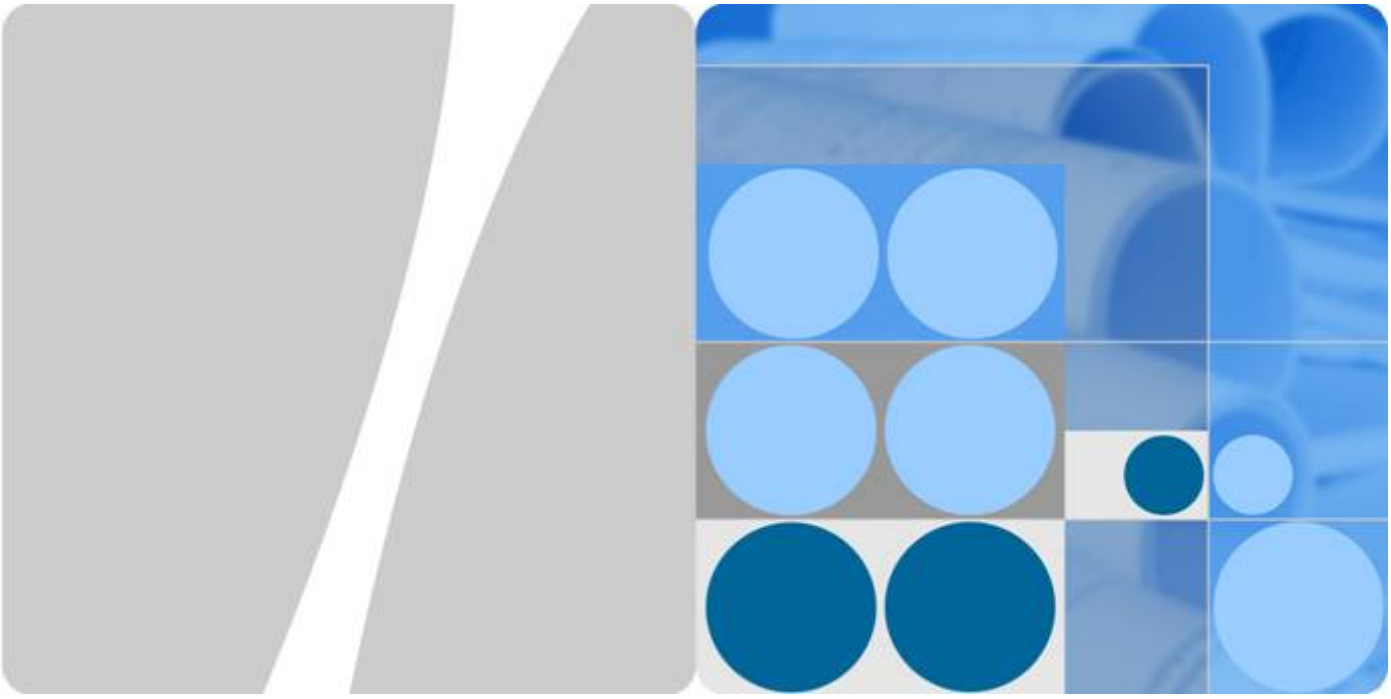

eSDK Storage Plugins

1.1.RC5

开发指南(OBS,Java)

文档版本　01

发布日期　2017-11-30

华为技术有限公司

HUAWEI

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 简介

1.1 OBS SDK 能做什么

OBS 概述

对象存储服务（OBS，Object Storage Service）是一个基于对象的海量存储服务，为客户提供海量、安全、高可靠、低成本的数据存储能力，包括：创建、修改、删除桶，上传、下载、删除对象等。

OBS为用户提供了超大存储容量的能力，适合存放任意类型的文件，适合普通用户、网站、企业和开发者使用。由于OBS是一项面向Internet访问的服务，提供了基于HTTP/HTTPS协议的Web服务接口，用户可以随时随地在任意可以连接至Internet的电脑上，通过OBS管理控制台或客户端访问和管理存储在OBS中的数据。此外，OBS支持Java、Python、PHP、.NET、iOS、C、Android、Ruby、Node.js等多种语言的SDK、以及兼容Amazon S3原生接口的REST API接口，可使用户方便管理自己存储在OBS上的数据，以及开发多种类型的上层业务应用。

云服务实现了在多地域部署基础设施，具备高度的可扩展性和可靠性，用户可根据自身需要指定地域使用OBS服务，由此获得更快的访问速度和实惠的服务价格。

想要了解更多，请访问华为云的**对象存储服务页面**。

SDK 概述

对象存储服务软件开发工具包（OBS SDK，Object Storage Service Software Development Kit）是对OBS服务提供的REST API进行的封装，以简化用户的开发工作。用户直接调用OBS SDK提供的接口函数即可实现使用OBS服务业务能力的目的。

1.2 内容导航

本SDK开发指南指导您如何安装和配置开发环境、如何通过调用OBS SDK提供的接口函数进行二次开发。主要内容如下：

章节	内容
简介	简要介绍OBS的概念和OBS SDK的概念。
相关资源	介绍使用OBS SDK进行二次开发过程中涉及到的资源信息，包括SDK下载地址、OBS服务环境搭建、技术支持渠道等。
安装	介绍OBS SDK的安装方式，并包含示例工程和示例代码。
初始化	介绍使用OBS SDK进行二次开发前需要进行的初始化工作，包括确定服务地址、配置密钥、创建OBS客户端、配置OBS客户端等。
快速入门	介绍使用OBS SDK进行的常用操作，包括初始化OBS客户端、创建桶、上传对象、下载对象、列举对象和删除对象。如果您想快速使用OBS SDK进行开发，可以先阅读这一章节。
管理桶	详细介绍如何使用OBS SDK管理桶，包括创建桶、列举桶、删除桶、获取桶元数据、管理桶访问权限、获取桶区域位置等。
上传对象	详细介绍使用OBS SDK上传对象的多种方式，包括流式上传、文件上传、分段上传、断点续传上传、基于表单上传等。
下载对象	详细介绍使用OBS SDK下载对象的多种方式，包括流式下载、范围下载、限定条件下载、断点续传下载等。
管理对象	详细介绍如何使用OBS SDK管理对象，包括获取对象属性、管理对象访问权限、列举对象、删除对象、复制对象等。
临时授权访问	详细介绍如何使用OBS SDK进行临时授权访问，包括V2临时授权访问、V4临时授权访问。
多版本控制	详细介绍如何使用OBS SDK进行多版本控制，包括设置桶多版本状态、查看桶多版本状态、获取多版本对象、删除多版本对象、列举多版本对象等。
生命周期管理	详细介绍如何使用OBS SDK进行桶的生命周期管理，包括设置生命周期规则、查看生命周期规则、删除生命周期规则。

章节	内容
跨域资源共享	详细介绍如何使用OBS SDK配置桶的跨域资源共享，包括设置跨域规则、查看跨域规则、删除跨域规则。
设置访问日志	详细介绍如何使用OBS SDK设置桶的访问日志，包括开启桶日志、查看桶日志、关闭桶日志。
静态网站托管	详细介绍如何使用OBS SDK进行静态网站托管，包括网站文件托管、设置托管配置、查看托管配置、清除托管配置。
标签管理	详细介绍如何使用OBS SDK进行桶的标签管理，包括设置桶标签、查看桶标签、删除桶标签。
事件通知	详细介绍如何使用OBS SDK进行配置桶的事件通知，包括设置事件通知、查看事件通知、删除事件通知。
服务端加密	详细介绍如何使用OBS SDK进行服务端加密。
异常处理	介绍使用OBS SDK过程中遇到异常时的解决办法。
常见问题	解答使用OBS SDK过程中的常见问题。

2 相关资源

2.1 SDK 下载

- OBS Java SDK最新版本 2.1.17：eSDK_Storage_OBS_V2.1.17_Java.zip。
- API文档地址：**API Doc**。

2.2 兼容性

- 支持的JDK版本：JDK 1.7.0_51及以上版本。
- 命名空间：移除旧版本（1.5.x）对org.jets3t.service.*的相关依赖，重写相关代码将依赖调整到包com.obs.services.internal.*中。
- 接口函数：与旧版本（1.5.x）保持兼容。

2.3 OBS 服务环境搭建

步骤1 注册云服务帐号

使用OBS之前必须要有一个云服务帐号。

1. 打开浏览器。
2. 登录公有云网站**www.huaweicloud.com**。
3. 在页面右上角单击**“注册”**。
4. 按需填写注册信息并单击“同意协议并注册”。

步骤2 开通OBS服务

使用OBS服务之前必须先充值，才能正常使用OBS服务。

1. 登录OBS管理控制台。
2. 单击页面右上角的用户名并选择“用户中心”进入用户中心页面。
3. 单击“充值”，系统自动跳转到充值窗口。
4. 根据界面提示信息，对账户进行充值。
5. 充值成功后，关闭充值窗口，返回管理控制台首页。
6. 单击“对象存储服务”，开通并进入OBS管理控制台。

步骤3 创建访问密钥

OBS通过用户帐户中的AK和SK进行签名验证，确保通过授权的帐户才能访问指定的OBS资源。以下是对AK和SK的解释说明：

- AK：Access Key ID，接入键标识，用户在对象存储服务系统中的接入键标识，一个接入键标识唯一对应一个用户，一个用户可以同时拥有多个接入键标识。对象存储服务系统通过接入键标识识别访问系统的用户。
- SK：Secret Access Key，安全接入键，用户在对象存储服务系统中的安全接入键，是用户访问对象存储服务系统的密钥，用户根据安全接入键和请求头域生成鉴权信息。安全接入键和接入键标识一一对应。

创建访问密钥的操作步骤如下：

1. 登录OBS控制台。
2. 单击页面右上角的用户名，并选择“我的凭证”。
3. 进入“我的凭证”页面，单击“管理访问密钥”页签下方的“新增访问密钥”。
4. 在弹出的“新增访问密钥”对话框中，输入登录密码和对应验证码。

   说明

   - 用户如果未绑定邮箱和手机，则只需输入登录密码。
   - 用户如果同时绑定了邮箱和手机，可以选择其中一种方式进行验证。
5. 单击“确定”。
6. 在弹出的“下载确认”提示框中，单击“确定”后，密钥会直接保存到浏览器默认的下载文件夹中。
7. 打开下载下来的“credentials.csv”文件既可获取到访问密钥（AK和SK）。

说明

- 每个用户最多可创建两个有效的访问密钥。
- 为防止访问密钥泄露，建议您将其保存到安全的位置。如果用户在此提示框中单击“取消”，则不会下载密钥，后续也将无法重新下载。如果需要使用访问密钥，可以重新创建新的访问密钥。

----结束

2.4 技术支持渠道

开发者社区提供的技术支持渠道如下：

- 开发过程中，您有任何问题可以在**华为开发者论坛**中发帖求助。
- 开发过程中，您有任何问题可以在**DevCenter**系统中提单跟踪。
- 如果您在使用过程中有任何疑问，都可以通过以下方式联系我们。

a. 技术支持热线电话：400-8828-000
b. 技术支持邮箱：**esdk@huawei.com**

3 安装

3.1 环境准备

- 从**Oracle官网**下载并安装JDK1.7.0_51以上版本。
- 从**Eclipse官网**下载并安装Eclipse IDE for Java Developers最新版本。

3.2 安装方式

在Eclipse Java项目中导入JAR包，步骤如下：

步骤1 下载OBS Java SDK开发包。

步骤2 解压该开发包。

步骤3 将解压后的文件：esdk-obs-java-<versionId>.jar 以及third_party文件夹下所有的JAR包拷贝到您的项目中。

步骤4 在Eclipse中选择您的工程，右击选择 Properties > Java Build Path > Add JARs。

步骤5 选中您在第3步拷贝的所有JAR文件，单击“确定”，完成JAR包的导入。

----结束

3.3 示例工程

OBS Java SDK提供了基于Eclipse的示例工程，您可以在本地设备上编译运行示例工程。您也可以以示例工程为基础开发您的应用。

- 示例工程代码：huawei-obs-java-sdk-demo.zip。

说明

- 编译运行前，请修改HelloOBS.java中endPoint，ak，sk，bucketName为您的真实信息。
- 工程的编译运行方法，参看工程目录下README.md。

