n" +
                                    "RestoreCapacity_ia: " +
statistics.getStandard_iaCapacity().getCapacity() + "\n"
                    );
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }


    }
```

5.4 GetDeleteCapacity

此操作用来查询用户删除的容量。

示例代码

```
private static void getDeleteCapacity(AmazonS3 client) {
        CtyunDeleteCapacityRequest request = new CtyunDeleteCapacityRequest();
        Date date = new Date();
        request.setBeginDate(new Date(date.getTime() - 1000 * 3600 * 24));
        request.setEndDate(date);
        // Set the sampling granularity of the statistical data: byHour | by5min | byDay,
with the default being byHour
        request.setFreq("byHour");
        // Specify the data location for querying capacity
        // If no data location name is specified, the total capacity of all data
locations under the account will be queried
        request.setRegion("QingDao");
        // Specify the storage type to query: STANDARD/STANDARD_IA/all, with the default
being STANDARD
        request.setStorageClass(StorageClass.All);
        request.setBucket(BUCKET_NAME);
        try {
            CtyunGetDeleteCapacityResult result = client.ctyunGetDeleteCapacity(request);
            System.out.println("User: " + result.getBucketName());
            for (CtyunDeleteCapacityStatistics statistics :
result.getDeleteCapacityStatistics()) {
                if (statistics.getStatisticsData() != null) {
                    System.out.println(
                            "Date: " + statistics.getStatisticsDate() + "\n" +
                                    "DeleteCapacity: " +
statistics.getStatisticsData().getDeleteCapacity() + "\n"
                    );
                }
                if (statistics.getStatisticsIA() != null) {
                    System.out.println(
                            "Date: " + statistics.getStatisticsDate() + "\n" +
```

```
                                "DeleteCapacity_ia: " +
statistics.getStatisticsIA().getDeleteCapacity() + "\n"
                    );
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

5.5 GetTraffics

此操作用来查询用户的流量。

示例代码

```
private static void getTraffics(AmazonS3 client) {
        CtyunGetTrafficsRequest request = new CtyunGetTrafficsRequest();
        Date date = new Date();
        request.setBeginDate(new Date(date.getTime() - 1000 * 3600 * 24));
        request.setEndDate(date);
        request.setBucket(BUCKET_NAME);
        // Specify the storage type to query: STANDARD/STANDARD_IA/all, with the default
being STANDARD
        request.setStorageClass(StorageClass.All);
        // Set the sampling granularity of the statistical data: byHour | by5min | byDay,
with the default being byHour
        request.setFreq("byHour");
        // Specify the data location for querying capacity
        // If no data location name is specified, the total capacity of all data
locations under the account will be queried
        request.setRegion("QingDao");
        // Set the type of traffic to be returned: inbound/outbound/all, with the default
being all
        request.setInOutType("all");
        // Set the type of traffic to be returned: internet/noninternet/all, with the
default being all
        request.setInternetType("all");
        // Set the type of traffic to be returned: direct/roam/all, with the default
being all
        request.setTrafficsType("all");
        try {
            CtyunGetTrafficsResult result = client.ctyunGetTraffics(request);
            for (CtyunGetTrafficsStatistics statistics : result.getTrafficsStatistics())
{
                if (statistics.getStatisticsData() != null) {
                    CtyunGetTrafficsStatisticsData data = statistics.getStatisticsData();
```

```
                    System.out.println(
                            "Date: " + statistics.getStatisticsDate() + "\n" +
                                    "InternetDirectInbound: " +
data.getInternetDirectInbound() + "\t" +
                                    "InternetRoamInbound: " + data.getInternetRoamInbound()
+ "\t" +
                                    "NonInternetDirectInbound: " +
data.getNonInternetDirectInbound() + "\t" +
                                    "NonInternetRoamInbound: " +
data.getNonInternetRoamInbound() + "\t" +
                                    "InternetDirectOutbound: " +
data.getInternetDirectOutbound() + "\t" +
                                    "InternetRoamOutbound: " +
data.getInternetRoamOutbound() + "\t" +
                                    "NonInternetDirectOutbound: " +
data.getNonInternetDirectOutbound() + "\t" +
                                    "NonInternetRoamOutbound: " +
data.getNonInternetRoamOutbound() + "\n"
                    );
                }
                if (statistics.getStatisticsData_ia() != null) {
                    CtyunGetTrafficsStatisticsData data_ia =
statistics.getStatisticsData_ia();
                    System.out.println(
                            "Date " + statistics.getStatisticsDate() + "\n" +
                                    "InternetDirectInbound_ia: " +
data_ia.getInternetDirectInbound() + "\t" +
                                    "InternetRoamInbound_ia: " +
data_ia.getInternetRoamInbound() + "\t" +
                                    "NonInternetDirectInbound_ia: " +
data_ia.getNonInternetDirectInbound() + "\t" +
                                    "NonInternetRoamInbound_ia: " +
data_ia.getNonInternetRoamInbound() + "\t" +
                                    "InternetDirectOutbound_ia: " +
data_ia.getInternetDirectOutbound() + "\t" +
                                    "InternetRoamOutbound_ia: " +
data_ia.getInternetRoamOutbound() + "\t" +
```

```
                                    "NonInternetDirectOutbound_ia: " +
data_ia.getNonInternetDirectOutbound() + "\t" +
                                    "NonInternetRoamOutbound_ia: " +
data_ia.getNonInternetRoamOutbound() + "\n"
                    );
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

5.6 GetRequests

此操作用来查询用户的请求次数

示例代码

```
private static void getRequests(AmazonS3 client) {
        CtyunGetRequestsRequest request = new CtyunGetRequestsRequest();
        Date date = new Date();
        request.setBeginDate(new Date(date.getTime() - 1000 * 3600 * 24));
        request.setEndDate(date);
        request.setBucket(BUCKET_NAME);
        // Specify the data location for querying capacity
        // If no data location name is specified, the total capacity of all data
locations under the account will be queried
        request.setRegion("QingDao");
        // Set the type of traffic to be returned: internet/noninternet/all, with the
default being all
        request.setInternetType("all");
        // Set the type of requests to be returned: put/get/delete/head/post/others/all,
with the default being all
        request.setRequestsType("all");
        // Set the sampling granularity of the statistical data: byHour | by5min | byDay,
with the default being byHour
        request.setFreq("byHour");
        // Specify the storage type to query: STANDARD/STANDARD_IA/all, with the default
being STANDARD
        request.setStorageClass(StorageClass.All);
        try {
            CtyunGetRequestsResult result = client.ctyunGetRequests(request);
            for (CtyunGetRequestsStatistics statistics : result.getRequestsStatistics())
{
                if (statistics.getStatisticsData() != null) {
                    CtyunGetRequestsStatisticsData data = statistics.getStatisticsData();
                    System.out.println(
                            "Date: " + statistics.getStatisticsDate() + "\n" +
                                    "Internet.GetRequest: " + data.getGetRequest() + "\n" +
```

```
                                    "Internet.HeadRequest: " + data.getHeadRequest() + "\n"
+
                                    "Internet.PutRequest: " + data.getPutRequest() + "\n" +
                                    "Internet.PostRequest: " + data.getPostRequest() + "\n"
+
                                    "Internet.DeleteRequest: " + data.getDeleteRequest() +
"\n" +
                                    "Internet.OthersRequest: " + data.getOtherRequest() +
"\n" +
                                    "NonInternet.GetRequest: " +
data.getNonInternetGetRequest() + "\n" +
                                    "NonInternet.HeadRequest: " +
data.getNonInternetHeadRequest() + "\n" +
                                    "NonInternet.PutRequest: " +
data.getNonInternetPutRequest() + "\n" +
                                    "NonInternet.PostRequest: " +
data.getNonInternetPostRequest() + "\n" +
                                    "NonInternet.DeleteRequest: " +
data.getNonInternetDeleteRequest() + "\n" +
                                    "NonInternet.OthersRequest: " +
data.getNonInternetOtherRequest() + "\n"
                    );
                }
                if (statistics.getStatisticsData_ia() != null) {
                    CtyunGetRequestsStatisticsData data_ia =
statistics.getStatisticsData_ia();
                    System.out.println(
                            "Date: " + statistics.getStatisticsDate() + "\n" +
                                    "Internet.GetRequest_ia: " + data_ia.getGetRequest() +
"\n" +
                                    "Internet.HeadRequest_ia: " + data_ia.getHeadRequest()
+ "\n" +
                                    "Internet.PutRequest_ia: " + data_ia.getPutRequest() +
"\n" +
                                    "Internet.PostRequest_ia: " + data_ia.getPostRequest()
+ "\n" +
```

```
                                    "Internet.DeleteRequest_ia: " +
data_ia.getDeleteRequest() + "\n" +
                                    "Internet.OthersRequest_ia: " +
data_ia.getOtherRequest() + "\n" +
                                    "NonInternet.GetRequest_ia: " +
data_ia.getNonInternetGetRequest() + "\n" +
                                    "NonInternet.HeadRequest_ia: " +
data_ia.getNonInternetHeadRequest() + "\n" +
                                    "NonInternet.PutRequest_ia: " +
data_ia.getNonInternetPutRequest() + "\n" +
                                    "NonInternet.PostRequest_ia: " +
data_ia.getNonInternetPostRequest() + "\n" +
                                    "NonInternet.DeleteRequest_ia: " +
data_ia.getNonInternetDeleteRequest() + "\n" +
                                    "NonInternet.OthersRequest_ia: " +
data_ia.getNonInternetOtherRequest() + "\n"
                    );
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