3.4 示例程序

OBS Java SDK提供了丰富的示例程序，方便用户参考或直接使用。您可以从OBS Java SDK开发包中获取示例程序，如esdk-obs-java-<versionId>.zip，解压后esdk-obs-java-<versionId>/samples为示例程序。您也可以从下面表格中直接下载示例程序。

示例包括以下内容：

示例代码	说明
BucketOperationsSample.zip	展示了桶相关接口的用法
ObjectOperationsSample.zip	展示了对象相关接口的用法
DownloadSample.zip	展示了下载对象的用法
CreateFolderSample.zip	展示了创建文件夹的用法
DeleteObjectsSample.zip	展示了批量删除对象的用法
ListObjectsSample.zip	展示了列举对象的用法
ListVersionsSample.zip	展示了列举多版本对象的用法
ListObjectsInFolderSample.zip	展示了列举文件夹内对象的用法
ObjectMetaSample.zip	展示了自定义对象元数据的用法
SimpleMultipartUploadSample.zip	展示了分段上传的基本用法
RestoreObjectSample.zip	展示了下载归档存储对象的用法
ConcurrentCopyPartSample.zip	展示了分段并发复制大对象的用法
ConcurrentDownloadObjectSample.zip	展示了分段并发下载大对象的用法
ConcurrentUploadPartSample.zip	展示了分段并发上传大对象的用法
PostObjectSample.zip	展示了表单上传对象的用法
V2TemporarySignatureSample.zip	展示了临时授权访问的用法
V4TemporarySignatureSample.zip	展示了V4临时授权访问的用法

4 初始化

OBS客户端（ObsClient）是访问OBS服务的Java客户端，它为调用者提供一系列与OBS服务进行交互的接口，用于管理、操作桶（Bucket）和对象（Object）等OBS服务上的资源。使用OBS Java SDK发起OBS请求，您需要初始化一个ObsClient实例，并根据需要修改ObsConfiguration的默认配置项。

4.1 服务地址

- 您可以从**这里**查看OBS当前开通的服务地址和区域信息。
- 服务地址可以是域名，也可以是IP。
- OBS服务按对桶的访问类型可分为路径访问方式和虚拟主机访问方式，具体示例如下：

示例	说明
http://obs.cn-north-1.myhwclouds.com/bucket001	通过HTTP协议，以路径访问方式，访问华北区1的桶bucket001
https://obs.cn-north-1.myhwclouds.com/bucket001	通过HTTPS协议，以路径访问方式，访问华北区1的桶bucket001
http://bucket001.obs.cn-south-1.myhwclouds.com	通过HTTP协议，以虚拟主机访问方式，访问华南区1的桶bucket001
https://bucket001.obs.cn-south-1.myhwclouds.com	通过HTTPS协议，以虚拟主机访问方式，访问华南区1的桶bucket001

示例	说明
http://10.10.10.10/bucket001	通过HTTP协议，以路径访问方式，访问IP为10.10.10.10的桶bucket001
https://10.10.10.10/bucket001	通过HTTPS协议，以路径访问方式，访问IP为10.10.10.10的桶bucket001

说明

- 当服务地址为IP时，只支持路径访问方式。
- 推荐使用虚拟主机访问方式，该方式下可减少HTTP请求转发，提高性能。
- SDK通过OBS客户端配置参数实现对虚拟主机访问方式和路径访问方式的支持，用户初始化OBS客户端配置服务地址时，使用实际的域名即可。

4.2 配置密钥

要接入OBS服务，您需要拥有一组有效的访问密钥（AK和SK）用来进行签名认证。具体可参考**OBS服务环境搭建**。

获取AK和SK之后，您便可以按照以下步骤进行初始化。

4.3 创建 OBS 客户端

直接使用服务地址创建OBS客户端（ObsClient）：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 使用访问OBS

// 关闭obsClient
obsClient.close();
```

使用配置类（ObsConfiguration）创建OBS客户端（ObsClient）：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsConfiguration配置类实例
ObsConfiguration config = new ObsConfiguration();
config.setEndPoint(endPoint);
config.setDisableDnsBucket(false);
config.setHttpsOnly(true);

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, config);

// 使用访问OBS

// 关闭obsClient
obsClient.close();
```

说明

- 您的工程中可以有多个ObsClient，也可以只有一个ObsClient。
- ObsClient是线程安全的，可以并发使用。
- ObsClient在调用ObsClient.close方法关闭后不能再次使用。
- ObsClient支持路径访问和虚拟主机访问两种请求方式，您可通过ObsConfiguration.setDisableDnsBucket进行配置。

4.4 配置 OBS 客户端

您可通过ObsConfiguration配置类对ObsClient进行配置，可配置代理、连接超时、最大连接数等参数。通过ObsConfiguration可以设置的参数见下表：

参数	描述	方法	建议值
connectionTimeout	建立HTTP/HTTPS连接的超时时间（单位：毫秒）。默认为60000毫秒。	ObsConfiguration.setConnectionTimeout	[10000, 60000]
connectionRequestTimeout	从连接池中获取连接的超时时间（单位：毫秒）。默认不超时。	ObsConfiguration.setConnectionRequestTimeout	默认
socketTimeout	Socket层传输数据的超时时间（单位：毫秒）。默认为60000毫秒。	ObsConfiguration.setSocketTimeout	[10000, 60000]
idleConnectionTime	如果空闲时间超过此参数的设定值，则关闭连接（单位：毫秒）。默认为60000毫秒。	ObsConfiguration.setIdleConnectionTime	默认
maxConnections	允许打开的最大HTTP连接数。默认为1000。	ObsConfiguration.setMaxConnections	默认
maxErrorRetry	请求失败（请求异常、服务端报500或503错误）后最大的重试次数。默认3次。	ObsConfiguration.setMaxErrorRetry	[1, 5]
endPoint	连接OBS的服务地址。默认obs.myhwclouds.com。	ObsConfiguration.setEndPoint	N/A
endpointHttpPort	HTTP请求的端口号。默认为80。	ObsConfiguration.setEndpointHttpPort	N/A

参数	描述	方法	建议值
endpointHttpsPort	HTTPS请求的端口号。默认为443。	ObsConfiguration.setEndpointHttpsPort	N/A
httpsOnly	是否使用HTTPS连接OBS服务。默认为true。	ObsConfiguration.setHttpsOnly	默认
disableDnsBucket	是否禁用虚拟主机方式访问OBS服务。默认为false。	ObsConfiguration.setDisableDnsBucket	默认（如果服务地址是IP则必须设置为true。）
signatString	连接OBS使用的鉴权方式，支持v2和v4两种鉴权方式。默认为v4。	ObsConfiguration.setSignatString	默认
httpProxy	HTTP代理配置。默认为空。	ObsConfiguration.setHttpProxy	N/A
validateCertificate	是否验证服务端证书。默认为false。	ObsConfiguration.setValidateCertificate	N/A
verifyResponseContentType	是否验证响应头信息的ContentType。默认为true。	ObsConfiguration.setVerifyResponseContentType	默认
uploadStreamRetryBufferSize	上传流对象时使用的缓存大小（单位：字节）。默认为1048576字节（1M）。	ObsConfiguration.setUploadStreamRetryBufferSize	N/A
isNio	是否启用NIO模式。默认为false。	ObsConfiguration.enableNio, ObsConfiguration.disableNio	N/A
socketWriteBufferSize	Socket发送缓冲区大小（单位：字节），对应java.net.SocketOptions.SO_SNDBUF参数。默认为-1表示不设置。	ObsConfiguration.setSocketWriteBufferSize	默认

参数	描述	方法	建议值
sockeReadBufferSize	Socket接收缓冲区大小（单位：字节），对应java.net.SocketOptions.SO_RCVBUF参数。默认为-1表示不设置。	ObsConfiguration.setSockeReadBufferSize	默认
readBufferSize	上传对象到Socket流时的读缓存大小（单位：字节）。默认为8192字节。	ObsConfiguration.setReadBufferSize	默认
writeBufferSize	上传对象到Socket流时的写缓存大小（单位：字节）。默认为8192字节。	ObsConfiguration.setWriteBufferSize	默认
defaultBucketLocation	默认的区域位置。默认为CHINA。	ObsConfiguration.setDefaultBucketLocation	N/A

说明

- 建议值为N/A的表示需要根据实际情况进行设置。
- 如需提高并发性能，建议启用NIO模式。
- 如需提高大文件上传下载性能，在网络带宽满足的情况下，可对socketWriteBufferSize，sockeReadBufferSize，readBufferSize，writeBufferSize四个参数进行调优。
- 如网络状况不佳，建议增大connectionTimeout和socketTimeout的值。

4.5 配置 SDK 日志

OBS Java SDK基于Apache Log4j2开源库提供了日志功能，您可以通过加入日志配置文件开启日志功能。具体步骤如下：

步骤1 找到OBS Java SDK开发包中的log4j2.xml文件，或者直接下载log4j2.xml。

步骤2 将log4j2.xml文件放到classpath根目录。

步骤3 根据实际情况修改log4j2.xml中的日志级别。

----结束

说明

- 如果不加入日志配置文件，则视为关闭日志功能，不会有日志输出。
- 您可以从**日志分析**章节获取更多关于SDK日志的信息。
- 日志文件默认存放于JDK系统变量user.dir所代表的路径下，可在log4j2.xml中进行修改。

4.6 配置 SDK 验证服务端证书

OBS Java SDK提供了对服务端证书验证的支持，以确保OBS服务来自于受信服务端。以Windows操作系统为例（Linux操作系统下请将%JAVA_HOME%修改为$JAVA_HOME），配置验证服务端证书的步骤如下：

说明

- 如果访问的OBS服务端的根证书是由权威机构颁发的，可以忽略步骤1,2,3（JDK的证书库中默认已加入了权威机构的根证书）。

步骤1 获取OBS服务端的根证书，并保存为文件obs.cer。

步骤2 运行命令**%JAVA_HOME%/bin/keytool -import -alias obs -file obs.cer -storepass changeit -keystore %JAVA_HOME%/jre/lib/security/cacerts**导入证书。

步骤3 运行命令**%JAVA_HOME%/bin/keytool -list -v -alias obs -storepass changeit -keystore %JAVA_HOME%/jre/lib/security/cacerts**查看证书是否导入成功。

步骤4 配置OBS客户端使用HTTPS协议ObsConfiguration.setHttpsOnly(true)，并开启服务端证书验证ObsConfiguration.setValidateCertificate(true)。

----结束

4.7 透传访问密钥

OBS Java SDK提供了直接在接口方法中透传AK和SK的OBS客户端（SecretFlexibleObsClient）。示例代码如下：

```
String endPoint = "obs.myhwclouds.com";
// 创建ObsConfiguration配置类实例
ObsConfiguration config = new ObsConfiguration();
config.setEndPoint(endPoint);
config.setDisableDnsBucket(false);
config.setHttpsOnly(true);

// 创建SecretFlexibleObsClient实例
SecretFlexibleObsClient obsClient = new SecretFlexibleObsClient(config);
// 使用访问OBS
String ak1 = "*** Provide your Access Key 1 ***";
String sk1 = "*** Provide your Secret Key 1 ***";
obsClient.listBuckets(ak1, sk1);

String ak2 = "*** Provide your Access Key 2 ***";
String sk2 = "*** Provide your Secret Key 2 ***";
obsClient.listBuckets(ak2, sk2);

// 关闭obsClient
obsClient.close();
```

说明

SecretFlexibleObsClient 继承自ObsClient，可作为ObsClient使用。

5 快速入门

请确认您已经熟悉OBS的基本概念，如**桶（Bucket）**、**对象（Object）**、**区域**、**访问密钥（AK和SK）**等。

本节您将看到如何快速使用OBS Java SDK，完成常见操作，如创建桶、上传对象、下载对象等。

注意

- 使用OBS客户端进行接口调用操作完成后，没有异常抛出，则表明返回值有效；若抛出异常，则说明操作失败，此时应从**SDK自定义异常**实例中获取错误信息。
- 使用OBS客户端进行接口调用成功后，均会返回包含响应头信息的**SDK公共响应头**实例（或其子类实例）。

5.1 初始化 OBS 客户端

向OBS发送任一HTTP/HTTPS请求之前，必须先创建一个ObsClient实例：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 使用访问OBS

// 关闭obsClient
obsClient.close();
```

说明

更多OBS客户端初始化的内容请参考**初始化**。

5.2 创建桶

桶是OBS全局命名空间，相当于数据的容器、文件系统的根目录，可以存储若干对象。以下代码展示如何新建一个桶：

```
obsClient.createBucket("bucketname");
```

说明

- 桶的名字是全局唯一的，所以您需要确保不与已有的桶名称重复。
- 桶命名规则如下：
  - 3～63个字符，数字或字母开头，支持小写字母、数字、“-”、“.”。
  - 禁止使用IP地址。
  - 禁止以“-”或“.”开头及结尾。
  - 禁止两个“.”相邻（如：“my..bucket”）。
  - 禁止“.”和“-”相邻（如：“my-.bucket”和“my.-bucket”）。
- 更多创建桶的信息，请参见**管理桶**。

5.3 上传对象

以下代码展示如何上传对象至OBS：

```
obsClient.putObject("bucketname", "objectkey", new ByteArrayInputStream("Hello OBS".getBytes()));
```

说明

更多上传对象的信息，请参见**上传对象**。

5.4 下载对象

以下代码展示如何获取对象的内容：

```
S3Object s3Object = obsClient.getObject("bucketname", "objectkey");
InputStream content = s3Object.getObjectContent();
if (content != null)
{
    BufferedReader reader = new BufferedReader(new InputStreamReader(content));
    while (true)
    {
        String line = reader.readLine();
        if (line == null)
            break;
        System.out.println("\n" + line);
    }
    reader.close();
}
```

说明

- 调用ObsClient.getObject返回一个S3Object实例，该实例包含对象内容及其属性。
- 调用S3Object.getObjectContent获取对象输入流，可读取此输入流获取其内容，用完之后请关闭这个流。
- 更多下载对象的信息，请参见**下载对象**。

5.5 列举对象

当完成一系列上传对象操作后，可能需要查看桶下包含哪些对象。以下代码展示如何列举指定桶下的对象：

```
ObjectListing objectListing = obsClient.listObjects("bucketname");
for(S3Object s3Object : objectListing.getObjectSummaries()){
    System.out.println(" - " + s3Object.getObjectKey() + "  " +  "(size = " +
s3Object.getMetadata().getContentLength() + ")");
}
```

说明

- 调用ObsClient.listObjects返回ObjectListing实例，该实例包含此次listObject请求的返回结果，可通过ObjetListing.getObjectSummaries获取所有对象（Object）的描述信息。
- 上面的代码默认列举1000个对象（Object）。
- 更丰富的列举功能，请参见**列举对象**。

5.6 删除对象

以下代码展示如何删除指定的对象：

```
obsClient.deleteObject("bucketname", "objectkey");
```

6 管理桶

6.1 创建桶

您可以通过ObsClient.createBucket创建桶。

简单创建

以下代码展示如何新建一个桶：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// 创建桶
obsClient.createBucket("bucketname");
// 关闭obsClient
obsClient.close();
```

说明

- 桶的名字是全局唯一的，所以您需要确保不与已有的桶名称重复。
- 桶命名规则如下：
  - 3～63个字符，数字或字母开头，支持小写字母、数字、“-”、“.”。
  - 禁止使用IP地址。
  - 禁止以“-”或“.”开头及结尾。
  - 禁止两个“.”相邻（如：“my..bucket”）。
  - 禁止“.”和“-”相邻（如：“my-.bucket”和“my.-bucket”）。
- 同一用户多次创建同名桶不会报错，创建的桶属性以第一次请求为准。
- 桶的**访问权限**默认是私有读写，存储类型默认是标准类型，区域位置默认是服务地址所在的默认区域。

带参数创建

上面代码创建的桶，桶访问权限是私有读写，存储类型是标准存储（STANDARD），区域位置是服务地址所在的默认区域。OBS支持的桶的存储类型有三类，参见**桶存储类型**。创建桶时可以指定桶的访问权限、存储类型和区域位置，示例代码如下：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

S3Bucket s3Bucket = new S3Bucket();
s3Bucket.setBucketName("bucketname");
// 设置桶访问权限为公共读，默认是私有读写
s3Bucket.setAcl(AccessControlList.REST_CANNED_PUBLIC_READ);
// 设置桶的存储类型为归档类型，默认是标准存储类型，支持STANDARD（标准存储类型），STANDARD_IA（低频访问存储类型），GLACIER（归档存储类型）
s3Bucket.setStorageClass(S3Bucket.GLACIER);
// 设置桶区域位置
s3Bucket.setLocation("bucketlocation");
// 创建桶
obsClient.createBucket(s3Bucket);

// 关闭obsClient
obsClient.close();
```

6.2 列举桶

您可以通过ObsClient.listBuckets列举桶。以下代码展示如何获取桶列表：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 列举桶
List<S3Bucket> buckets = obsClient.listBuckets();
for(S3Bucket bucket : buckets){
    System.out.println(" - " + bucket.getBucketName());
    System.out.println(" - " + bucket.getCreationDate());
}
// 关闭obsClient
obsClient.close();
```

说明

桶列表按照桶名字典顺序排列。

6.3 删除桶

您可以通过ObsClient.deleteBucket删除桶。以下代码展示如何删除一个桶：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 删除桶
obsClient.deleteBucket("bucketname");

// 关闭obsClient
obsClient.close();
```

说明

- 如果桶不为空（包含对象或分段上传碎片），则该桶无法删除。
- 删除桶非幂等操作，删除不存在的桶会报错。

6.4 判断桶是否存在

您可以通过ObsClient.headBucket接口判断该桶是否已存在。以下代码展示如何判断指定桶是否存在：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

boolean exists = obsClient.headBucket("bucketname");

// 关闭obsClient
obsClient.close();
```

6.5 获取桶元数据

您可以通过ObsClient.getBucketMetadata接口获取桶元数据。以下代码展示如何获取桶元数据：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketMetadataInfoRequest request = new BucketMetadataInfoRequest("bucketname");
request.setOrigin("http://www.a.com");
// 获取桶元数据
BucketMetadataInfoResult result = obsClient.getBucketMetadata(request);
System.out.println("\t:" + result.getDefaultStorageClass());
System.out.println("\t:" + result.getAllowOrigin());
System.out.println("\t:" + result.getMaxAge());
System.out.println("\t:" + result.getAllowHeaders());
System.out.println("\t:" + result.getAllowMethods());
System.out.println("\t:" + result.getExposeHeaders());
// 关闭obsClient
obsClient.close();
```

说明

BucketMetadataInfoResult.getAllowMethods等方法的值参见桶的**跨域资源共享**（CORS）配置。

6.6 管理桶访问权限

桶访问权限（**ACL**）可以通过三种方式设置：

1. 创建桶时指定预定义访问策略。
2. 调用ObsClient.setBucketAcl指定预定义访问策略。
3. 调用ObsClient.setBucketAcl直接设置。

OBS支持的桶或对象权限包含五类，见下表：

权限	描述	OBS Java SDK对应值
读权限	若有桶的读权限，则可以获取该桶内对象列表和桶的元数据。 若有对象的读权限，则可以获取该对象内容和元数据。	Permission.PERMISSION_READ
写权限	若有桶的写权限，则可以上传、覆盖和删除该桶内任何对象。 此权限在对象上不适用。	Permission.PERMISSION_WRITE
读ACP权限	若有读ACP的权限，则可以获取对应的桶或对象的权限控制列表（ACL）。 桶或对象的所有者永远拥有读对应桶或对象ACP的权限。	Permission.PERMISSION_READ_ACP
写ACP权限	若有写ACP的权限，则可以更新对应桶或对象的权限控制列表（ACL）。 桶或对象的所有者永远拥有写对应桶或对象的ACP的权限。 拥有了写ACP的权限，由于可以更改权限控制策略，实际上意味着拥有了完全访问的权限。	Permission.PERMISSION_WRITE_ACP

权限	描述	OBS Java SDK对应值
完全控制权限	若有桶的完全控制权限意味着拥有READ、WRITE、READ_ACP和WRITE_ACP的权限。 若有对象的完全控制权限意味着拥有READ、READ_ACP和WRITE_ACP的权限。	Permission.PERMISSION_FULL_CONTROL

OBS预定义的访问策略包含七类，见下表：

权限	描述	OBS Java SDK对应值
私有读写	桶或对象的所有者拥有完全控制的权限，其他任何人都没有访问权限。	AccessControlList.REST_CANNED_PRIVATE
公共读私有写	桶或对象的所有者拥有完全控制的权限，其他所有用户包括匿名用户拥有读的权限。	AccessControlList.REST_CANNED_PUBLIC_READ
公共读写	桶或对象的所有者拥有完全控制的权限，其他所有用户包括匿名用户拥有读和写的权限。	AccessControlList.REST_CANNED_PUBLIC_READ_WRITE
授权用户读私有写	桶或对象的所有者拥有完全控制的权限，其他OBS授权用户拥有读权限。	AccessControlList.REST_CANNED_AUTHENTICATED_READ
桶所有者读对象所有者读写	对象的所有者拥有完全控制的权限，桶的所有者拥有只读的权限。	AccessControlList.REST_CANNED_BUCKET_OWNER_READ
桶所有者读写对象所有者读写	对象的所有者拥有完全控制的权限，桶的所有者拥有完全控制的权限。	AccessControlList.REST_CANNED_BUCKET_OWNER_FULL_CONTROL
日志投递组写	日志投递用户组拥有对桶的写权限以及读ACP的权限。	AccessControlList.REST_CANNED_LOG_DELIVERY_WRITE

创桶时指定预定义访问策略

以下代码展示如何在创建桶时指定预定义访问策略：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
```

```
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

S3Bucket s3Bucket = new S3Bucket();
s3Bucket.setBucketName("bucketname");
// 设置桶访问权限为公共读写
s3Bucket.setAcl(AccessControlList.REST_CANNED_PUBLIC_READ_WRITE);
// 创建桶
obsClient.createBucket("bucketname");

// 关闭obsClient
obsClient.close();
```

为桶设置预定义访问策略

以下代码展示如何为桶设置预定义访问策略：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 设置桶访问权限为日志投递组写
obsClient.setBucketAcl("bucketname", AccessControlList.REST_CANNED_LOG_DELIVERY_WRITE);

// 关闭obsClient
obsClient.close();
```

直接设置桶访问权限

以下代码展示如何直接设置桶访问权限：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
owner.setId("ownerid");
acl.setOwner(owner);
// 为所有用户设置读权限
acl.grantPermission(GroupGrantee.ALL_USERS, Permission.PERMISSION_READ);
// 为授权用户设置写权限
acl.grantPermission(GroupGrantee.AUTHENTICATED_USERS, Permission.PERMISSION_WRITE);
// 为日志投递组设置写权限和读ACP权限
acl.grantPermission(GroupGrantee.LOG_DELIVERY, Permission.PERMISSION_WRITE);
acl.grantPermission(GroupGrantee.LOG_DELIVERY, Permission.PERMISSION_READ_ACP);
// 直接设置桶访问权限
obsClient.setBucketAcl("bucketname", acl);

// 关闭obsClient
obsClient.close();
```