5.7 GetReturnCode

此操作用来查询用户的请求返回码次数。

示例代码

```
private static void getReturnCode(AmazonS3 client) {
        CtyunGetReturnCodeRequest request = new CtyunGetReturnCodeRequest();
        Date date = new Date();
        request.setBeginDate(new Date(date.getTime() - 1000 * 3600 * 24));
        request.setEndDate(date);
        request.setBucket(BUCKET_NAME);
        // Specify the data location for querying capacity
        // If no data location name is specified, the total capacity of all data
locations under the account will be queried
        request.setRegion("QingDao");
        // Set the type of traffic to be returned: internet/noninternet/all, with the
default being all
        request.setInternetType("all");
        List<String> requestTypeList = new ArrayList<>();
        requestTypeList.add("all");
        // Set the type of requests to be returned: put/get/delete/head/post/others/all,
with the default being all
        request.setRequestsTypeList(requestTypeList);

        List<String> responseTypeList = new ArrayList<>();
        responseTypeList.add("all");
        // Specify the request type for returning the count of return codes:
        //
Response200/Response204/Response206/Response403/Response404/Response4xx/Response500/Resp
onse503/all,
        // with the default being all
        request.setResponseTypeList(responseTypeList);
        // Set the sampling granularity of the statistical data: byHour | by5min | byDay,
with the default being byHour
        request.setFreq("byHour");
```

```
        // Specify the storage type to query: STANDARD/STANDARD_IA/all, with the default
being STANDARD
        request.setStorageClass(StorageClass.All);
        try {
            CtyunGetReturnCodeResult result = client.ctyunGetReturnCode(request);
            for (CtyunGetReturnCodeStatistics statistics :
result.getReturnCodeStatistics()) {
                if (statistics.getStatisticsData() != null) {
                    CtyunGetReturnCodeStatisticsData data =
statistics.getStatisticsData();
                    System.out.println(
                            "GetReturnCode:"
                                    + "\n Date:" + statistics.getStatisticsDate()

                                    + "\n Internet Get 200 :" + data.getGet200()
                                    + "\n Internet Get 206 :" + data.getGet206()
                                    + "\n Internet Get 403 :" + data.getGet403()
                                    + "\n Internet Get 404 :" + data.getGet404()
                                    + "\n Internet Get 4xx :" + data.getGet4xx()

                                    + "\n Internet Head 200 :" + data.getHead200()
                                    + "\n Internet Head 206 :" + data.getHead206()
                                    + "\n Internet Head 403 :" + data.getHead403()
                                    + "\n Internet Head 404 :" + data.getHead404()
                                    + "\n Internet Head 4xx :" + data.getHead4xx()

                                    + "\n Internet Put 200 :" + data.getPut200()
                                    + "\n Internet Put 403 :" + data.getPut403()
                                    + "\n Internet Put 404 :" + data.getPut404()
                                    + "\n Internet Put 4xx :" + data.getPut4xx()

                                    + "\n Internet Post 200 :" + data.getPost200()
                                    + "\n Internet Post 403 :" + data.getPost403()
                                    + "\n Internet Post 404 :" + data.getPost404()
                                    + "\n Internet Post 4xx :" + data.getPost4xx()

                                    + "\n Internet Delete 200 :" + data.getDelete200()
```

```
                                + "\n Internet Delete 204 :" + data.getDelete204()
                                + "\n Internet Delete 403 :" + data.getDelete403()
                                + "\n Internet Delete 404 :" + data.getDelete404()
                                + "\n Internet Delete 4xx :" + data.getDelete4xx()

                                + "\n Internet Others 200 :" + data.getOthers200()
                                + "\n Internet Others 403 :" + data.getOthers403()
                                + "\n Internet Others 404 :" + data.getOthers404()
                                + "\n Internet Others 4xx :" + data.getOthers4xx()



                                + "\n NonInternet Get 200 :" +
data.getGet200NonInternet()
                                + "\n NonInternet Get 206 :" +
data.getGet206NonInternet()
                                + "\n NonInternet Get 403 :" +
data.getGet403NonInternet()
                                + "\n NonInternet Get 404 :" +
data.getGet404NonInternet()
                                + "\n NonInternet Get 4xx :" +
data.getGet4xxNonInternet()

                                + "\n NonInternet Head 200 :" +
data.getHead200NonInternet()
                                + "\n NonInternet Head 206 :" +
data.getHead206NonInternet()
                                + "\n NonInternet Head 403 :" +
data.getHead403NonInternet()
                                + "\n NonInternet Head 404 :" +
data.getHead404NonInternet()
                                + "\n NonInternet Head 4xx :" +
data.getHead4xxNonInternet()

                                + "\n NonInternet Put 200 :" +
data.getPut200NonInternet()
                                + "\n NonInternet Put 403 :" +
data.getPut403NonInternet()
```

```
                                + "\n NonInternet Put 404 :" +
data.getPut404NonInternet()
                                + "\n NonInternet Put 4xx :" +
data.getPut4xxNonInternet()

                                + "\n NonInternet Post 200 :" +
data.getPost200NonInternet()
                                + "\n NonInternet Post 403 :" +
data.getPost403NonInternet()
                                + "\n NonInternet Post 404 :" +
data.getPost404NonInternet()
                                + "\n NonInternet Post 4xx :" +
data.getPost4xxNonInternet()

                                + "\n NonInternet Delete 200 :" +
data.getDelete200NonInternet()
                                + "\n NonInternet Delete 204 :" +
data.getDelete204NonInternet()
                                + "\n NonInternet Delete 403 :" +
data.getDelete403NonInternet()
                                + "\n NonInternet Delete 404 :" +
data.getDelete404NonInternet()
                                + "\n NonInternet Delete 4xx :" +
data.getDelete4xxNonInternet()

                                + "\n NonInternet Others 200 :" + data.getOthers200()
                                + "\n NonInternet Others 403 :" + data.getOthers403()
                                + "\n NonInternet Others 404 :" + data.getOthers404()
                                + "\n NonInternet Others 4xx :" + data.getOthers4xx()
                    );
                }
                if (statistics.getStatisticsData_ia() != null) {
                    CtyunGetReturnCodeStatisticsData data_ia =
statistics.getStatisticsData_ia();
                    System.out.println(
                            "GetReturnCode_ia:"
                                    + "\n Date:" + statistics.getStatisticsDate()
```

```
                                + "\n Internet Get_ia 200 :" + data_ia.getGet200()
                                + "\n Internet Get_ia 206 :" + data_ia.getGet206()
                                + "\n Internet Get_ia 403 :" + data_ia.getGet403()
                                + "\n Internet Get_ia 404 :" + data_ia.getGet404()
                                + "\n Internet Get_ia 4xx :" + data_ia.getGet4xx()

                                + "\n Internet Head_ia 200 :" + data_ia.getHead200()
                                + "\n Internet Head_ia 206 :" + data_ia.getHead206()
                                + "\n Internet Head_ia 403 :" + data_ia.getHead403()
                                + "\n Internet Head_ia 404 :" + data_ia.getHead404()
                                + "\n Internet Head_ia 4xx :" + data_ia.getHead4xx()

                                + "\n Internet Put_ia 200 :" + data_ia.getPut200()
                                + "\n Internet Put_ia 403 :" + data_ia.getPut403()
                                + "\n Internet Put_ia 404 :" + data_ia.getPut404()
                                + "\n Internet Put_ia 4xx :" + data_ia.getPut4xx()

                                + "\n Internet Post_ia 200 :" + data_ia.getPost200()
                                + "\n Internet Post_ia 403 :" + data_ia.getPost403()
                                + "\n Internet Post_ia 404 :" + data_ia.getPost404()
                                + "\n Internet Post_ia 4xx :" + data_ia.getPost4xx()

                                + "\n Internet Delete_ia 200 :" +
data_ia.getDelete200()
                                + "\n Internet Delete_ia 204 :" +
data_ia.getDelete204()
                                + "\n Internet Delete_ia 403 :" +
data_ia.getDelete403()
                                + "\n Internet Delete_ia 404 :" +
data_ia.getDelete404()
                                + "\n Internet Delete_ia 4xx :" +
data_ia.getDelete4xx()

                                + "\n Internet Others_ia 200 :" +
data_ia.getOthers200()
```

```
                                    + "\n Internet Others_ia 403 :" +
data_ia.getOthers403()
                                    + "\n Internet Others_ia 404 :" +
data_ia.getOthers404()
                                    + "\n Internet Others_ia 4xx :" +
data_ia.getOthers4xx()



                                    + "\n NonInternet Get_ia 200 :" +
data_ia.getGet200NonInternet()
                                    + "\n NonInternet Get_ia 206 :" +
data_ia.getGet206NonInternet()
                                    + "\n NonInternet Get_ia 403 :" +
data_ia.getGet403NonInternet()
                                    + "\n NonInternet Get_ia 404 :" +
data_ia.getGet404NonInternet()
                                    + "\n NonInternet Get_ia 4xx :" +
data_ia.getGet4xxNonInternet()

                                    + "\n NonInternet Head_ia 200 :" +
data_ia.getHead200NonInternet()
                                    + "\n NonInternet Head_ia 206 :" +
data_ia.getHead206NonInternet()
                                    + "\n NonInternet Head_ia 403 :" +
data_ia.getHead403NonInternet()
                                    + "\n NonInternet Head_ia 404 :" +
data_ia.getHead404NonInternet()
                                    + "\n NonInternet Head_ia 4xx :" +
data_ia.getHead4xxNonInternet()

                                    + "\n NonInternet Put_ia 200 :" +
data_ia.getPut200NonInternet()
                                    + "\n NonInternet Put_ia 403 :" +
data_ia.getPut403NonInternet()
                                    + "\n NonInternet Put_ia 404 :" +
data_ia.getPut404NonInternet()
```

```
                                    + "\n NonInternet Put_ia 4xx :" +
data_ia.getPut4xxNonInternet()

                                    + "\n NonInternet Post_ia 200 :" +
data_ia.getPost200NonInternet()
                                    + "\n NonInternet Post_ia 403 :" +
data_ia.getPost403NonInternet()
                                    + "\n NonInternet Post_ia 404 :" +
data_ia.getPost404NonInternet()
                                    + "\n NonInternet Post_ia 4xx :" +
data_ia.getPost4xxNonInternet()

                                    + "\n NonInternet Delete_ia 200 :" +
data_ia.getDelete200NonInternet()
                                    + "\n NonInternet Delete_ia 204 :" +
data_ia.getDelete204NonInternet()
                                    + "\n NonInternet Delete_ia 403 :" +
data_ia.getDelete403NonInternet()
                                    + "\n NonInternet Delete_ia 404 :" +
data_ia.getDelete404NonInternet()
                                    + "\n NonInternet Delete_ia 4xx :" +
data_ia.getDelete4xxNonInternet()

                                    + "\n NonInternet Others_ia 200 :" +
data_ia.getOthers200()
                                    + "\n NonInternet Others_ia 403 :" +
data_ia.getOthers403()
                                    + "\n NonInternet Others_ia 404 :" +
data_ia.getOthers404()
                                    + "\n NonInternet Others_ia 4xx :" +
data_ia.getOthers4xx()
                    );
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }
```

5.8 GetConcurrentConnection

此操作用来查询用户的并发连接数。

示例代码

```
private static void getConcurrentConnection(AmazonS3 client) {
        CtyunGetConcurrentConnectionRequest request = new
CtyunGetConcurrentConnectionRequest();
        Date date = new Date();
        // The time span cannot exceed one day, and the sampling interval is 5 minutes
        request.setBeginDate(new Date(date.getTime() - 1000 * 3600 * 12));
        request.setEndDate(date);
        request.setBucket(BUCKET_NAME);
        // Specify the data location for querying capacity
        // If no data location name is specified, the total capacity of all data
locations under the account will be queried
        request.setRegion("QingDao");
        // Set the type of traffic to be returned: internet/noninternet/all, with the
default being all
        request.setInternetType("all");
        // Set the sampling granularity of the statistics data; it can only be by5min
        request.setFreq("by5min");
        try {
            CtyunGetConcurrentConnectionResult result =
client.ctyunGetConcurrentConnection(request);
            for (CtyunGetConcurrentConnectionStatistics statistics :
result.getConcurrentConnectionStatistics()) {
                if (statistics.getStatisticsData() != null) {
                    CtyunGetConcurrentConnectionStatisticsData data =
statistics.getStatisticsData();
                    System.out.println(
                            "Date: " + statistics.getStatisticsDate() + "\n" +
                                    "InternetConnection: " + data.getInternetConnection() +
"\t" +
                                    "NonInternetConnection: " +
data.getNonInternetConnection() + "\n"
```