说明

ACL中需要填写的所有者（Owner）或者被授权用户（Grantee）的ID，是指用户的帐户ID，可通过OBS控制台“我的凭证”页面查看。

获取桶访问权限

您可以通过ObsClient.getBucketAcl获取桶的访问权限。以下代码展示如何获取桶访问权限：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
```

```
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AccessControlList acl = obsClient.getBucketAcl("bucketname");
System.out.println(acl);

// 关闭obsClient
obsClient.close();
```

6.7 获取桶区域位置

您可以通过ObsClient.getBucketLocation获取桶的区域位置。以下代码展示如何获取桶区域位置：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

String location = obsClient.getBucketLocation("bucketname");
System.out.println("\t:" + location);
// 关闭obsClient
obsClient.close();
```

说明

创建桶时可以指定桶的区域位置，请参见**创建桶**。

6.8 获取桶存量信息

桶存量信息包括桶已使用的空间大小以及桶包含的对象个数。您可以通过ObsClient.getBucketStorageInfo获取桶的存量信息。以下代码展示如何获取桶存量信息：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketStorageInfo storageInfo = obsClient.getBucketStorageInfo("bucketname");
System.out.println("\t" + storageInfo.getObjectNumber());
System.out.println("\t" + storageInfo.getSize());

// 关闭obsClient
obsClient.close();
```

6.9 桶配额

设置桶配额

您可以通过ObsClient.setBucketQuota设置桶配额。以下代码展示如何设置桶配额：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
```

```
// 设置桶配额为100MB
BucketQuota quota = new BucketQuota(1024 * 1024 * 100);
obsClient.setBucketQuota("bucketname", quota);

// 关闭obsClient
obsClient.close();
```

说明

桶配额值必须为非负整数，单位为字节，支持的最大值为2^{63} - 1。

获取桶配额

您可以通过ObsClient.getBucketQuota获取桶配额。以下代码展示如何获取桶配额：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketQuota quota = obsClient.getBucketQuota("bucketname");
System.out.println("\t" + quota.getBucketQuota());

// 关闭obsClient
obsClient.close();
```

6.10 桶存储类型

OBS允许您对桶配置不同的存储类型，桶中对象的存储类型默认将与桶的存储类型保持一致。不同的存储类型可以满足客户业务对存储性能、成本的不同诉求。桶的存储类型分为三类，见下表：

类型	说明	OBS Java SDK对应值
标准存储	标准存储拥有低访问时延和较高的吞吐量，适用于有大量热点对象（平均一个月多次）或小对象（<1MB），且需要频繁访问数据的业务场景。	S3Bucket.STANDARD
低频访问存储	低频访问存储适用于不频繁访问（平均一年少于12次）但在需要时也要求能够快速访问数据的业务场景。	S3Bucket.STANDARD_IA
归档存储	归档存储适用于很少访问（平均一年访问一次）数据的业务场景。	S3Bucket.GLACIER

更多关于桶存储类型的内容请参考**桶的存储类别**。

设置桶存储类型

您可以通过ObsClient.setBucketStoragePolicy设置桶存储类型。以下代码展示如何设置桶存储类型：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 设置桶的存储类型为低频访问存储
BucketStoragePolicyConfiguration storgePolicy = new BucketStoragePolicyConfiguration();
storgePolicy.setStorageClass(S3Bucket.STANDARD_IA);
obsClient.setBucketStoragePolicy("bucketname", storgePolicy);

// 关闭obsClient
obsClient.close();
```

获取桶存储类型

您可以通过ObsClient.getBucketStoragePolicy获取桶存储类型。以下代码展示如何获取桶存储类型：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketStoragePolicyConfiguration storagePolicy = obsClient.getBucketStoragePolicy("bucketname");
System.out.println("\t" + storagePolicy.getStorageClass());

// 关闭obsClient
obsClient.close();
```

7 上传对象

在OBS中，用户操作的基本数据单元是对象。OBS Java SDK提供了丰富的对象上传接口，可以通过以下方式上传对象：

- 流式上传
- 文件上传
- 分段上传
- 断点续传上传
- 基于表单上传

流式上传和文件上传的内容大小不能超过5GB；当上传较大文件时，请使用分段上传，多段上传每段内容大小不能超过5GB；表单上传提供了基于浏览器表单上传对象的方式。

7.1 流式上传

流式上传使用InputStream作为对象的数据源。您可以通过ObsClient.putObject上传您的数据流到OBS。以下代码展示了如何进行流式上传：

上传字符串（byte 数组）

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
```

```
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

String content = "Hello OBS";
obsClient.putObject("bucketname", "objectkey", new ByteArrayInputStream(content.getBytes()));

// 关闭obsClient
obsClient.close();
```

上传网络流

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

InputStream inputStream = new URL("http://www.huawei.com").openStream();
obsClient.putObject("bucketname", "objectkey", inputStream);

// 关闭obsClient
obsClient.close();
```

上传文件流

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

FileInputStream fis = new FileInputStream(new File("localfile"));
obsClient.putObject("bucketname", "objectkey", fis);

// 关闭obsClient
obsClient.close();
```

注意

- 推荐使用**文件上传**的形式上传本地文件，而不是文件流形式。
- 大文件上传建议使用**分段上传**。

7.2 文件上传

文件上传使用本地文件作为对象的数据源。以下代码展示了如何进行文件上传：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.putObject("bucketname", "objectkey", new File("localfile"));

// 关闭obsClient
obsClient.close();
```

7.3 创建文件夹

OBS本身是没有文件夹的概念的，桶中存储的元素只有对象。创建文件夹实际上是创建了一个大小为0且对象名以“/”结尾的对象，这类对象与其他对象无任何差异，可以进行下载、删除等操作，只是OBS控制台会将这类以“/”结尾的对象以文件夹的方式展示。

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

final String keySuffixWithSlash = "parent_directory/";
obsClient.putObject("bucketname", keySuffixWithSlash, new ByteArrayInputStream(new byte[0]));

// 在文件夹下创建对象
obsClient.putObject("bucketname", keySuffixWithSlash + "objectkey", new
ByteArrayInputStream("Hello OBS".getBytes()));

// 关闭obsClient
obsClient.close();
```

说明

- 创建文件夹本质上来说是创建了一个大小为0且对象名以“/”结尾的对象。
- 多级文件夹创建最后一级即可，比如src1/src2/src3/，创建src1/src2/src3/即可，无需创建src1/、src1/src2/。

7.4 设置对象属性

您可以在上传对象时设置对象属性。对象属性包含对象长度、对象MIME类型、对象MD5值（用于校验）、对象存储类型、对象自定义元数据。其中，对象自定义元数据是用户对上传到桶中对象的自定义属性描述，以便对对象进行自定义管理。对象属性可以在多种上传方式下（流式上传、文件上传、分段上传、基于表单上传），或**复制对象**时进行设置。

对象属性详细说明见下表：

名称	描述	默认值
对象长度（Content-Length）	上传对象的长度，超过流/文件的长度会截断。	流/文件实际长度
对象MIME类型（Content-Type）	对象的MIME类型，定义对象的类型及网页编码，决定浏览器将以什么形式、什么编码读取对象。	application/octet-stream
对象MD5值（Content-MD5）	对象数据的MD5值（经过Base64编码），提供给OBS服务端，校验数据完整性。	无

名称	描述	默认值
对象存储类型	对象的存储类型，不同的存储类型可以满足客户业务对存储性能、成本的不同诉求。默认与桶的存储类型保持一致，可以设置为与桶的存储类型不同。	无
对象自定义元数据	用户对上传到桶中对象的自定义属性描述。	无

设置对象长度

您可以通过ObjectMetadata.setContentLength来设置对象长度。以下代码展示如何设置对象长度：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentLength(1024 * 1024);// 1MB
obsClient.putObject("bucketname", "objectkey", new File("localfile"), metadata);

// 关闭obsClient
obsClient.close();
```

设置对象 MIME 类型

您可以通过ObjectMetadata.setContentType来设置对象MIME类型。以下代码展示如何设置对象MIME类型：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 上传图片
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentType("image/jpeg");
obsClient.putObject("bucketname", "objectkey.jpg", new File("localimage.jpg"), metadata);

// 关闭obsClient
obsClient.close();
```

说明

如果不设置对象MIME类型，SDK会根据上传对象的后缀名自动判断对象MIME类型，如.xml判断为application/xml文件；.html判断为text/html文件。

设置对象 MD5 值

您可以通过ObjectMetadata.setContentMd5来设置对象MD5值。以下代码展示如何设置对象MD5值：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
```

```
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// 上传图片
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentMd5("your md5 which should be encoded by base64");
obsClient.putObject("bucketname", "objectkey", new File("localimage.jpg"), metadata);
// 关闭obsClient
obsClient.close();
```

说明

- 对象数据的MD5值必须经过Base64编码。
- OBS服务端会将该MD5值并与对象数据计算出的MD5值进行对比，如果不匹配则上传失败，返回HTTP 400错误。
- 如果不设置对象的MD5值，OBS服务端会忽略对对象数据的MD5值校验。

设置对象存储类型

您可以通过ObjectMetadata.setStorageClass来设置对象存储类型。以下代码展示如何设置对象存储类型：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObjectMetadata metadata = new ObjectMetadata();
// 设置对象存储类型为归档存储
metadata.setStorageClass(S3Bucket.GLACIER);
obsClient.putObject("bucketname", "objectkey", new File("localfile"), metadata);

// 关闭obsClient
obsClient.close();
```

说明

- 如果不设置，对象的存储类型默认与桶的存储类型保持一致。
- 对象的存储类型分为三类，其含义与**桶存储类型**一致。
- 下载归档存储类型的对象前必须将其取回，参见**下载归档存储对象**。

设置对象自定义元数据

您可以通过ObjectMetadata.addUserMetadata来设置对象自定义元数据。以下代码展示如何设置对象自定义元数据：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObjectMetadata metadata = new ObjectMetadata();
metadata.addUserMetadata("property1", "property-value1");
metadata.getMetadata().put("property2", "property-value2");
obsClient.putObject("bucketname", "objectkey", new File("localfile"), metadata);

// 关闭obsClient
obsClient.close();
```

说明

- 在上面设置对象自定义元数据示例代码中，用户自定义了一个名称为“property1”，值为“property-value1”的元数据和一个名称为“property2”，值为“property-value2”的元数据。
- 一个对象可以有多个元数据，总大小不能超过8KB。
- 对象的自定义元数据可以通过ObsClient.getObjectMetadata获取，参见**获取对象元数据**。
- 使用ObsClient.getObject下载对象时，对象的自定义元数据也会同时下载。

7.5 分段上传

对于较大文件上传，可以切分成段上传。用户可以在如下的应用场景内（但不仅限于此），使用分段上传的模式：

- 上传超过100MB大小的文件。
- 网络条件较差，和OBS服务端之间的链接经常断开。
- 上传前无法确定将要上传文件的大小。

分段上传分为如下3个步骤：

步骤1 初始化分段上传任务（initiateMultipartUpload）。

步骤2 逐个或并行上传段（uploadPart）。

步骤3 合并段（completeMultipartUpload）或取消分段上传任务(abortMultipartUpload)。

----结束

初始化分段上传任务

使用分段上传方式传输数据前，必须先通知OBS初始化一个分段上传任务。该操作会返回一个OBS服务端创建的全局唯一标识（Upload ID），用于标识本次分段上传任务。您可以根据这个唯一标识来发起相关的操作，如取消分段上传任务、列举分段上传任务、列举已上传的段等。

您可以通过ObsClient.initiateMultipartUpload初始化一个分段上传任务：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest("bucketname",
"objectkey");
ObjectMetadata metadata = new ObjectMetadata();
metadata.addUserMetadata("property", "property-value");
metadata.setContentType("text/plain");
request.setMetadata(metadata);
InitiateMultipartUploadResult result = obsClient.initiateMultipartUpload(request);

String uploadId = result.getUploadId();
System.out.println("\t" + uploadId);

// 关闭obsClient
obsClient.close();
```

说明

- 用InitiateMultipartUploadRequest指定上传对象的名称和所属桶。
- 在InitiateMultipartUploadRequest中，您可以设置对象MIME类型、对象自定义元数据。
- InitiateMultipartUploadResult.getUploadId返回分段上传任务的全局唯一标识（Upload ID），在后面的操作中将用到它。

上传段

初始化一个分段上传任务之后，可以根据指定的对象名和Upload ID来分段上传数据。每一个上传的段都有一个标识它的号码——分段号（Part Number，范围是1~10000）。对于同一个Upload ID，该分段号不但唯一标识这一段数据，也标识了这段数据在整个对象内的相对位置。如果您用同一个分段号上传了新的数据，那么OBS上已有的这个段号的数据将被覆盖。除了最后一段以外，其他段的大小范围是5MB~5GB；最后段大小范围是0~5GB。每个段不需要按顺序上传，甚至可以在不同进程、不同机器上上传，OBS会按照分段号排序组成最终对象。

您可以通过ObsClient.uploadPart上传段：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

List<PartEtag> partEtags = new ArrayList<PartEtag>();
// 上传第一段
UploadPartRequest request = new UploadPartRequest("bucketname", "objectkey");
// 设置Upload ID
request.setUploadId(uploadId);
// 设置分段号，范围是1~10000，
request.setPartNumber(1);
// 设置将要上传的大文件
request.setFile(new File("localfile"));

// 设置分段大小
request.setPartSize(5 * 1024 * 1024l);
UploadPartResult result = obsClient.uploadPart(request);
partEtags.add(new PartEtag(result.getEtag(), result.getPartNumber()));

// 上传第二段
request = new UploadPartRequest("bucketname", "objectkey");
// 设置Upload ID
request.setUploadId(uploadId);
// 设置分段号
request.setPartNumber(2);
// 设置将要上传的大文件
request.setFile(new File("localfile"));
// 设置第二段的段偏移量
request.setOffset(5 * 1024 * 1024l);
// 设置分段大小
request.setPartSize(5 * 1024 * 1024l);
result = obsClient.uploadPart(request);
partEtags.add(new PartEtag(result.getEtag(), result.getPartNumber()));

// 关闭obsClient
obsClient.close();
```

说明

- 上传段接口要求除最后一段以外，其他的段大小都要大于5MB。但是上传段接口并不会立即校验上传段的大小（因为不知道是否为最后一块）；只有调用合并段接口时才会校验。
- OBS会将服务端收到段数据的ETag值（段数据的MD5值）返回给用户。
- 为了保证数据在网络传输过程中不出现错误，可以通过设置UploadPartRequest.setAttachMd5为true（默认为false）来让SDK自动计算每段数据的MD5值，并放到Content-MD5请求头中；OBS服务端会计算上传数据的MD5值与SDK计算的MD5值比较，保证数据完整性。
- 可以通过UploadPartRequest.setContentMd5直接设置上传数据的MD5值，提供给OBS服务端用于校验数据完整性。如果设置了该值，UploadPartRequest.setAttachMd5参数会被忽略。
- 段号的范围是1~10000。如果超出这个范围，OBS将返回400 Bad Request错误。

合并段

所有分段上传完成后，需要调用合并段接口来在OBS服务端生成最终对象。在执行该操作时，需要提供所有有效的分段列表（包括分段号和分段ETag值）；OBS收到提交的分段列表后，会逐一验证每个段的有效性。当所有段验证通过后，OBS将把这些分段组合成最终的对象。

您可以通过ObsClient.completeMultipartUpload合并段：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

List<PartEtag> partEtags = new ArrayList<PartEtag>();

Collections.sort(partEtags, new Comparator<PartEtag>()
{
    @Override
    public int compare(PartEtag o1, PartEtag o2)
    {
        return o1.getPartNumber() - o2.getPartNumber();
    }
});

CompleteMultipartUploadRequest request = new CompleteMultipartUploadRequest("bucketname",
"objectkey", uploadId, partEtags);

obsClient.completeMultipartUpload(request);

// 关闭obsClient
obsClient.close();
```

说明

- 上面代码中的 partEtags是进行上传段后保存的分段号和分段ETag值的列表，它必须是按分段号升序排列。
- 分段可以是不连续的。

并发分段上传

分段上传的主要目的是解决大文件上传或网络条件较差的情况。下面的示例代码展示了如何使用分段上传并发上传大文件：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
final String bucketName = "bucketname";
```

```
final String objectKey = "objectkey";
// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 初始化线程池
ExecutorService executorService = Executors.newFixedThreadPool(20);
final File largeFile = new File("localfile");

// 初始化分段上传任务
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(bucketName, objectKey);
InitiateMultipartUploadResult result = obsClient.initiateMultipartUpload(request);

final String uploadId = result.getUploadId();
System.out.println("\t"+ uploadId + "\n");

// 每段上传100MB
long partSize = 100 * 1024 * 1024l;
long fileSize = largeFile.length();

// 计算需要上传的段数
long partCount = fileSize % partSize == 0 ? fileSize / partSize : fileSize / partSize + 1;

final List<PartEtag> partEtags = Collections.synchronizedList(new ArrayList<PartEtag>());

// 执行并发上传段
for (int i = 0; i < partCount; i++)
{
    // 分段在文件中的起始位置
    final long offset = i * partSize;
    // 分段大小
    final long currPartSize = (i + 1 == partCount) ? fileSize - offset : partSize;
    // 分段号
    final int partNumber = i + 1;
    executorService.execute(new Runnable()
    {
        @Override
        public void run()
        {
            UploadPartRequest uploadPartRequest = new UploadPartRequest();
            uploadPartRequest.setBucketName(bucketName);
            uploadPartRequest.setObjectKey(objectKey);
            uploadPartRequest.setUploadId(uploadId);
            uploadPartRequest.setFile(largeFile);
            uploadPartRequest.setPartSize(currPartSize);
            uploadPartRequest.setOffset(offset);
            uploadPartRequest.setPartNumber(partNumber);

            UploadPartResult uploadPartResult;
            try
            {
                uploadPartResult = obsClient.uploadPart(uploadPartRequest);
                System.out.println("Part#" + partNumber + " done\n");
                partEtags.add(new PartEtag(uploadPartResult.getEtag(),
uploadPartResult.getPartNumber()));
            }
            catch (ObsException e)
            {
                e.printStackTrace();
            }
        }
    });
}

// 等待上传完成
executorService.shutdown();
while (!executorService.isTerminated())
{
    try
    {
```

```
        executorService.awaitTermination(5, TimeUnit.SECONDS);
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
}

// 合并段
Collections.sort(partEtags, new Comparator<PartEtag>()
{

    @Override
    public int compare(PartEtag o1, PartEtag o2)
    {
        return o1.getPartNumber() - o2.getPartNumber();
    }
});

CompleteMultipartUploadRequest completeMultipartUploadRequest = new
CompleteMultipartUploadRequest(bucketName, objectKey, uploadId, partEtags);
obsClient.completeMultipartUpload(completeMultipartUploadRequest);

// 关闭obsClient
obsClient.close();
```

说明

大文件分段上传时，使用UploadPartRequest.setOffset和UploadPartRequest.setPartSize来设置每段的起始结束位置。

取消分段上传任务

分段上传任务可以被取消，当一个分段上传任务被取消后，就不能再使用其Upload ID做任何操作，已经上传段也会被OBS删除。

您可以通过ObsClient.abortMultipartUpload取消分段上传任务：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AbortMultipartUploadRequest request = new AbortMultipartUploadRequest("bucketname", "objectkey",
uploadId);

obsClient.abortMultipartUpload(request);

// 关闭obsClient
obsClient.close();
```

列举已上传的段

您可使用ObsClient.listParts列举出某一分段上传任务所有已经上传成功的段。

该接口可设置的参数如下：

参数	作用	OBS Java SDK对应方法
bucketName	分段上传任务所属的桶名。	ListPartsRequest.setBucketName

参数	作用	OBS Java SDK对应方法
key	分段上传任务所属的对象名。	ListPartsRequest.setKey
uploadId	分段上传任务全局唯一标识，从initiateMultipartUpload返回的结果获取。	ListPartsRequest.setUploadId
maxParts	表示列举已上传的段返回结果最大段数目，即分页时每一页中段数目。	ListPartsRequest.setMaxParts
partNumberMarker	表示待列出段的起始位置，只有Part Number大于该参数的段会被列出。	ListPartsRequest.setPartNumberMarker

- 简单列举

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

//列举已上传的段，其中uploadId来自于initiateMultipartUpload
ListPartsRequest request = new ListPartsRequest("bucketname", "objectkey");
request.setUploadId(uploadId);
ListPartsResult result = obsClient.listParts(request);

for(Multipart part : result.getMultipartList()){
    // 分段号，上传时候指定
   System.out.println("\t"+part.getPartNumber());
    // 段数据大小
   System.out.println("\t"+part.getSize());
    // 分段的ETag值
   System.out.println("\t"+part.getEtag());
    // 段的最后上传时间
   System.out.println("\t"+part.getLastModified());
}

// 关闭obsClient
obsClient.close();
```

说明

- 列举段至多返回1000个段信息，如果指定的Upload ID包含的段数量大于1000，则返回结果中ListPartsResult.isTruncated为true表明本次没有返回全部段，并可通过ListPartsResult.getNextPartNumberMarker获取下次列举的起始位置。
- 如果想获取指定Upload ID包含的所有分段，可以采用分页列举的方式。

- 列举所有段

由于ObsClient.listParts只能列举至多1000个段，如果段数量大于1000，列举所有分段请参考如下示例：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// 创建ObsClient实例
```

```
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 列举所有已上传的段，其中uploadId来自于initiateMultipartUpload
ListPartsRequest request = new ListPartsRequest("bucketname", "objectkey");
request.setUploadId(uploadId);
ListPartsResult result;

do{
    result = obsClient.listParts(request);
    for(Multipart part : result.getMultipartList()){
        // 分段号，上传时候指定
       System.out.println("\t"+part.getPartNumber());
        // 段数据大小
       System.out.println("\t"+part.getSize());
        // 分段的ETag值
       System.out.println("\t"+part.getEtag());
        // 段的最后上传时间
       System.out.println("\t"+part.getLastModified());
    }
    request.setPartNumberMarker(Integer.parseInt(result.getNextPartNumberMarker()));
}while(result.isTruncated());

// 关闭obsClient
obsClient.close();
```

- 分页列举所有段

上面的获取所有已上传段（每页1000个段）是分页的一种特殊情况。如果需要指定每页段的数量，请参考以下代码：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 分页列举已上传的段，其中uploadId来自于initiateMultipartUpload
ListPartsRequest request = new ListPartsRequest("bucketname", "objectkey");
request.setUploadId(uploadId);
// 每页100个段
request.setMaxParts(100);
ListPartsResult result;

do{
    result = obsClient.listParts(request);
    for(Multipart part : result.getMultipartList()){
        // 分段号，上传时候指定
       System.out.println("\t"+part.getPartNumber());
        // 段数据大小
       System.out.println("\t"+part.getSize());
        // 分段的ETag值
       System.out.println("\t"+part.getEtag());
        // 段的最后上传时间
       System.out.println("\t"+part.getLastModified());
    }

    request.setPartNumberMarker(Integer.parseInt(result.getNextPartNumberMarker()));
}while(result.isTruncated());