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

5.9 GetUsage

此操作用来查询用户的 Bucket 的使用情况。

说明：不建议使用此操作，建议根据要查询的项，使用下列接口：GetCapacity、GetDeleteCapcity、GetRequests、GetReturnCode、GetConcurrentConnection。

示例代码

```
private static void getUsage(AmazonS3 client) {
        CtyunGetUsageRequest request = new CtyunGetUsageRequest();
        Date date = new Date();
        // Set the start time to three hours ago
        request.setBeginDate(new Date(date.getTime() - 1000 * 3600 * 3));
        // Set the end time to the current time
        request.setEndDate(date);
        // With a statistical unit of 5 minutes
        request.setMinutesFrequenceEnabled(true);
        request.setBucketName(BUCKET_NAME);
        try {
            CtyunGetUsageResult result = client.ctyunGetUsage(request);
            System.out.println(
                    "UserName: " + result.getUserName() + "\n" +
                    "BucketName: " + result.getBucketName()
            );
            for (CtyunPoolUsage poolUsage : result.getDatailUsage()) {
                if (poolUsage.getCtyunPoolUsageData() != null) {
                    System.out.println("PoolName: " + poolUsage.getPoolName());
                    for (CtyunPoolUsageData poolUsageData :
poolUsage.getCtyunPoolUsageData()) {
                        System.out.println(
                                "Date: " + poolUsageData.getDate() + "\n" +
                                "Capacity: " + poolUsageData.getCapacity() + "\n" +
                                "Upload: " + poolUsageData.getUpload() + "\n" +
                                "Download: " + poolUsageData.getDownload() + "\n" +
                                "DeleteFlow: " + poolUsageData.getDeleteFlow() + "\n" +
                                "RoamUpload: " + poolUsageData.getRoamUpload() + "\n" +
```

```
                                "RoamDownload: " + poolUsageData.getRoamDownload() + "\n" +
                                "GetRequest: " + poolUsageData.getGetRequest() + "\n" +
                                "HeadRequest: " + poolUsageData.getHeadRequest() + "\n" +
                                "PutRequest: " + poolUsageData.getPutRequest() + "\n" +
                                "PostRequest: " + poolUsageData.getPostRequest() + "\n" +
                                "DeleteRequest: " + poolUsageData.getDeleteRequest() + "\n"
+
                                "OtherRequest: " + poolUsageData.getOtherRequest() + "\n" +
                                "NonInternetUpload: " +
poolUsageData.getNonInternetUpload() + "\n" +
                                "NonInternetDownload: " +
poolUsageData.getNonInternetDownload() + "\n" +
                                "NonInternetDeleteFlow: " +
poolUsageData.getNonInternetDeleteFlow() + "\n" +
                                "NonInternetRoamUpload: " +
poolUsageData.getNonInternetRoamUpload() + "\n" +
                                "NonInternetRoamDownload: " +
poolUsageData.getNonInternetRoamDownload() + "\n" +
                                "NonInternetGeRequest: " +
poolUsageData.getNonInternetGetRequest() + "\n" +
                                "NonInternetHeadRequest: " +
poolUsageData.getNonInternetHeadRequest() + "\n" +
                                "NonInternetPutRequest: " +
poolUsageData.getNonInternetPutRequest() + "\n" +
                                "NonInternetPostRequest: " +
poolUsageData.getNonInternetPostRequest() + "\n" +
                                "NonInternetDeleteRequest: " +
poolUsageData.getNonInternetDeleteRequest() + "\n" +
                                "NonInternetOtherRequest: " +
poolUsageData.getNonInternetOtherRequest()
                        );
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }
```

5.10 GetBandWidth

此操作用来查询用户的已用带宽。

说明：不建议使用此操作，建议根据要查询的项，使用下列接口：GetCapacity、GetDeleteCapcity、GetRequests、GetReturnCode、GetConcurrentConnection。

示例代码

```
private static void getBandWidth(AmazonS3 client) {
        CtyunGetBandwidthRequest request = new CtyunGetBandwidthRequest();
        Date date = new Date();
        // Set the start time to three hours ago
        request.setBeginDate(new Date(date.getTime() - 1000 * 3600 * 3));
        // Set the end time to the current time
        request.setEndDate(date);
        request.setBucketName(BUCKET_NAME);

        try {
            CtyunGetBandwidthResult result = client.ctyunGetBandwidth(request);
            System.out.println(
                    "UserName: " + result.getUserName() + "\n" +
                    "BucketName: " + result.getBucketName()
            );
            for (CtyunPoolBandwidth poolBandwidth : result.getDetailBandwidth()) {
                if (poolBandwidth.getCtyunPoolBandwidthData() != null) {
                    System.out.println("PoolName: " + poolBandwidth.getPoolName());
                    for (CtyunPoolBandwidthData poolBandwidthData :
poolBandwidth.getCtyunPoolBandwidthData()) {
                        System.out.println(
                                "Date: " + poolBandwidthData.getDate() + "\n" +
                                "UploadBW: " + poolBandwidthData.getUploadBW() + "\n" +
                                "DownloadBW: " + poolBandwidthData.getDownloadBW() + "\n" +
                                "RoamUploadBW: " + poolBandwidthData.getRoamUploadBW() +
"\n" +
                                "RoamDownloadBW: " + poolBandwidthData.getRoamDownloadBW()
+ "\n" +
```

```
                                "NonInternetUploadBW: " +
poolBandwidthData.getNonInternetRoamUploadBW() + "\n" +
                                "NonInternetDownloadBW: " +
poolBandwidthData.getNonInternetRoamDownloadBW() + "\n" +
                                "NonInternetRoamUploadBW: " +
poolBandwidthData.getNonInternetRoamUploadBW() + "\n" +
                                "NonInternetRoamDownloadBW: " +
poolBandwidthData.getNonInternetRoamDownloadBW()
                        );
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

6 操作跟踪服务代码示例

操作跟踪服务代码示例在 example 的子目录：\src\cn\ctyun\test\CloudTrailTest.java。

注意：要设置成统计分析（CloudTrail API）服务的 Endpoint。

设置 trailclient 的示例：

```
private static CloudTrailClient getCloudTrailClient() {
        ClientConfiguration clientConfig = new ClientConfiguration();
        // 设置 https，仅支持 https
        clientConfig.setProtocol(Protocol.HTTPS);
        // 使用 v4 签名
        System.setProperty(SDKGlobalConfiguration.ENABLE_S3_SIGV4_SYSTEM_PROPERTY,
"true");
        System.setProperty(SDKGlobalConfiguration.ENABLE_STS_SIGV4_SYSTEM_PROPERTY,
"true");
        // 创建 client,其中 OOSCredentials.properties 中存放着用户名和密码
        CloudTrailClient trailClient = new CloudTrailClient(new
PropertiesCredentials(TestConfig.OOS_ACCESS_KEY, TestConfig.OOS_SECRET_KEY),
clientConfig);
        // 设置 endpoint
        trailClient.setEndpoint(TestConfig.OOS_ENDPOINT_TRAIL);
        return trailClient;
    }
```

6.1 CreateTrail

此操作用来创建一个新的跟踪。

示例代码

```
private static void createTrail(CloudTrailClient client) {
        // Initialize the createTrail operation request
        CreateTrailRequest request = new CreateTrailRequest();
        // Parameter assignment
        request.setName(TRAIL_NAME);
        request.setS3BucketName(BUCKET_NAME);
        request.setS3KeyPrefix(PREFIX);
        try {
            // Execute the createTrail request and obtain the result
            CreateTrailResult result = client.createTrail(request);
            // Output the result
            System.out.println("createTrail success! " +
                    "\n Name:" + result.getName() +
                    "\t S3BucketName :" + result.getS3BucketName() +
                    "\t S3KeyPrefix :" + result.getS3KeyPrefix() +
                    "\t TrailARN :" + result.getTrailARN());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

6.2 DeleteTrail

此操作用来删除指定的跟踪。

示例代码

```
private static void deleteTrail(CloudTrailClient client) {
        DeleteTrailRequest request = new DeleteTrailRequest();
        request.setName(TRAIL_NAME);
        try {
            client.deleteTrail(request);
            System.out.println("delete " + TRAIL_NAME + " success!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

6.3 DescribeTrails

此操作用来获取操作跟踪的设置信息。

示例代码

```
private static void describeTrails(CloudTrailClient client) {
       DescribeTrailsRequest request = new DescribeTrailsRequest();
       // Build a TRAIL_NAME list to query information for multiple trails
       ArrayList<String> TRAIL_NAMEList = new ArrayList<>();
       TRAIL_NAMEList.add(TRAIL_NAME);
       TRAIL_NAMEList.add("otherTrailName");
       request.setTrailNameList(TRAIL_NAMEList);
       try {
           DescribeTrailsResult result = client.describeTrails(request);
           int index = 1;
           System.out.println("describeTrails success!" +
                   "\n" + "describeTrails:");
           for (Trail trail : result.getTrailList()) {
               System.out.println("index: " + index + "\t Name:" + trail.getName() + "\t
S3BucketName: " + trail.getS3BucketName()
                       + "\t S3KeyPrefix: " + trail.getS3KeyPrefix() + "\t TrailARN: " +
trail.getTrailARN());
               index++;
           }
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

6.4 GetTrailStatus

此操作用来获取跟踪状态信息。

示例代码

```
private static void getTrailStatus(CloudTrailClient client) {
       GetTrailStatusRequest request = new GetTrailStatusRequest();
       request.setName(TRAIL_NAME);
       try {
           GetTrailStatusResult result = client.getTrailStatus(request);
           System.out.println("getTrailStatus success!" +
                   "\n" + "TrailStatus:" +
                   "\n" + "IsLogging: " + result.getIsLogging() +
                   "\t LatestDeliveryTime :" + result.getLatestDeliveryTime() +
                   "\t StartLoggingTime :" + result.getStartLoggingTime() +
                   "\t StopLoggingTime :" + result.getStopLoggingTime());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

6.5 PutEventSelectors

此操作用来配置跟踪的管理事件筛选器。

示例代码