// 关闭obsClient
obsClient.close();
```

列举分段上传任务

您可以通过ObsClient.listMultipartUploads列举分段上传任务。列举分段上传任务可设置的参数如下：

参数	作用	OBS Java SDK对应方法
bucketName	桶名。	ListMultipartUploadsRequest.setBucketName
prefix	限定返回的分段上传任务中的对象名必须带有prefix前缀。	ListMultipartUploadsRequest.setPrefix
delimiter	用于对分段上传任务中的对象名进行分组的字符。对于对象名中包含delimiter的任务，其对象名（如果请求中指定了prefix，则此处的对象名需要去掉prefix）中从首字符至第一个delimiter之间的字符串将作为一个分组并作为commonPrefix返回。	ListMultipartUploadsRequest.setDelimiter
maxUploads	列举分段上传任务的最大数目，取值范围为1~1000，当超出范围时，按照默认的1000进行处理。	ListMultipartUploadsRequest.setMaxUploads
keyMarker	表示列举时返回指定的keyMarker之后的分段上传任务。	ListMultipartUploadsRequest.setKeyMarker
uploadIdMarker	只有与keyMarker参数一起使用时才有意义，用于指定返回结果的起始位置，即列举时返回指定keyMarker的uploadIdMarker之后的分段上传任务。	ListMultipartUploadsRequest.setUploadIdMarker

- 简单列举分段上传任务

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListMultipartUploadsRequest request = new ListMultipartUploadsRequest("bucketname");

MultipartUploadListing result = obsClient.listMultipartUploads(request);
for(MultipartUpload upload : result.getMultipartTaskList()){
    System.out.println("\t" + upload.getUploadId());
    System.out.println("\t" + upload.getObjectKey());
    System.out.println("\t" + upload.getInitiatedDate());
}

// 关闭obsClient
obsClient.close();
```

说明

- 列举分段上传任务至多返回1000个任务信息，如果指定的桶包含的分段上传任务数量大于1000，则MultipartUploadListing.isTruncated为true表明本次没有返回全部结果，并可通过MultipartUploadListing.getNextKeyMarker和MultipartUploadListing.getNextUploadIdMarker获取下次列举的起点。
- 如果想获取指定桶包含的所有分段上传任务，可以采用分页列举的方式。

- 列举全部分段上传任务

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListMultipartUploadsRequest request = new ListMultipartUploadsRequest("bucketname");
MultipartUploadListing result;

do{
    result = obsClient.listMultipartUploads(request);
    for(MultipartUpload upload : result.getMultipartTaskList()){
        System.out.println("\t" + upload.getUploadId());
        System.out.println("\t" + upload.getObjectKey());
        System.out.println("\t" + upload.getInitiatedDate());
    }
    request.setKeyMarker(result.getNextKeyMarker());
    request.setUploadIdMarker(result.getNextUploadIdMarker());
}while(result.isTruncated());

// 关闭obsClient
obsClient.close();
```

- 分页列举全部分段上传任务

上面的获取所有分段上传任务（每页1000个任务）是分页的一种特殊情况。如果需要指定每页任务的数量，请参考以下代码：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListMultipartUploadsRequest request = new ListMultipartUploadsRequest("bucketname");
// 每页100个分段上传任务
request.setMaxUploads(100);
MultipartUploadListing result;

do{
    result = obsClient.listMultipartUploads(request);
    for(MultipartUpload upload : result.getMultipartTaskList()){
        System.out.println("\t" + upload.getUploadId());
        System.out.println("\t" + upload.getObjectKey());
        System.out.println("\t" + upload.getInitiatedDate());
    }
    request.setKeyMarker(result.getNextKeyMarker());
    request.setUploadIdMarker(result.getNextUploadIdMarker());
}while(result.isTruncated());

// 关闭obsClient
obsClient.close();
```

7.6 分段复制

分段复制是分段上传的一种特殊情况，即分段上传任务中的段通过复制OBS指定桶中现有对象（或对象的一部分）来实现。您可以通过ObsClient.copyPart来复制段。以下代码展示了如何使用分段复制模式复制大对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

final String destBucketName = "destbucketname";
final String destObjectKey = "destobjectkey";
final String sourceBucketName = "sourcebucketname";
final String sourceObjectKey = "sourceobjectkey";
// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 初始化线程池
ExecutorService executorService = Executors.newFixedThreadPool(20);

// 初始化分段上传任务
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(destBucketName,
destObjectKey);
InitiateMultipartUploadResult result = obsClient.initiateMultipartUpload(request);

final String uploadId = result.getUploadId();
System.out.println("\t"+ uploadId + "\n");

// 获取大对象信息
ObjectMetadata metadata = obsClient.getObjectMetadata(sourceBucketName, sourceObjectKey);
// 每段复制100MB
long partSize = 100 * 1024 * 1024l;
long objectSize = metadata.getContentLength();

// 计算需要复制的段数
long partCount = objectSize % partSize == 0 ? objectSize / partSize : objectSize / partSize + 1;

final List<PartEtag> partEtags = Collections.synchronizedList(new ArrayList<PartEtag>());

// 执行并发复制段
for (int i = 0; i < partCount; i++)
{
    // 复制段起始位置
    final long rangeStart = i * partSize;
    // 复制段结束位置
    final long rangeEnd = (i + 1 == partCount) ? objectSize - 1 : rangeStart + partSize - 1;
    // 分段号
    final int partNumber = i + 1;
    executorService.execute(new Runnable()
    {

        @Override
        public void run()
        {
            CopyPartRequest request = new CopyPartRequest();
            request.setUploadId(uploadId);
            request.setSourceBucketName(sourceBucketName);
            request.setSourceObjectKey(sourceObjectKey);
            request.setDestinationBucketName(destBucketName);
            request.setDestinationObjectKey(destObjectKey);
            request.setByteRangeStart(rangeStart);
            request.setByteRangeEnd(rangeEnd);
            request.setPartNumber(partNumber);
            CopyPartResult result;
            try
            {
```

```
                result = obsClient.copyPart(request);
                System.out.println("Part#" + partNumber + " done\n");
                partEtags.add(new PartEtag(result.getEtag(), result.getPartNumber()));
            }
            catch (ObsException e)
            {
                e.printStackTrace();
            }
        }
    });
}

// 等待复制完成
executorService.shutdown();
while (!executorService.isTerminated())
{
    try
    {
        executorService.awaitTermination(5, TimeUnit.SECONDS);
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
}

// 合并段
Collections.sort(partEtags, new Comparator<PartEtag>()
{

    @Override
    public int compare(PartEtag o1, PartEtag o2)
    {
        return o1.getPartNumber() - o2.getPartNumber();
    }
});
CompleteMultipartUploadRequest completeMultipartUploadRequest = new
CompleteMultipartUploadRequest(destBucketName, destObjectKey, uploadId, partEtags);
obsClient.completeMultipartUpload(completeMultipartUploadRequest);

// 关闭obsClient
obsClient.close();
```

7.7 断点续传上传

当上传大文件时，经常出现因网络不稳定或程序崩溃导致上传失败的情况。失败后再次重新上传不仅浪费资源，而且当网络不稳定时仍然有上传失败的风险。断点续传上传接口能有效地解决此类问题引起的上传失败，其原理是将待上传的文件分成若干个分段分别上传，并实时地将每段上传结果统一记录在checkpoint文件中，仅当所有分段都上传成功时返回上传成功的结果，否则抛出异常提醒用户再次调用接口进行重新上传（重新上传时因为有checkpoint文件记录当前的上传进度，避免重新上传所有分段，从而节省资源提高效率）。

您可以通过ObsClient.uploadFile进行断点续传上传。该接口可设置的参数如下：

参数	作用	OBS Java SDK对应方法
bucketName	桶名，必选参数。	UploadFileRequest.setBucketName
objectKey	对象名，必选参数。	UploadFileRequest.setObjectKey

参数	作用	OBS Java SDK对应方法
uploadFile	待上传的本地文件，必选参数。	UploadFileRequest.setUploadFile
partSize	分段大小，单位字节，取值范围是5MB~5GB，默认为5MB。	UploadFileRequest.setPartSize
taskNum	分段上传时的最大并发数，默认为1。	UploadFileRequest.setTaskNum
enableCheckpoint	是否开启断点续传模式，默认为false，表示不开启。	UploadFileRequest.setEnableCheckpoint
checkpointFile	记录上传进度的文件，只在断点续传模式下有效。当该值为空时，默认与待上传的本地文件同目录。	UploadFileRequest.setCheckpointFile
objectMetadata	对象的属性。	UploadFileRequest.setObjectMetadata
enableCheckSum	是否校验待上传文件的内容，只在断点续传模式下有效。默认为false，表示不校验。	UploadFileRequest.setEnableCheckSum

以下代码展示了如何使用断点续传上传接口上传文件：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

UploadFileRequest request = new UploadFileRequest("bucketname", "objectkey");
// 设置待上传的本地文件
request.setUploadFile("localfile");
// 设置分段上传时的最大并发数
request.setTaskNum(5);
// 设置分段大小为10MB
request.setPartSize(10 * 1024 * 1024);
// 开启断点续传模式
request.setEnableCheckpoint(true);
try{
    // 进行断点续传上传
    CompleteMultipartUploadResult result = obsClient.uploadFile(request);
}catch (ObsException e) {
    // 发生异常时可再次调用断点续传上传接口进行重新上传
}finally{
    // 关闭obsClient
    obsClient.close();
}
```

说明

- 断点续传上传接口是利用**分段上传**特性实现的，是对分段上传的封装和加强。
- 断点续传上传接口不仅能在失败重传时节省资源提高效率，还因其对分段进行并发上传的机制能加快上传速度，帮助用户快速完成上传业务；且其对用户透明，用户不用关心checkpoint文件的创建和删除、分段任务的切分、并发上传的实现等内部细节。
- **enableCheckpoint参数**默认是false，代表不启用断点续传模式，此时断点续传上传接口退化成对分段上传的简单封装，不会产生checkpoint文件。
- **checkpointFile参数**和**enableCheckSum参数**仅在**enableCheckpoint参数**为true时有效。

7.8 基于表单上传

您可以通过ObsClient.createV4PostSignature生成基于表单上传的请求参数。使用Java代码模拟表单上传的完整代码示例，参见PostObjectSample.zip。您也可以通过如下步骤进行表单上传：

步骤1 使用ObsClient.createV4PostSignature生成用于鉴权的请求参数。

步骤2 准备表单HTML页面。

步骤3 将生成的请求参数填入HTML页面。

步骤4 选择本地文件，进行表单上传。

----结束

说明

使用SDK生成的用于鉴权的请求参数包括五个:

- policy，对应表单中policy字段，
- algorithm，对应表单中的x-amz-algorithm字段。
- credential，对应表单中的x-amz-credential字段。
- date，对应表单中的x-amz-date字段。
- signature，对应表单中的x-amz-signature字段。

以下代码展示了如何生成基于表单上传的请求参数：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

V4PostSignatureRequest request = new V4PostSignatureRequest();
// 设置表单参数
Map<String, Object> formParams = new HashMap<String, Object>();
// 设置对象访问权限为公共读
formParams.put("acl", "public-read");
// 设置对象MIME类型
formParams.put("content-type", "text/plain");

// 设置对象自定义元数据
formParams.put("property1", "property-value1");
formParams.put("property2", "property-value2");

request.setFormParams(formParams);
// 设置表单上传请求有效期，单位：秒
request.setExpires(3600);
V4PostSignatureResponse response = obsClient.createV4PostSignature(request);
```

```
// 获取表单上传请求参数
System.out.println("\t" + response.getPolicy());
System.out.println("\t" + response.getAlgorithm());
System.out.println("\t" + response.getCredential());
System.out.println("\t" + response.getDate());
System.out.println("\t" + response.getSignature());