```
private static void putEventSelectors(CloudTrailClient client) {
        PutEventSelectorsRequest request = new PutEventSelectorsRequest();
        request.setTrailName(TRAIL_NAME);
        // Build an EventSelector list
        // Since only management events are available at the moment, only one event
selector can be set
        ArrayList<EventSelector> eventSelectorList = new ArrayList<>();
        EventSelector eventSelector1 = new EventSelector();
        // All|ReadOnly|WriteOnly
        eventSelector1.setReadWriteType("WriteOnly");
        eventSelectorList.add(eventSelector1);
        request.setEventSelectors(eventSelectorList);
        try {
            PutEventSelectorsResult result = client.putEventSelectors(request);
            System.out.println("putEventSelectors success!");
            for (EventSelector eventSelector : result.getEventSelectors()) {
                System.out.println("EventSelector:" +
                        "\n" + "ReadWriteType: " + eventSelector.getReadWriteType());
            }
            System.out.println("TrailARN: " + result.getTrailARN());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

6.6 GetEventSelectors

此操作用来查看管理事件筛选器的设置信息。

示例代码

```
private static void getEventSelectors(CloudTrailClient client) {
       GetEventSelectorsRequest request = new GetEventSelectorsRequest();
       request.setTrailName(TRAIL_NAME);
       try {
           GetEventSelectorsResult result = client.getEventSelectors(request);
           System.out.println("getEventSelectors success!");
           for (EventSelector eventSelector : result.getEventSelectors()) {
               System.out.println("EventSelector:" +
                       "\n" + "ReadWriteType: " + eventSelector.getReadWriteType());
           }
           System.out.println("TrailARN: " + result.getTrailARN());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

6.7 UpdateTrail

此操作用来更新跟踪设置参数，包括跟踪日志数据保存的 OOS Bucket、跟踪日志前缀等。

示例代码

```
private static void updateTrail(CloudTrailClient client) {
       UpdateTrailRequest request = new UpdateTrailRequest();
       // Specify the trail name that needs to be updated
       request.setName(TRAIL_NAME);
       // Set the content that needs to be updated
       request.setS3BucketName(BUCKET_NAME);
       request.setS3KeyPrefix("");
       try {
           UpdateTrailResult result = client.updateTrail(request);
           System.out.println("updateTrail success!" +
                   "\n" + "Name: " + result.getName() +
                   "\t S3BucketName: " + result.getS3BucketName() +
                   "\t S3KeyPrefix: " + result.getS3KeyPrefix() +
                   "\t TrailARN: " + result.getTrailARN());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

6.8 StartLogging

此操作用来开启跟踪。

说明：使用 CreateTrail 创建跟踪后，跟踪默认是关闭状态，需要调用 StartLogging 开启跟踪。

示例代码

```
private static void startLogging(CloudTrailClient client) {
        StartLoggingRequest request = new StartLoggingRequest();
        request.setName(TRAIL_NAME);
        try {
            client.startLogging(request);
            System.out.println("starting logging.....");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

6.9 StopLogging

此操作用来停止跟踪功能。

示例代码

```
private static void stopLogging(CloudTrailClient client) {
       StopLoggingRequest request = new StopLoggingRequest();
       request.setName(TRAIL_NAME);
       try {
           client.stopLogging(request);
           System.out.println("stopping logging.....");
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

6.10 LookupEvents

此操作用来查看账户中的管理事件。用户可以查看近 6 个月内发生的管理事件。

示例代码

```
private static void lookupEvents(CloudTrailClient client) {
        LookupEventsRequest request = new LookupEventsRequest();

        // Set the query time range
        request.setStartTime(new Date(System.currentTimeMillis() - 1000 * 60 * 100));
        request.setEndTime(new Date(System.currentTimeMillis() + 1000 * 60 * 15));

        // Set the query attributes. currently, only one query attribute is allowed
        List<LookupAttribute> lookupAttributeList = new ArrayList<>();
        LookupAttribute lookupAttribute = new LookupAttribute();
        // Set the property key for querying management events
        lookupAttribute.setAttributeKey("EventName");
        // Set the property value for querying management events
        lookupAttribute.setAttributeValue("value1");
        lookupAttributeList.add(lookupAttribute);
        request.setLookupAttributes(lookupAttributeList);

// The number of results returned, ranging from 1 to 50. The default is to return 50 results
        request.setMaxResults(30);

        // Set the token
        request.setNextToken("nexttoken");
        try {
            LookupEventsResult result = client.lookupEvents(request);
            System.out.println("lookupEvents success!");
            for (CloudTrailEvent event : result.getEvents()) {
                System.out.println("Event:" +
                        "\n" + "CloudTrailEvent: " + event.toString());
            }
            if (result.getNextToken() != null) {
                System.out.println("NextToken: " + result.getNextToken());
```

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

7 IAM 服务代码示例

IAM 的服务代码示例是 example 的子目录：\src\cn\ctyun\test\IAMtest.java。

设置 IAM client 示例：

```
private static AmazonIdentityManagement getIamClient() {
        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
        clientConfig.setProtocol(Protocol.HTTPS); // 必须设置 https

        // 必须使用 V4 签名
        System.setProperty(SDKGlobalConfiguration.ENABLE_S3_SIGV4_SYSTEM_PROPERTY,
"true");

        /* 创建 client,其中 OOSCredentials.properties 中存放着用户名和密码 */
        AmazonIdentityManagement iamClient = new AmazonIdentityManagementClient(
                new PropertiesCredentials(TestConfig.OOS_ACCESS_KEY,
TestConfig.OOS_SECRET_Key), clientConfig);

        // 设置 endpoint IAM
        iamClient.setEndpoint(TestConfig.OOS_IAM_ENDPOINT_ACCESS);

        return iamClient;
    }
```

7.1 用户管理接口

7.1.1 CreateUser

此操作用来创建新的 IAM 用户。

示例代码

```
private static void createUser(AmazonIdentityManagement client) {
       CreateUserRequest request = new CreateUserRequest();
       request.setUserName("user1");
       // The list of tags to be attached to the newly created user
       SdkInternalList<Tag> tags = new SdkInternalList<Tag>();
       Tag tag = new Tag();
       tag.setKey("key1");// The key of the tag
       tag.setValue("value2");// The value of the tag
       tags.add(tag);
       Tag tag1 = new Tag();
       tag1.setKey("key2");
       tag1.setValue("value2");
       tags.add(tag1);
       request.setTags(tags);
       try {
           CreateUserResult result = client.createUser(request);
           System.out.println("createUser success!" +
                   "\n" + result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.1.2 GetUser

此操作用来获取 IAM 用户信息。如果未指定用户名，根据签名认证中的 AccessKeyID 来决定用户名。

示例代码

```
private static void getUser(AmazonIdentityManagement client) {
        GetUserRequest request = new GetUserRequest();
        request.setUserName("user1");
        try {
            GetUserResult result = client.getUser(request);
            System.out.println(result.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.1.3 ListUsers

此操作用来列出 IAM 用户。如果账户中没有创建 IAM 用户，则返回空列表。

示例代码

```
private static void listUser(AmazonIdentityManagement client) {
        ListUsersRequest request = new ListUsersRequest();
        // Perform a fuzzy search for userName and accessKeyId to query results that
satisfy multiple query conditions simultaneously
        request.setUserName("user1");
        // Set it to the value of the Marker element in the response
        // that you received to indicate where the next call should start
//        request.setMarker("");
        // The maximum number of items returned in the response
//        request.setMaxItems(10);
        // Specifies the AccessKeyId, perform fuzzy matching search according to
AccessKeyId
//        request.setAccessKeyId("");
        try {
            ListUsersResult result = client.listUsers(request);
            System.out.println(result.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.1.4 DeleteUser

此操作用来删除指定的 IAM 用户。

示例代码

```
private static void deleteUser(AmazonIdentityManagement client) {
       DeleteUserRequest request = new DeleteUserRequest();
       request.setUserName("user1");
       try {
           client.deleteUser(request);
           System.out.println(request + " deleteUser finish");
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.1.5 TagUser

此操作用来为 IAM 用户添加标签。

说明：

- 可以同时添加一个或多个标签，一个 IAM 用户最多能添加 10 个标签。
- 如果 Tags.member.N.Key 已经存在，其值则会被新添加的 value 覆盖。

示例代码

```
private static void tagUser(AmazonIdentityManagement client) {
        TagUserRequest request = new TagUserRequest();
        request.setUserName("user1");
        // Tags for the new user, with each tag containing a key and a value.
        // A maximum of 10 tags can be created, which can be used to find or retrieve
associated values by key
        SdkInternalList<Tag> tags = new SdkInternalList<Tag>();
        Tag tag = new Tag();
        // User tag key. Expression: [\p{L}\p{Z}\p{N}_.:/=+\-@]+
        tag.setKey("key1");
        // User tag value. Expression: [\p{L}\p{Z}\p{N}_.:/=+\-@]*
        tag.setValue("value1");
        tags.add(tag);
        Tag tag1 = new Tag();
        tag1.setKey("key2");
        tag1.setValue("v2");
        tags.add(tag1);
        request.setTags(tags);
        try {
            client.tagUser(request);
            System.out.println("tagUser success!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.1.6 UntagUser

此操作用来删除用户的指定标签。

示例代码

```
private static void untagUser(AmazonIdentityManagement client) {
       UntagUserRequest request = new UntagUserRequest();
       // Delete the tag under the user 'user1'
       request.setUserName("user1");
       SdkInternalList<String> tags = new SdkInternalList<>();
       // Delete the key2 tag
       tags.add("key2");
       request.setTagKeys(tags);
       try {
           client.untagUser(request);
           System.out.println("untagUser success!");
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.1.7 ListUserTags

此操作用来列出指定 IAM 用户的标签。

示例代码

```
private static void listUserTags(AmazonIdentityManagement client) {
       ListUserTagsRequest request = new ListUserTagsRequest();
       // List all tags under the user 'user1'
       request.setUserName("user1");
       // Set it to the value of the Marker element in the response
       // that you received to indicate where the next call should start
//       request.setMarker("");
       // The maximum number of items returned in the response
//       request.setMaxItems(10);
       try {
           ListUserTagsResult result = client.listUserTags(request);
           System.out.println(result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.1.8 ListGroupsForUser

此操作用来列出指定 IAM 用户所属的 IAM 用户组。

示例代码

```
private static void listGroupsForUser(AmazonIdentityManagement client) {
       ListGroupsForUserRequest request = new ListGroupsForUserRequest();
       // List all groups under the user 'user1'
       request.setUserName("user1");
       // Set it to the value of the Marker element in the response
       // that you received to indicate where the next call should start
//       request.setMarker("");
       // The maximum number of items returned in the response
//       request.setMaxItems(10);
       try {
           ListGroupsForUserResult result = client.listGroupsForUser(request);
           System.out.println(result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.1.9 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

```
private static void createAccessKey(AmazonIdentityManagement client) {
       try {
           CreateAccessKeyRequest request = new CreateAccessKeyRequest();
           // The name of the IAM user
//           request.setUserName("");
           CreateAccessKeyResult result = client.createAccessKey(request);
           System.out.println("createAccessKey success!" +
                   "\n" + result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.1.10 ListAccessKeys

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

```
private static void listAccessKey(AmazonIdentityManagement client) {
       try {
           ListAccessKeysRequest request = new ListAccessKeysRequest();
           // The name of the IAM user
//           request.setUserName("");
           // Set it to the value of the Marker element in the response
           // that you received to indicate where the next call should start
//           request.setMarker("");
           // The maximum number of items returned in the response
//           request.setMaxItems(10);
           ListAccessKeysResult result = client.listAccessKeys(request);
           System.out.println(result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.1.11 GetAccessKeyLastUsed

此操作用来查询指定密钥最后一次使用的时间及服务名称。

示例代码

```
private static void getAccessKeyLastUsed(AmazonIdentityManagement client) {
       GetAccessKeyLastUsedRequest request = new GetAccessKeyLastUsedRequest();
       request.setAccessKeyId("");
       try {
           GetAccessKeyLastUsedResult result = client.getAccessKeyLastUsed(request);
           System.out.println(result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.1.12 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。
- 调用此操作可以管理根用户的密钥。

示例代码

```
private static void updateAccessKey(AmazonIdentityManagement client) {
        UpdateAccessKeyRequest request = new UpdateAccessKeyRequest();
        // The name of the IAM user whose AccessKey will be updated
//        request.setUserName("");
        // AccessKeyID
//        request.setAccessKeyId("");
        // The status of the access key, values: Active/Inactive
//        request.setStatus("Active");
        try {
            client.updateAccessKey(request);
            System.out.println(request + " updateAccessKey success!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.1.13 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。
- 调用此操作也可以用来删除根用户的 AccessKey。

示例代码

```
private static void deleteAccessKey(AmazonIdentityManagement client) {
        DeleteAccessKeyRequest request = new DeleteAccessKeyRequest();
        // The name of IAM user whose AccessKey will be deleted
//        request.setUserName("");
        // The ID of the AccessKey to be deleted
//        request.setAccessKeyId("");
        try {
            client.deleteAccessKey(request);
            System.out.println(request + " deleteAccessKey success!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.1.14 GetSessionToken

OOS 为用户提供临时授权访问。此操作用来获取临时访问密钥。子用户默认拥有调用此操作的权限。如果配置了禁止子用户调用该接口的 IAM 策略，该策略不会生效。

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，可以为第三方应用或用户颁发一个自定义时效的访问凭证。第三方应用或用户可以使用该访问凭证直接调用 OOS API，或者使用 OOS 提供的 SDK 来访问 OOS API。

使用临时授权访问 OOS API 时，用户需要将安全令牌（SessionToken）携带在请求 header 中或者预签名 URL 中。携带在请求 header 中，V4 和 V2 签名的 X-Amz-Security-Token 请求头不区分大小写。携带在预签名 URL 中，V4 签名的标头为“X-Amz-Security-Token”，V2 签名的标头为“x-amz-security-token”。

注意：使用临时访问令牌（AccessKeyId、SecretAccessKey、SessionToken）的用户不能调用该接口。使用临时访问令牌调用其他接口时，权限同生成该临时访问令牌的用户。为了安全起见，不建议根用户调用该接口。

示例代码

```
private static void getSessionToken(AmazonIdentityManagement client) {
        GetSessionTokenRequest request = new GetSessionTokenRequest();
        request.setDurationSeconds(7200); // Unit: seconds
        String policyText = "   {   "
                + "        'Version':'2012-10-17',   "
                + "        'Id':'http referer policy example',   "
                + "              'Statement':[   "
                + "            {   "
                + "             'Sid':'1',   "
                + "                'Effect':'Deny',   "
//              + "                'Effect':'Allow',   "
                + "                'Principal':{   "
                + "                     'AWS':[   "
                + "                      '*'   "
                + "                          ]   "
                + "                  },   "
                + "               'Action':'s3:*',   "
```

```
            + "                'Resource':'arn:aws:s3:::" + "BUCKET_NAME" + "',   "
            + "                'Condition':{   "
            + "                    'StringLike':{   "
            + "                        'aws:Referer':[   "
            + "                            '*'  "
            + "                        ]  "
            + "                 }  "
            + "             }  "
            + "          }  "
            + "       ]  "
            + "     }  ";
        request.setPolicyDocument(policyText);
        try {
            GetSessionTokenResult result = client.getSessionToken(request);
            System.out.println(result.getCredentials().getAccessKeyId());
            System.out.println(result.getCredentials().getSecretAccessKey());
            System.out.println(result.getCredentials().getSessionToken());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.1.15 CreateLoginProfile

此操作用来为指定 IAM 用户创建控制台的登录密码。

示例代码

```
private static void createLoginProfile(AmazonIdentityManagement client) {
        CreateLoginProfileRequest request = new CreateLoginProfileRequest();
        request.setUserName("user1");
        // Login password
        request.setPassword("");
        // Whether the IAM user needs to reset the password after logging into the
console with the initial password
        // The default value is false
        request.setPasswordResetRequired(true);
        try {
            CreateLoginProfileResult result = client.createLoginProfile(request);
            System.out.println("createLoginProfile success!" +
                    "\n" + result.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.1.16 GetLoginProfile

此操作用来获取 IAM 用户控制台登录密码创建的时间、及用户首次登录后是否需要修改密码。

示例代码

```
private static void getLoginProfile(AmazonIdentityManagement client) {
        GetLoginProfileRequest request = new GetLoginProfileRequest();
        request.setUserName("user1");
        try {
            GetLoginProfileResult result = client.getLoginProfile(request);
            System.out.println(result.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.1.17 UpdateLoginProfile

此操作用来更改指定 IAM 用户控制台的登录密码。

示例代码

```
private static void updateLoginProfile(AmazonIdentityManagement client) {
        UpdateLoginProfileRequest request = new UpdateLoginProfileRequest();
        request.setUserName("user1");
        request.setPassword("");
        // Whether the IAM user needs to reset the password after logging into the
console with the initial password
        request.setPasswordResetRequired(true);
        try {
            client.updateLoginProfile(request);
            System.out.println(request + " updateLoginProfile success!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.1.18 DeleteLoginProfile

此操作用来删除指定 IAM 用户控制台的登录密码。调用此操作后，指定用户将不能通过控制台进行 OOS 服务。

示例代码

```
private static void deleteLoginProfile(AmazonIdentityManagement client) {
        DeleteLoginProfileRequest request = new DeleteLoginProfileRequest();
        request.setUserName("user1");
        try {
            client.deleteLoginProfile(request);
            System.out.println(request + " deleteLoginProfile success!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.1.19 ChangePassword

更改正在调用此操作的 IAM 用户的密码。调用此操作，不会影响 OOS root 用户密码。

示例代码

```
private static void changePassword(AmazonIdentityManagement client) {
       ChangePasswordRequest request = new ChangePasswordRequest();
       request.setNewPassword("");
       request.setOldPassword("");
       try {
           client.changePassword(request);
           System.out.println(request + " changePassword success!");
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.1.20 CreateVirtualMFADevice

此操作用来创建虚拟 MFA 设备。创建虚拟 MFA 后，可以使用 EnableMFADevice 启用虚拟 MFA 设备，并将该虚拟 MFA 设备与指定的 IAM 用户关联。

注意：QR 代码和 Base32 字符串中包含的 MFA 设备密钥，就像您的密码一样，应被妥善保管和处理。

示例代码

```
private static void createVirtualMFADevice(AmazonIdentityManagement client) {
        String deviceName = "MFAName";
        CreateVirtualMFADeviceRequest request = new CreateVirtualMFADeviceRequest();
        request.setVirtualMFADeviceName(deviceName);
        try (FileOutputStream fos = new FileOutputStream("")) {
            CreateVirtualMFADeviceResult result = client.createVirtualMFADevice(request);
            // Obtain the string-type MFA device key
            System.out.println("MFA device key: " +
result.getVirtualMFADevice().getBase32StringSeedStr());
            // Obtain the ByteBuffer type MFA device key
            java.nio.ByteBuffer bbf = result.getVirtualMFADevice().getBase32StringSeed();
            // Obtain the serial number that uniquely identifies the MFA device
            System.out.println("MFA serialNumber: " +
result.getVirtualMFADevice().getSerialNumber());
            // Generate the MFA key QR code
            fos.write(result.getVirtualMFADevice().getQRCodePNG().array());
        } catch (AmazonClientException | IOException e) {
            e.printStackTrace();
        }
    }
```

7.1.21 EnableMFADevice

此操作用来启用指定的虚拟 MFA 设备，并将该虚拟 MFA 设备与指定的 IAM 用户关联。

示例代码

```
private static void enableMFADevice(AmazonIdentityManagement client) {
        EnableMFADeviceRequest request = new EnableMFADeviceRequest();
        // The verification code issued by the virtual MFA device is a six-digit code
        request.setAuthenticationCode1("");
        // The six-digit verification code issued by the MFA device following
AuthenticationCode1
        request.setAuthenticationCode2("");
        // The serial number that uniquely identifies the MFA device.
        // For virtual MFA devices, the serial number is the device ARN
        request.setSerialNumber("");
        request.setUserName("user1");
        try {
            client.enableMFADevice(request);
            System.out.println(request + " enableMFADevice success!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.1.22 ListVirtualMFADevices

此操作用来按分配状态列出 OOS 账户中定义的虚拟 MFA 设备。如果未指定分配状态，则将返回所有虚拟 MFA 设备的列表。

示例代码

```
private static void ListVirtualMFADevices(AmazonIdentityManagement client) {
       try {
           ListVirtualMFADevicesRequest request = new ListVirtualMFADevicesRequest();
           // Specifies the assignment status of the virtual MFA device to be listed
//           request.setAssignmentStatus(AssignmentStatusType.Assigned);
           // Set it to the value of the Marker element in the response
           // that you received to indicate where the next call should start
//           request.setMarker("");
           // The maximum number of items returned in the response
//           request.setMaxItems(10);
           ListVirtualMFADevicesResult result = client.listVirtualMFADevices();
           System.out.println(result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.1.23 ListMFADevices

此操作用来列出 IAM 用户的虚拟 MFA 设备。

说明：如果请求中没有指定用户，则会列出请求用户自己名下的虚拟 MFA 设备。

示例代码