// 关闭obsClient
obsClient.close();
```

示例表单HTML代码如下：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<!-- 写法1：OBS服务地址 + 桶名 -->
<form action="http://obs.esdk.com/bucketname" method="post" enctype="multipart/form-data">

<!-- 写法2：桶名.OBS服务地址 -->
<!--
<form action="http://bucketname.obs.esdk.com/" method="post" enctype="multipart/form-data">
-->
Object key
<!-- 对象名 -->
<input type="text" name="key" value="objectkey" />
<p>
ACL
<!-- 对象ACL权限 -->
<input type="text" name="acl" value="public-read" />
<p>
Content-Type
<!-- 对象MIME类型 -->
<input type="text" name="content-type" value="text/plain" />
<p>
<!-- 自定义元数据 -->
<input type="text" name="x-amz-meta-property1" value="property-value1" />
<p>
<input type="text" name="x-amz-meta-property2" value="property-value2" />
<!-- policy的base64编码值 -->
<input type="hidden" name="policy" value="*** Provide your policy ***" />
<!-- 算法 -->
<input type="hidden" name="x-amz-algorithm" value="AWS4-HMAC-SHA256"/>
<!-- AK+短日期+区域+/s3/aws4_request -->
<input type="hidden" name="x-amz-credential" value="*** Provide your credential ***"/>
<!-- 日期 -->
<input type="hidden" name="x-amz-date" value="*** Provide your date ***"/>
<!-- 签名串信息 -->
<input type="hidden" name="x-amz-signature" value="*** Provide your signature ***"/>

<input name="file" type="file" />
<input name="submit" value="Upload" type="submit" />
</form>
</body>
</html>
```

说明

- HTML表单中的policy，x-amz-algorithm，x-amz-credential，x-amz-date，x-amz-signature的值均是从ObsClient.createV4PostSignature的返回结果中获取。
- 您可以直接下载表单HTML示例PostDemo.html。

8 下载对象

OBS Java SDK提供了丰富的对象下载接口，可以通过以下方式下载对象：

- 流式下载
- 范围下载
- 限定条件下载
- 断点续传下载

您可以通过ObsClient.getObject下载对象。

8.1 流式下载

以下代码展示了如何进行流式下载：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

S3Object s3Object = obsClient.getObject("bucketname", "objectkey");

// 读取对象内容
System.out.println("Object content:");
BufferedReader br = new BufferedReader(new InputStreamReader(s3Object.getObjectContent(),
"UTF-8"));

String line;
```

```
while((line = br.readLine()) != null){
    System.out.println(line);
    System.out.println();
}

br.close();

// 关闭obsClient
obsClient.close();
```

说明

- ObsClient.getObject的返回实例S3Object实例包含对象所在的桶、对象名、对象属性、对象输入流等。
- 通过操作对象输入流可将对象的内容读取到本地文件或者内存中。

注意

S3Object.getObjectContent获取的对象输入流一定要显式关闭，否则会造成资源泄露。

8.2 范围下载

如果只需要下载对象的其中一部分数据，可以使用范围下载，下载指定范围的数据。如果指定的下载范围是0~1000，则返回第0到第1000个字节的数据，包括第1000个，共1001字节的数据，即[0, 1000]。如果指定的范围无效，则返回整个对象的数据。以下代码展示了如何进行范围下载：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

GetObjectRequest request = new GetObjectRequest("bucketname", "objectkey");
// 指定开始和结束范围
request.setRangeStart(0l);
request.setRangeEnd(1000l);
S3Object s3Object = obsClient.getObject(request);

// 读取数据
byte[] buf = new byte[1024];
InputStream in = s3Object.getObjectContent();
for (int n = 0; n != -1; ) {
    n = in.read(buf, 0, buf.length);
}

in.close();

// 关闭obsClient
obsClient.close();
```

说明

- 如果指定的范围无效（比如开始位置、结束位置为负数，大于文件大小），则会返回整个对象。
- 可以利用范围下载并发下载大对象，详细代码示例请参考ConcurrentDownloadObjectSample.zip。

8.3 限定条件下载

下载对象时，可以指定一个或多个限定条件，满足限定条件时则进行下载，否则抛出异常，下载对象失败。

您可以使用的限定条件如下：

参数	作用	OBS Java SDK对应方法
If-Modified-Since	如果对象的修改时间晚于该参数值指定的时间，则返回对象内容，否则抛出异常。	GetObjectRequest.setIfModifiedSince
If-Unmodified-Since	如果对象的修改时间早于该参数值指定的时间，则返回对象内容，否则抛出异常。	GetObjectRequest.setIfUnmodifiedSince
If-Match	如果对象的ETag值与该参数值相同，则返回对象内容，否则抛出异常。	GetObjectRequest.setIfMatchTag
If-None-Match	如果对象的ETag值与该参数值不相同，则返回对象内容，否则抛出异常。	GetObjectRequest.setIfNoneMatchTag

说明

- 对象的ETag值是指对象数据的MD5校验值。
- 如果包含If-Unmodified-Since并且不符合或者包含If-Match并且不符合，抛出异常中HTTP状态码为：412 precondition failed。
- 如果包含If-Modified-Since并且不符合或者包含If-None-Match并且不符合，抛出异常中HTTP状态码为：304 Not Modified。

以下代码展示了如何进行限定条件下载：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

GetObjectRequest request = new GetObjectRequest("bucketname", "objectkey");
request.setRangeStart(01);
request.setRangeEnd(10001);

request.setIfModifiedSince(new SimpleDateFormat("yyyy-MM-dd").parse("2016-01-01"));
S3Object s3Object = obsClient.getObject(request);

s3Object.getObjectContent().close();

// 关闭obsClient
obsClient.close();
```

8.4 重写响应头

下载对象时，可以重写部分HTTP/HTTPS响应头信息。可重写的响应头信息见下表：

参数	作用	OBS Java SDK对应方法
contentType	重写HTTP/HTTPS响应中的Content-Type	ObjectRepleaceMetadata.setContentType
contentLanguage	重写HTTP/HTTPS响应中的Content-Language	ObjectRepleaceMetadata.setContentLanguage
expires	重写HTTP/HTTPS响应中的Expires	ObjectRepleaceMetadata.setExpires
cacheControl	重写HTTP/HTTPS响应中的Cache-Control	ObjectRepleaceMetadata.setCacheControl
contentDisposition	重写HTTP/HTTPS响应中的Content-Disposition	ObjectRepleaceMetadata.setContentDisposition
contentEncoding	重写HTTP/HTTPS响应中的Content-Encoding	ObjectRepleaceMetadata.setContentEncoding

以下代码展示了如何重写响应头：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

GetObjectRequest request = new GetObjectRequest("bucketname", "objectkey");
ObjectRepleaceMetadata replaceMetadata = new ObjectRepleaceMetadata();
replaceMetadata.setContentType("image/jpeg");
request.setReplaceMetadata(replaceMetadata);

S3Object object = obsClient.getObject(request);
System.out.println(object.getMetadata().getContentType());

// 关闭obsClient
obsClient.close();
```

8.5 获取自定义元数据

下载对象成功后会返回对象的自定义元数据。以下代码展示了如何获取自定义元数据：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 上传对象，设置自定义元数据
```

```
PutObjectRequest request = new PutObjectRequest("bucketname", "objectkey");
ObjectMetadata metadata = new ObjectMetadata();
metadata.addUserMetadata("property", "property-value");
request.setMetadata(metadata);
obsClient.putObject(request);

// 下载对象，获取对象自定义元数据
S3Object s3Object = obsClient.getObject("bucketname", "objectkey");
System.out.println(s3Object.getMetadata().getUserMetadata("property"));

s3Object.getObjectContent().close();

// 关闭obsClient
obsClient.close();
```

8.6 下载归档存储对象

归档存储对象是指存放于归档存储类型（GLACIER）桶中的对象。如果要下载归档存储对象，需要先将归档存储对象取回。取回归档存储对象的取回选项可支持三类，见下表：

选项	说明	OBS Java SDK对应值
快速取回	取回耗时1~5分钟。	RestoreObjectRequest.EXPEDITED
标准取回	取回耗时3~5小时。默认值。	RestoreObjectRequest.STANDARD
批量取回	取回耗时5~12小时。	RestoreObjectRequest.BULK

您可以通过ObsClient.restoreObject取回归档存储对象。以下代码展示了如何下载归档存储对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 取回归档存储对象
RestoreObjectRequest request = new RestoreObjectRequest();
request.setBucketName("bucketname");
request.setObjectKey("objectkey");
request.setDays(1);
request.setTier(RestoreObjectRequest.EXPEDITED);
obsClient.restoreObject(request);

// 等待对象取回
Thread.sleep(60 * 6 * 1000);

// 下载对象
S3Object s3Object = obsClient.getObject("bucketname", "objectkey");

s3Object.getObjectContent().close();

// 关闭obsClient
obsClient.close();
```

说明

- ObsClient.restoreObject中指定的桶必须是归档存储类型，否则调用该接口会报错。
- RestoreObjectRequest.setDays指定取回对象保存的时间，取值范围是1~30。
- RestoreObjectRequest.setTier指定取回选项，表示取回对象所耗的时间。

8.7 断点续传下载

当下载大对象到本地文件时，经常出现因网络不稳定或程序崩溃导致下载失败的情况。失败后再次重新下载不仅浪费资源，而且当网络不稳定时仍然有下载失败的风险。断点续传下载接口能有效地解决此类问题引起的下载失败，其原理是将待下载的对象分成若干个分段分别下载，并实时地将每段下载结果统一记录在checkpoint文件中，仅当所有分段都下载成功时返回下载成功的结果，否则抛出异常提醒用户再次调用接口进行重新下载（重新下载时因为有checkpoint文件记录当前的下载进度，避免重新下载所有分段，从而节省资源提高效率）。

您可以通过ObsClient.downloadFile进行断点续传下载。该接口可设置的参数如下：

参数	作用	OBS Java SDK对应方法
bucketName	桶名，必选参数。	DownloadFileRequest.setBucketName
objectKey	对象名，必选参数。	DownloadFileRequest.setObjectKey
downloadFile	下载对象的本地文件路径。当该值为空时，默认为当前程序的运行目录。	DownloadFileRequest.setDownloadFile
partSize	分段大小，单位字节，取值范围是5MB~5GB，默认为5MB。	DownloadFileRequest.setPartSize
taskNum	分段下载时的最大并发数，默认为1。	DownloadFileRequest.setTaskNum
enableCheckpoint	是否开启断点续传模式，默认为false，表示不开启。	DownloadFileRequest.setEnableCheckpoint
checkpointFile	记录下载进度的文件，只在断点续传模式下有效。当该值为空时，默认与下载对象的本地文件路径同目录。	DownloadFileRequest.setCheckpointFile
versionId	对象的版本号。	DownloadFileRequest.setVersionId
ifModifiedSince	如果对象的修改时间晚于该参数值指定的时间，则返回对象内容，否则抛出异常。	DownloadFileRequest.setIfModifiedSince

参数	作用	OBS Java SDK对应方法
ifUnmodifiedSince	如果对象的修改时间早于该参数值指定的时间，则返回对象内容，否则抛出异常。	DownloadFileRequest.setIfUnmodifiedSince
ifMatchTag	如果对象的ETag值与该参数值相同，则返回对象内容，否则抛出异常。	DownloadFileRequest.setIfMatchTag
ifNoneMatchTag	如果对象的ETag值与该参数值不相同，则返回对象内容，否则抛出异常。	DownloadFileRequest.setIfNoneMatchTag