```
private static void ListMFADevices(AmazonIdentityManagement client) {
        ListMFADevicesRequest request = new ListMFADevicesRequest();
        // The name of the IAM user
        request.setUserName("user1");
        // Set it to the value of the Marker element in the response that
        // you received to indicate where the next call should start
//        request.setMarker("");
        // The maximum number of items returned in the response
//        request.setMaxItems(10);
        try {
            ListMFADevicesResult result = client.listMFADevices(request);
            System.out.println(result.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.1.24 DeactivateMFADevice

此操作用来终止使用指定的 MFA 设备，并与用户解除关联。

示例代码

```
private static void deactivateMFADevice(AmazonIdentityManagement client) {
       DeactivateMFADeviceRequest request = new DeactivateMFADeviceRequest();
       request.setUserName("user1");
       request.setSerialNumber("");
       try {
           client.deactivateMFADevice(request);
           System.out.println(request + " deactivateMFADevice success!");
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.1.25 DeleteVirtualMFADevice

此操作用来删除指定的虚拟 MFA 设备。

示例代码

```
private static void deleteVirtualMFADevice(AmazonIdentityManagement client) {
       DeleteVirtualMFADeviceRequest request = new DeleteVirtualMFADeviceRequest();
       request.setSerialNumber("");
       try {
           client.deleteVirtualMFADevice(request);
           System.out.println(request + " deleteVirtualMFADevice success!");
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.2 用户组管理接口

7.2.1 CreateGroup

此操作用来创建新的 IAM 用户组。

示例代码

```
private static void createGroup(AmazonIdentityManagement client) {
       CreateGroupRequest request = new CreateGroupRequest();
       request.setGroupName("groupName1");
       try {
           CreateGroupResult createGroupResult = client.createGroup(request);
           System.out.println("createGroup success!" +
                   "\n" + createGroupResult.getGroup().getArn());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.2.2 GetGroup

此操作用来获取指定 IAM 用户组及组内 IAM 用户列表。

示例代码

```
private static void getGroup(AmazonIdentityManagement client) {
       GetGroupRequest request = new GetGroupRequest();
       // The name of the IAM user group
       request.setGroupName("groupName1");
       // Set it to the value of the Marker element in the response that
       // you received to indicate where the next call should start
//       request.setMarker("");
       // The maximum number of items returned in the response
//       request.setMaxItems(10);
       try {
           GetGroupResult result = client.getGroup(request);
           System.out.println(result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.2.3 AddUserToGroup

此操作用来将指定的 IAM 用户加入到指定的 IAM 用户组，每次只能将一个用户加入到指定用户组。

说明：多次调用此操作将同一个用户加入同一个组，不会报错。

示例代码

```
private static void addUserToGroup(AmazonIdentityManagement client) {
        AddUserToGroupRequest request = new AddUserToGroupRequest();
        request.setGroupName("groupName1");
        request.setUserName("user1");
        try {
            client.addUserToGroup(request);
            System.out.println("addUserToGroup success!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.2.4 RemoveUserFromGroup

此操作用来将指定用户从指定用户组移除。

示例代码

```
private static void removeUserFromGroup(AmazonIdentityManagement client) {
       RemoveUserFromGroupRequest request = new RemoveUserFromGroupRequest();
       request.setGroupName("groupName1");
       request.setUserName("user1");
       try {
           client.removeUserFromGroup(request);
           System.out.println(request + " removeUserFromGroup success!");
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.2.5 ListGroups

此操作用来列出所有的 IAM 用户组。

示例代码

```
private static void listGroups(AmazonIdentityManagement client) {
        ListGroupsRequest request = new ListGroupsRequest();
        request.setGroupName("groupName1");
        // This parameter should only be used when paginating results and only after
receiving a response indicating that the results have been truncated
        // Set it to the value of the Marker element from the response you received, to
indicate where the next call should start
        request.setMarker("marker");
        // Set the maximum number of items you want in the response
        // If there are additional items exceeding the maximum you specified, the
IsTruncated response element will be true
        // If this parameter is not included, the default number of items will be 100
        request.setMaxItems(100);
        try {
            ListGroupsResult result = client.listGroups(request);
            System.out.println(result.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.2.6 DeleteGroup

此操作用来删除指定的 IAM 用户组。

示例代码

```
private static void deleteGroup(AmazonIdentityManagement client) {
        DeleteGroupRequest request = new DeleteGroupRequest();
        request.setGroupName("groupName1");
        try {
            client.deleteGroup(request);
            System.out.println(request + " deleteGroup success!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.3 权限策略管理接口

7.3.1 CreatePolicy

此操作用来为账户创建策略。如果策略名已存在，再创建同一名称的策略，后创建的策略会将已存在的同名策略覆盖。

示例代码

```
private static void createPolicy(AmazonIdentityManagement client) {
        CreatePolicyRequest request = new CreatePolicyRequest();
        request.setPolicyName("policyName");
        request.setPolicyDocument("{\"Version\":\"2012-10-17\""
                +
",\"Statement\":[{\"Effect\":\"Allow\",\"Action\":\"*\",\"Resource\":\"*\"}]}");
        // The description of the policy
//        request.setDescription("");
        try {
            CreatePolicyResult result = client.createPolicy(request);
            System.out.println("createPolicy success!" +
                    "\n" + result.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.3.2 GetPolicy

此操作用来获取策略相关信息。

示例代码

```
private static void getPolicy(AmazonIdentityManagement client) {
       GetPolicyRequest request = new GetPolicyRequest();
       // This can be obtained through the createPolicy interface
       request.setPolicyArn("");
       try {
           GetPolicyResult result = client.getPolicy(request);
           System.out.println(result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.3.3 ListPolicies

此操作用来列出账户下所有的策略。

示例代码

```
private static void listPolicies(AmazonIdentityManagement client) {
        ListPoliciesRequest request = new ListPoliciesRequest();
        // Pagination token. This parameter will be returned in the previous response
when there are additional policies to return
        // When viewing hidden items, this parameter needs to be included in the request
parameters
        request.setMarker("policyName");
        // Set the maximum number of items to return in the response
        // When viewing hidden items, you need to include the value of the Marker
response parameter
        request.setMaxItems(10);
        // Used to indicate whether to display only the policies associated with IAM
users or IAM user groups. The default value is false
        request.setOnlyAttached(true);
        // Permission policy name
        // It can be queried with fuzzy matching
        request.setPolicyName("testPolicy");
        // Scope used to filter policies, default is All: Local | OOS | All
        request.setScope("Local");
        try {
            ListPoliciesResult result = client.listPolicies(request);
            System.out.println(result.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.3.4 ListEntitiesForPolicy

此操作用来列出指定策略所附加的所有 IAM 用户或 IAM 用户组。

示例代码

```
private static void listEntitiesForPolicy(AmazonIdentityManagement client) {
       ListEntitiesForPolicyRequest request = new ListEntitiesForPolicyRequest();
       // This can be obtained through the createPolicy interface
       request.setPolicyArn("");
       // Specifies the type of the entity to filter
//       request.setEntityFilter("");
       // Set it to the value of the Marker element in the response
       // that you received to indicate where the next call should start
//       request.setMarker("");
       // The maximum number of items returned in the response
//       request.setMaxItems(10);
       try {
           ListEntitiesForPolicyResult result = client.listEntitiesForPolicy(request);
           System.out.println(result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.3.5 DeletePolicy

此操作用来删除指定的策略。

注意：在删除策略前，要确保该策略没有附加到任何 IAM 用户或 IAM 用户组，可以按照下列步骤进行解除关联 IAM 用户和 IAM 用户组

1. 使用 ListEntitiesForPolicy 接口查看关联的 IAM 用户和用户组。
2. 使用 DetachUserPolicy 解除策略关联的 IAM 用户，使用 DetachGroupPolicy 解除策略关联的 IAM 用户组。

示例代码

```
private static void deletePolicy(AmazonIdentityManagement client) {
       DeletePolicyRequest request = new DeletePolicyRequest();
       // This can be obtained through the createPolicy interface
       request.setPolicyArn("");
       try {
           client.deletePolicy(request);
           System.out.println(request + " deletePolicy success!");
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.3.6 AttachUserPolicy

此操作用来将指定的策略与指定的 IAM 用户关联。

示例代码

```
private static void attachUserPolicy(AmazonIdentityManagement client) {
        AttachUserPolicyRequest request = new AttachUserPolicyRequest();
        request.setUserName("user1");
        // This can be obtained through the createPolicy interface
        request.setPolicyArn("");
        try {
            AttachUserPolicyResult result = client.attachUserPolicy(request);
            System.out.println(result.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.3.7 ListAttachedUserPolicies

此操作用来列出与指定用户关联的策略。

示例代码

```
private static void listAttachedUserPolicies(AmazonIdentityManagement client) {
       ListAttachedUserPoliciesRequest request = new ListAttachedUserPoliciesRequest();
       // The name of the IAM user
       request.setUserName("user1");
       // Set it to the value of the Marker element in the response
       // that you received to indicate where the next call should start
//       request.setMarker("");
       // The maximum number of items returned in the response
//       request.setMaxItems(10);
       try {
           ListAttachedUserPoliciesResult result =
client.listAttachedUserPolicies(request);
           System.out.println(result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.3.8 DetachUserPolicy

此操作用来解除指定用户关联的指定策略。

示例代码

```
private static void detachUserPolicy(AmazonIdentityManagement client) {
       DetachUserPolicyRequest request = new DetachUserPolicyRequest();
       request.setUserName("user1");
       // This can be obtained through the createPolicy interface
       request.setPolicyArn("");
       try {
           client.detachUserPolicy(request);
           System.out.println(request + " detachUserPolicy success!");
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.3.9 AttachGroupPolicy

此操作用来将指定的策略与指定的 IAM 用户组关联。

示例代码

```
private static void attachGroupPolicy(AmazonIdentityManagement client) {
        AttachGroupPolicyRequest request = new AttachGroupPolicyRequest();
        request.setGroupName("groupName");
        // This can be obtained through the createPolicy interface
        request.setPolicyArn("");
        try {
            client.attachGroupPolicy(request);
            System.out.println("attachGroupPolicy success!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.3.10 ListAttachedGroupPolicies

此操作用来列出与指定 IAM 用户组关联的策略。

示例代码

```
private static void listAttachedGroupPolicies(AmazonIdentityManagement client) {
        ListAttachedGroupPoliciesRequest request = new ListAttachedGroupPoliciesRequest();
        // The name of the IAM group to list attached
        request.setGroupName("groupName");
        // Set it to the value of the Marker element in the response
        // that you received to indicate where the next call should start
//        request.setMarker("");
        // The maximum number of items returned in the response
//        request.setMaxItems(10);
        try {
            ListAttachedGroupPoliciesResult result =
client.listAttachedGroupPolicies(request);
            System.out.println(result.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.3.11 DetachGroupPolicy

此操作用来解除指定 IAM 用户组关联的指定策略。

示例代码

```
private static void detachGroupPolicy(AmazonIdentityManagement client) {
       DetachGroupPolicyRequest request = new DetachGroupPolicyRequest();
       request.setGroupName("groupName");
       // This can be obtained through the createPolicy interface
       request.setPolicyArn("");
       try {
           client.detachGroupPolicy(request);
           System.out.println(request + " detachGroupPolicy success!");
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.3.12 UpdateAccountPasswordPolicy

此操作用来更新账户的密码规则设置。

示例代码

```
private static void updateAccountPasswordPolicy(AmazonIdentityManagement client) {
        UpdateAccountPasswordPolicyRequest request = new UpdateAccountPasswordPolicyRequest();
// Whether IAM users are allowed to change their console passwords. The default value is true
        request.setAllowUsersToChangePassword(true);
        // Whether to prevent users from changing their passwords after the console password has expired
        request.setHardExpiry(true);
        // The password validity period for IAM users, ranging from 0 to 1095 days, where 0 means never expires. The default value is 0
        request.setMaxPasswordAge(100);
        // The minimum length for the console login password, ranging from 8 to 128. The default value is 8
        request.setMinimumPasswordLength(8);
        // When specifying a new login password for the IAM user, it cannot be the same as any of the previous X passwords
        // Value ranges from 0 to 24, where 0 means that IAM users are not prevented from setting a previous login password as the new login password
        // The default value is 0
        request.setPasswordReusePrevention(2);
        // Specify whether the console login password must include lowercase letters (a-z). The default value is true
        request.setRequireLowercaseCharacters(true);
        // Specify whether the console login password must include numbers (0-9). The default value is true.
        request.setRequireNumbers(true);
        // Specify whether the console login password must include special characters: ! @ # $ % ^ & * ( ) _ + - = [ ] { } | '
        // The default value is false
        request.setRequireSymbols(true);
        // Specify whether the console login password must include uppercase letters (A-Z). The default value is false
```

```
        request.setRequireUppercaseCharacters(true);
        try {
            client.updateAccountPasswordPolicy(request);
            System.out.println(request + " updateAccountPasswordPolicy success!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.3.13 GetAccountPasswordPolicy

此操作用来获取账户的密码策略。

示例代码

```
private static void getAccountPasswordPolicy(AmazonIdentityManagement client) {
       GetAccountPasswordPolicyRequest request = new GetAccountPasswordPolicyRequest();
       try {
           GetAccountPasswordPolicyResult result = client.getAccountPasswordPolicy(request);
           System.out.println(result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.3.14 DeleteAccountPasswordPolicy

此操作用来将账户的密码规则恢复到默认密码规则。

示例代码

```
private static void deleteAccountPasswordPolicy(AmazonIdentityManagement client) {
       DeleteAccountPasswordPolicyRequest request = new
DeleteAccountPasswordPolicyRequest();
       try {
           client.deleteAccountPasswordPolicy(request);
           System.out.println(request + " deleteAccountPasswordPolicy success!");
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.3.15 UpdateAccountLoginSecurityPolicy

此操作用来更新 IAM 用户登录安全策略设置。

实例代码

```
private static void updateAccountLoginSecurityPolicy(AmazonIdentityManagement client) {
       UpdateAccountLoginSecurityPolicyRequest request = new
UpdateAccountLoginSecurityPolicyRequest();
       // Optional.Set the time limit for failed login attempts. Unit: minutes, default
is 15
       request.setPeriodWithLoginFailures(15);
       // Optional.Set the number of consecutive login failures allowed for IAM users
within a specified time,
       // with a default value of 5 and a range of 5 to 15
       request.setLoginFailedTimes(5);
       // Optional.Set the duration for which IAM users are locked out, in minutes,
       // with a default of 15 and a range from 15 to 60
       request.setLockoutDuration(15);
       // Optional.Set whether IAM users are allowed to log in from multiple clients at
the same time; the default value is true
       request.setAllowSingleUsersSimultaneousLogin(true);
       // Optional.Set the duration to keep the session active when an IAM user logs
into the console and there is no activity, in minutes,
       // with a default of 30 and a range from 10 to 30
       request.setSessionDuration(30);
       try {
           client.updateAccountLoginSecurityPolicy(request);
           System.out.println(request + " updateAccountLoginSecurityPolicy success!");
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.3.16 GetAccountLoginSecurityPolicy

此操作用来获取 IAM 用户登录安全策略。

示例代码

```
private static void getAccountLoginSecurityPolicy(AmazonIdentityManagement client) {
       GetAccountLoginSecurityPolicyRequest request = new GetAccountLoginSecurityPolicyRequest();
       try {
           GetAccountLoginSecurityPolicyResult result =
client.getAccountLoginSecurityPolicy(request);
           System.out.println(result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

7.3.17 DeleteAccountLoginSecurityPolicy

此操作用来将 IAM 用户登录安全策略恢复为默认值。默认值为：

- PeriodWithLoginFailures：15 分钟。
- LoginFailedTimes：5。
- LockoutDuration：15 分钟。
- AllowSingleUsersSimultaneousLogin：true。
- SessionDuration：30 分钟

示例代码

```
private static void deleteAccountLoginSecurityPolicy(AmazonIdentityManagement client) {
        DeleteAccountLoginSecurityPolicyRequest request = new
DeleteAccountLoginSecurityPolicyRequest();
        try {
            client.deleteAccountLoginSecurityPolicy(request);
            System.out.println(request + " deleteAccountLoginSecurityPolicy success!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7.4 服务数量查询

7.4.1 GetAccountSummary

此操作用来获取账户中的实体数量和服务限制信息。

示例代码

```
private static void getAccountSummary(AmazonIdentityManagement client) {
       GetAccountSummaryRequest request = new GetAccountSummaryRequest();
       try {
           GetAccountSummaryResult result = client.getAccountSummary(request);
           System.out.println(result.toString());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
```

8 客户端加密代码示例

客户端加密是指数据发送到 OOS 之前，已经使用加密客户端对数据进行加密。将加密后的数据发送至 OOS 服务器时，同时将加密方法与解密所需的必要信息存储到 OOS 中。在下载时，使用加密客户端，OOS SDK 会根据您提供的密钥与解密必要的信息进行解密，直接返回解密后的数据。

使用客户端加密，能够有效地避免数据泄露，保护客户数据安全。

说明：OOS 服务端接收您的加密数据，不会对其进行加密或解密操作。

注意：

- 使用客户端加密功能时，您必须确保主密钥的完整性和正确性。如果因为主密钥丢失或者用错，导致加密数据无法解密，引起的一切损失和后果需要由您自行承担。
- 在对加密数据进行复制或数据迁移时，您必须确保加密元数据的完整性和正确性。如果因为主密钥丢失或者用错，导致加密数据无法解密，引起的一切损失和后果需要由您自行承担。
- 上传/分片上传时，使用加密客户端加密的文件最大不能超过 64 GiB。

8.1 加密方式

8.1.1 主密钥

可以选择下列两种加密算法生成主密钥：

- 对称加密密钥（AES）：当用户选取 AES 生成主密钥时，密钥封装算法为：AES/GCM。
- 非对称加密密钥（RSA）：当用户选取 RSA 生成主密钥时，密钥封装算法为：RSA-OAEP-SHA1。

说明：

- 主密钥只参与客户端本地计算，不会在网络上进行传输或保存在服务端，以保证主密钥的数据安全。
- 主密钥用于加密数据密钥，加密后的内容会作为 Object 的元数据保存在服务端。

8.1.2 数据密钥

使用客户端加密时，会为每个 Object 随机生成一个数据密钥。通过 AES/GCM 算法，使用该数据密钥对 Object 的数据进行对称加密。

8.2 加密元数据

注意：用户如果修改加密元数据信息，会导致文件无法正常解密。

参数	描述
x-amz-key-v2	加密后的数据密钥。
x-amz-iv	随机生成的初始化偏移量。
x-amz-unencrypted-content-length	未加密的 Object 内容长度。
x-amz-tag-len	使用 AEAD 加密认证（Authenticated Encryption with Associated Data）时的标记长度。 取值：“128”。
x-amz-matdesc	主密钥的描述，JSON 格式。 建议为每个主密钥都配置描述信息，并保存好主密钥和描述之间的关系。
x-amz-wrap-alg	使用的主密钥封装算法。 取值： ● 对称算法："AES/GCM/NoPadding"。 ● 非对称算法："RSA-OAEP-SHA1"。 如果不存在此元数据，则表示用户未使用标准的密钥封装算法。
x-amz-cek-alg	所使用的数据加密算法。 取值:“AES/GCM/NoPadding”。

8.3 创建主密钥示例

注意：在保存密钥的时候如果相同路径下已存在同名主密钥，则不会生成新的密钥覆盖该密钥。如果用户需要生成同名的新的密钥，则需要删除原密钥。

使用 AES 创建主密钥并保存：

```
String KEY_DIR = "D:\\sdk-java\\branch\\sdk-java-
test\\6.5.8\\src\\cn\\ctyun\\sdktest\\keyDir";
SecretKey secretKey = AESUtil.loadSymmetricAESKey(KEY_DIR, "secret_oos.key");
```

请求参数

参数	描述	是否必须
KEY_DIR	主密钥存放的本地位置。	是
secret_oos.key	主密钥的名称。	是

使用 RSA 创建主密钥并保存：

```
String KEY_DIR = "D:\\sdk-java\\branch\\sdk-java-
test\\6.5.8\\src\\cn\\ctyun\\sdktest\\keyDir";
KeyPair keyPair = RSAUtil.loadRSAKeyPair(KEY_DIR, "public_oos.key", "private_oos.key");
```

请求参数

参数	描述	是否必须
KEY_DIR	主密钥存放的本地位置。	是
public_oos.key	公钥名称。加密客户端加密时使用的密钥。	是
private_oos.key	私钥名称。加密客户端解密时使用的密钥。	是

8.4 创建加密客户端

8.4.1 前提条件

客户端加密必须包含 jar 包：Bouncy Castle jar。

OOS JAVA SDK 中已经包含 Bouncy Castle jar，如果原来创建的项目中有 Bouncy Castle jar，使用加密客户端时请确保 Bouncy Castle jar 唯一，不冲突。

8.4.2 加密客户端示例

创建对称加密客户端示例如下：

```
//客户端设置，包括超时时间（单位毫秒）、http/https 协议以及最大连接数等。
ClientConfiguration clientConfiguration= new ClientConfiguration();
        clientConfiguration.setConnectionTimeout(100000);
        clientConfiguration.setSocketTimeout(100000);
        clientConfiguration.setProtocol(Protocol.HTTP);
        clientConfiguration.setMaxConnections(100);
        clientConfiguration.setMaxErrorRetry(0);
//设置 v4 签名（false 为 v2 签名）
System.setProperty(SDKGlobalConfiguration.ENABLE_S3_SIGV4_SYSTEM_PROPERTY, "true");
//设置加密方式为对称加密方式，同时设置主密钥的名称和存放位置
SecretKey secretKey = AESUtil.loadSymmetricAESKey(KEY_DIR, "secret_oos1");
// 初始化加密客户端
AmazonS3EncryptionV2 s3EncryptionV2 = AmazonS3EncryptionClientV2Builder.standard()
                .withEncryptionMaterialsProvider(new
StaticEncryptionMaterialsProvider(new EncryptionMaterials(secretKey)
                        .addDescription("test", "testObject")))
                .withClientConfiguration(clientConfiguration)
                .withCredentialsProvider(new AWSStaticCredentialsProvider(new
BasicAWSCredentials(AK ,SK)))
                .withCryptoConfiguration(new CryptoConfigurationV2()
                        .withCryptoMode(CryptoMode.StrictAuthenticatedEncryption)
//                          .withUnsafeUndecryptableObjectPassthrough(false)
                        .withStorageMode(CryptoStorageMode.ObjectMetadata))
                .withPayloadSigningEnabled(true)
                .withEndpoint(OOS_DOMAIN)
                .build();
```

创建非对称加密客户端示例如下：

```
//客户端设置，包括超时时间（单位毫秒）、http/https 协议以及最大连接数等。
ClientConfiguration clientConfiguration = new ClientConfiguration();
        clientConfiguration.setConnectionTimeout(100000);
        clientConfiguration.setSocketTimeout(100000);
        clientConfiguration.setProtocol(Protocol.HTTP);
        clientConfiguration.setMaxConnections(100);
        clientConfiguration.setMaxErrorRetry(0);
//设置 v4 签名（false 为 v2 签名）
System.setProperty(SDKGlobalConfiguration.ENABLE_S3_SIGV4_SYSTEM_PROPERTY, "true");
//设置加密方式为非对称加密方式，同时设置主密钥的名称和存放位置
KeyPair keyPair = RSAUtil.loadRSAKeyPair(KEY_DIR, "public_oos", "private_oos");
// 初始化加密客户端
AmazonS3EncryptionV2 s3EncryptionV2 = AmazonS3EncryptionClientV2Builder.standard()
                .withEncryptionMaterialsProvider(new
StaticEncryptionMaterialsProvider(new EncryptionMaterials(keyPair)
                        .addDescription("test", "testObject")))
                .withClientConfiguration(clientConfiguration)
                .withCredentialsProvider(new AWSStaticCredentialsProvider(new
BasicAWSCredentials(AK ,SK)))
                .withCryptoConfiguration(new CryptoConfigurationV2()
                        .withCryptoMode(CryptoMode.StrictAuthenticatedEncryption)
//                        .withUnsafeUndecryptableObjectPassthrough(false)
                        .withStorageMode(CryptoStorageMode.ObjectMetadata))
                .withPayloadSigningEnabled(true)
                .withEndpoint(OOS_DOMAIN)
                .build();
```

参数解释：

配置	**模式**	**是否必须**
EncryptionMaterials	主密钥。	是
Description	主密钥描述信息。该信息会存在元数据 x-amz-matdesc 中。	是

ClientConfiguration	加密客户端的配置信息。	是
CredentialsProvider	用户的 AK 和 SK。	是
CryptoConfiguration	加密相关的配置信息。	是
CryptoMode	加密模式。 取值： ● AuthenticatedEncryption：采用 AES/GCM、AESWrap 等认证加密方式。 ● StrictAuthenticatedEncryption：采用 AES/GCM、AESWrap 等认证加密方式，如果发现检索到的文件没有使用认证加密进行保护，将抛出安全异常。 默认值为 StrictAuthenticatedEncryption。 **注意**：由于 NIST 推荐的 AES/GCM 的安全性限制，在这种模式下可以加密的最大消息大小(以字节为单位)的限制是 2^{36}-32，或~64 G。在使用 AES-GCM 进行解密时，在开始使用解密数据之前，需要先读取整个文件。这是为了验证文件在加密后没有被修改。	否
UnsafeUndecryptableObjectPassthrough	如果在下载操作时没有找到加密信息，设置是否在不解密的情况下直接下载文件。 **说明**：CryptoMode 设置为 AuthenticatedEncryption 时，才需要填写此值，否则无效。 取值： ● ture：直接下载。 ● false：不直接下载。 默认值为 false。	否
StorageMode	密钥的存储模式。	否

	取值： ● ObjectMetadata：如果使用 ObjectMetadata 模式，则加密信息将存储在加密文件的元数据中。 **注意**：元数据的大小有 2 KiB 的限制。 ● InstructionFile：如果使用 InstructionFile 模式，则加密信息将存储在单独的指令文件中，该文件将存储在与加密文件相同的 Bucket 中，与加密文件一同上传。此文件中只存储加密相关的元数据信息，其他元数据信息仍存在文件 header 中。 默认值为 ObjectMetadata。	
PayloadSigningEnabled	设置 content-sha256 一致性校验。 取值： ● true：设置 content-sha256 一致性校。 ● false：不设置。 默认值为 false。	否

8.5 PUT Object

使用加密客户端上传 Object 操作，代码示例如下：

```
s3EncryptionV2.putObject(BUCKET_NAME, OBJECT_NAME, FILE);
```

请求参数

参数	描述	是否必须
BUCKET_NAME	Bucket 名称。	是
OBJECT_NAME	上传文件的名称。	是
FILE	上传文件的文件。	是

8.6 Get Object

使用加密客户端进行下载 Object 操作，代码示例如下：

```
s3EncryptionV2.getObject(BUCKET_NAME, OBJECT_NAME);
```

请求参数

参数	描述	是否必须
BUCKET_NAME:	Bucket 名称。	是
OBJECT_NAME	下载文件的名称。	是

8.7 DELETE Object

使用加密客户端进行删除 Object 操作。

注意：使用加密客户端删除文件时，如果存在 InstructionFile 文件则会同步删除。

代码示例如下：

```
s3EncryptionV2.deleteObject(BUCKET_NAME, OBJECT_NAME);
```

请求参数

参数	描述	是否必须
BUCKET_NAME	Bucket 名称。	是
OBJECT_NAME	要删除的文件名称。	是

8.8 Initial Multipart Upload

使用加密客户端进行初始化一个分片上传，代码示例如下：

```
private static String initiateMultipartUpload(AmazonS3EncryptionV2 client, String
objectName) {
       InitiateMultipartUploadRequest request = new
InitiateMultipartUploadRequest(BUCKET_NAME, objectName);
       InitiateMultipartUploadResult res = client.initiateMultipartUpload(request);
       String uploadID = res.getUploadId();
       return uploadID;
    }
```

请求参数

参数	描述	是否必须
BUCKET_NAME	Bucket 名称。	是
OBJECT_NAME	初始化的文件名称。	是

8.9 Upload Part

使用加密客户端进行分片上传。

注意：上传的分片文件，除最后一片外，其余每个分片的大小必须是 16 字节的整数倍。加密客户端初始化分片上传后，上传分片必须使用和初始化时相同的加密设置。因为加密过程需要第 N-1 块的上下文来加密第 N 块，所以用 AmazonS3EncryptionClient 上传的部分必须是连续且有序的。否则，在加密当前部分时，无法使用前面的加密上下文。

代码示例

```
private static String uploadPart(AmazonS3EncryptionV2 client, String uploadId, int
currentPartNumber, long currentPartSize, List<PartETag> parts, long partNum, long
offset) {
        boolean isLastPart = currentPartNumber == partNum;
        UploadPartRequest request = new UploadPartRequest()
                .withUploadId(uploadId)
                .withFile(FILE)
                .withBucketName(BUCKET_NAME)
                .withKey(OBJECT_NAME)
                .withPartNumber(currentPartNumber)
                .withPartSize(currentPartSize)
                .withLastPart(isLastPart);
        if (currentPartNumber != 1) {
            request.setFileOffset(offset);
        }
        UploadPartResult res = client.uploadPart(request);
        String eTag = res.getPartETag().getETag();
        PartETag e = new PartETag(currentPartNumber, eTag);
        parts.add(e);
        return eTag;
    }
```

请求参数

参数	描述	是否必须
client	初始化后的加密客户端。	是

uploadId	初始化分片返回得到的 uploadId。	是
currentPartNumber	上传的是第几片分片。	是
currentPartSize	上传分片的大小。	是
parts	保存所有已上传分片的 eTag 值。	是
partNum	总上传分片个数。	是
offset	上传分片在整体文件的偏置。	是
BUCKET_NAME	Bucket 名称。	是
OBJECT_NAME	文件名称。	是
FILE	上传的文件。	是

8.10 Complete Multipart Upload

使用加密客户端进行合并之前的上传片段完成一次分片上传过程，代码示例如下：

```
private static void completeMultipartUpload(AmazonS3EncryptionV2 client) {
       long partSize = 5 * 1024 * 1024; // 5MiB
       String uploadId = initiateMultipartUpload(client, OBJECT_NAME);
       long partNum = (FILE.length() % partSize) == 0 ? FILE.length() / partSize :
(FILE.length() / partSize) + 1;
       List<PartETag> parts = new ArrayList<>();
       for (int i = 1; i <= partNum; i++) {
           long offset = (i-1) * partSize;
           long currentPartSize = (i == partNum ? FILE.length() - (partSize * (i - 1)) :
partSize);
           uploadPart(client, uploadId, i, currentPartSize, parts, partNum, offset);
       }

       CompleteMultipartUploadRequest request = new
CompleteMultipartUploadRequest(BUCKET_NAME, OBJECT_NAME, uploadId, parts);
       CompleteMultipartUploadResult result = client.completeMultipartUpload(request);
       System.out.println(result.toString());
   }
```

请求参数

参数	描述	是否必须
client	初始化后的加密客户端。	是
uploadId	初始化分片返回得到的 uploadId。	是
parts	保存所有已上传分片的 eTag 值。	是
BUCKET_NAME	Bucket 名称	是
OBJECT_NAME	文件名称。	是

8.11 Abort Multipart Upload

使用加密客户端终止一次分片上传操作，代码示例如下：

```
private static void abortMultipart(AmazonS3EncryptionV2 client, String uploadId) {
        AbortMultipartUploadRequest request = new
AbortMultipartUploadRequest(BUCKET_NAME, OBJECT_NAME, uploadId);
        client.abortMultipartUpload(request);
    }
```

请求参数

参数	描述	是否必须
client	初始化后的加密客户端。	是
uploadId	中止分片的 uploadId。	是
BUCKET_NAME	Bucket 的名称。	是
OBJECT_NAME	中止分片的文件名。	是