以下代码展示了如何使用断点续传下载接口下载对象到本地文件时：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
DownloadFileRequest request = new DownloadFileRequest("bucketname", "objectkey");
// 设置下载对象的本地文件路径
request.setDownloadFile("localfile");
// 设置分段下载时的最大并发数
request.setTaskNum(5);
// 设置分段大小为10MB
request.setPartSize(10 * 1024 * 1024);
// 开启断点续传模式
request.setEnableCheckpoint(true);
try{
    // 进行断点续传下载
    DownloadFileResult result = obsClient.downloadFile(request);
}catch (ObsException e) {
    // 发生异常时可再次调用断点续传下载接口进行重新下载
}finally{
    // 关闭obsClient
    obsClient.close();
}
```

说明

- 断点续传下载接口是利用**范围下载**特性实现的，是对范围下载的封装和加强。
- 断点续传下载接口不仅能在失败重下时节省资源提高效率，还因其对分段进行并发下载的机制能加快下载速度，帮助用户快速完成下载业务；且其对用户透明，用户不用关心checkpoint文件的创建和删除、分段任务的切分、并发下载的实现等内部细节。
- **enableCheckpoint参数**默认是false，代表不启用断点续传模式，此时断点续传下载接口退化成对范围下载的简单封装，不会产生checkpoint文件。
- **checkpointFile参数**仅在**enableCheckpoint参数**为true时有效。

9 管理对象

9.1 获取对象属性

您可以通过ObsClient.getObjectMetadata来获取对象属性，包括对象长度，对象MIME类型，对象自定义元数据等信息。以下代码展示了如何获取对象属性：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObjectMetadata metadata = obsClient.getObjectMetadata("bucketname", "objectkey");
System.out.println("\t" + metadata.getContentType());
System.out.println("\t" + metadata.getContentLength());
System.out.println("\t" + metadata.getUserMetadata("property"));


// 关闭obsClient
obsClient.close();
```

9.2 管理对象访问权限

对象访问权限与桶访问权限类似，也可支持预定义访问策略或直接设置，参见**桶访问权限**。

对象访问权限（**ACL**）可以通过三种方式设置：

1. 上传对象时指定预定义访问策略。
2. 调用ObsClient.setObjectAcl指定预定义访问策略。

3. 调用ObsClient.setObjectAcl直接设置。

上传对象时指定预定义访问策略

以下代码展示如何在上传对象时指定预定义访问策略：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

PutObjectRequest request = new PutObjectRequest();
request.setBucketName("bucketname");
request.setObjectKey("objectkey");
request.setFile(new File("localfile"));
// 设置对象访问权限为公共读
request.setAcl(AccessControlList.REST_CANNED_PUBLIC_READ);
obsClient.putObject(request);

// 关闭obsClient
obsClient.close();
```

为对象设置预定义访问策略

以下代码展示如何为对象设置预定义访问策略：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 设置对象访问权限为私有读写
obsClient.setObjectAcl("bucketname", "objectkey", AccessControlList.REST_CANNED_PRIVATE);

// 关闭obsClient
obsClient.close();
```

直接设置对象访问权限

以下代码展示如何直接设置对象访问权限：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
owner.setId("ownerid");
acl.setOwner(owner);
// 为所有用户设置读权限
acl.grantPermission(GroupGrantee.ALL_USERS, Permission.PERMISSION_READ);
// 为授权用户设置写权限
acl.grantPermission(GroupGrantee.AUTHENTICATED_USERS, Permission.PERMISSION_WRITE);
obsClient.setObjectAcl("bucketname", "objectkey", acl);

// 关闭obsClient
obsClient.close();
```

说明

ACL中需要填写的所有者（Owner）或者被授权用户（Grantee）的ID，是指用户的帐户ID，可通过OBS控制台“我的凭证”页面查看。

获取对象访问权限

您可以通过ObsClient.getObjectAcl获取对象的访问权限。以下代码展示如何获取对象访问权限：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AccessControlList acl = obsClient.getObjectAcl("bucketname", "objectkey");
System.out.println(acl);

// 关闭obsClient
obsClient.close();
```

9.3 列举对象

您可以通过ObsClient.listObjects列举出桶里的对象。

该接口可设置的参数如下：

参数	作用	OBS Java SDK对应方法
bucketName	桶名。	ListObjectsRequest.setBucketName
prefix	限定返回的对象名必须带有prefix前缀。	ListObjectsRequest.setPrefix
marker	列举对象的起始位置，返回的对象列表将是对象名按照字典序排序后该参数以后的所有对象。	ListObjectsRequest.setMarker
maxKeys	列举对象的最大数目，取值范围为1~1000，当超出范围时，按照默认的1000进行处理。	ListObjectsRequest.setMaxKeys
delimiter	用于对对象名进行分组的字符。对于对象名中包含delimiter的对象，其对象名（如果请求中指定了prefix，则此处的对象名需要去掉prefix）中从首字符至第一个delimiter之间的字符串将作为一个分组并作为commonPrefix返回。	ListObjectsRequest.setDelimiter

简单列举

以下代码展示如何简单列举对象，最多返回1000个对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObjectListing result = obsClient.listObjects("bucketname");
for(S3Object s3Object : result.getObjectSummaries()){
    System.out.println("\t" + s3Object.getObjectKey());
    System.out.println("\t" + s3Object.getOwner());
}

// 关闭obsClient
obsClient.close();
```

说明

- 每次至多返回1000个对象，如果指定桶包含的对象数量大于1000，则返回结果中ObjectListing.isTruncated为true表明本次没有返回全部对象，并可通过ObjectListing.getNextMarker获取下次列举的起始位置。
- 如果想获取指定桶包含的所有对象，可以采用分页列举的方式。

指定数目列举

以下代码展示如何指定数目列举对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// 只列举100个对象
request.setMaxKeys(100);
ObjectListing result = obsClient.listObjects(request);
for(S3Object s3Object : result.getObjectSummaries()){
    System.out.println("\t" + s3Object.getObjectKey());
    System.out.println("\t" + s3Object.getOwner());
}

// 关闭obsClient
obsClient.close();
```

指定前缀列举

以下代码展示如何指定前缀列举对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// 设置列举带有prefix前缀的100个对象
request.setMaxKeys(100);
request.setPrefix("prefix");
ObjectListing result = obsClient.listObjects(request);
for(S3Object s3Object : result.getObjectSummaries()){
```

```
    System.out.println("\t" + s3Object.getObjectKey());
    System.out.println("\t" + s3Object.getOwner());
}

// 关闭obsClient
obsClient.close();
```

指定起始位置列举

以下代码展示如何指定起始位置列举对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// 设置列举对象名字典序在"test"之后的100个对象
request.setMaxKeys(100);
request.setMarker("test");
ObjectListing result = obsClient.listObjects(request);
for(S3Object s3Object : result.getObjectSummaries()){
    System.out.println("\t" + s3Object.getObjectKey());
    System.out.println("\t" + s3Object.getOwner());
}

// 关闭obsClient
obsClient.close();
```

分页列举全部对象

以下代码展示分页列举全部对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// 设置每页100个对象
request.setMaxKeys(100);

ObjectListing result;
do{
    result = obsClient.listObjects(request);
    for(S3Object s3Object : result.getObjectSummaries()){
        System.out.println("\t" + s3Object.getObjectKey());
        System.out.println("\t" + s3Object.getOwner());
    }

    request.setMarker(result.getNextMarker());
}while(result.isTruncated());

// 关闭obsClient
obsClient.close();
```

列举文件夹中的所有对象

OBS本身是没有文件夹的概念的，桶中存储的元素只有对象。文件夹对象实际上是一个大小为0且对象名以“/”结尾的对象，将这个文件夹对象名作为前缀，即可模拟列举文件夹中对象的功能。以下代码展示如何列举文件夹中的对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// 设置文件夹对象名"dir/"为前缀
request.setPrefix("dir/");
request.setMaxKeys(1000);

ObjectListing result;

do{
    result = obsClient.listObjects(request);
    for (S3Object s3Object : result.getObjectSummaries())
    {
        System.out.println("\t" + s3Object.getObjectKey());
        System.out.println("\t" + s3Object.getOwner());
    }
    request.setMarker(result.getNextMarker());
}while(result.isTruncated());

// 关闭obsClient
obsClient.close();
```

按文件夹分组列举所有对象

以下代码展示如何按文件夹分组，列举桶内所有对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
request.setMaxKeys(1000);
// 设置文件夹分隔符"/"
request.setDelimiter("/");
ObjectListing result = obsClient.listObjects(request);
System.out.println("根目录中的对象：");
for(S3Object s3Object : result.getObjectSummaries()){
    System.out.println("\t" + s3Object.getObjectKey());
    System.out.println("\t" + s3Object.getOwner());
}
listObjectsByPrefix(obsClient, request, result);

// 关闭obsClient
obsClient.close();
```

递归列出子文件夹中对象的函数*listObjectsByPrefix*的示例代码如下：

```
static void listObjectsByPrefix(ObsClient obsClient, ListObjectsRequest request, ObjectListing
result) throws ObsException
{
      for(String prefix : result.getCommonPrefixes()){
          System.out.println("文件夹 [" + prefix + "]中的对象:");
          request.setPrefix(prefix);
          result  = obsClient.listObjects(request);
          for(S3Object s3Object : result.getObjectSummaries()){
              System.out.println("\t" + s3Object.getObjectKey());
              System.out.println("\t" + s3Object.getOwner());
          }
          listObjectsByPrefix(obsClient, request, result);
      }
}
```

说明

- 以上代码示例没有考虑文件夹中对象数超过1000个的情况。
- 由于是需要列举出文件夹中的对象和子文件夹，且文件夹对象总是以“/”结尾，因此delimiter总是为“/”。
- 每次递归的返回结果中ObjectListing.getObjectSummaries包含的是文件夹中的对象；ObjectListing.getCommonPrefixes包含的是文件夹的子文件夹。

9.4 删除对象

删除单个对象

您可以通过ObsClient.deleteObject删除单个对象。以下代码展示如何删除单个对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
obsClient.deleteObject("bucketname", "objectkey");

// 关闭obsClient
obsClient.close();
```

批量删除对象

您可以通过ObsClient.deleteObjects批量删除对象。

每次最多删除1000个对象，并支持两种响应模式：详细（verbose）模式和简单（quiet）模式。

- 详细模式：返回的删除成功和删除失败的所有结果，默认模式。
- 简单模式：只返回的删除过程中出错的结果。

以下代码展示了如何进行批量删除对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

DeleteObjectsRequest request = new DeleteObjectsRequest("bucketname");
// 设置为verbose模式
request.setQuiet(false);

List<KeyAndVersion> toDelete = new ArrayList<KeyAndVersion>();

toDelete.add(new KeyAndVersion("objectkey1"));
toDelete.add(new KeyAndVersion("objectkey2"));
toDelete.add(new KeyAndVersion("objectkey3"));
request.setKeyAndVersions(toDelete.toArray(new KeyAndVersion[toDelete.size()]));

DeleteObjectsResult result = obsClient.deleteObjects(request);
// 获取删除成功的对象
System.out.println(result.getDeletedObjectResults());
// 获取删除失败的对象
System.out.println(result.getErrorResults());
```

```
// 关闭obsClient
obsClient.close();
```

9.5 复制对象

复制对象特性用来为OBS上已经存在的对象创建一个副本。

您可以通过ObsClient.copyObject来复制对象。复制对象时，可重新指定新对象的属性和设置对象权限，且支持条件复制。

简单复制

以下代码展示了如何进行简单复制：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectResult result = obsClient.copyObject("sourcebucketname", "sourceobjectkey",
"destbucketname", "destobjectkey");
System.out.println("\t" + result.getEtag());

// 关闭obsClient
obsClient.close();
```

重写对象属性

以下代码展示了如何在复制对象时重写对象属性：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectRequest request = new CopyObjectRequest("sourcebucketname", "sourceobjectkey",
"destbucketname", "destobjectkey");
// 设置进行对象属性重写
request.setReplaceMetadata(true);
ObjectMetadata newObjectMetadata = new ObjectMetadata();
newObjectMetadata.setContentType("image/jpeg");
newObjectMetadata.addUserMetadata("property", "property-value");
newObjectMetadata.setStorageClass(S3Bucket.STANDARD);
request.setNewObjectMetadata(newObjectMetadata);
CopyObjectResult result = obsClient.copyObject(request);
System.out.println("\t" + result.getEtag());

// 关闭obsClient
obsClient.close();
```

说明

CopyObjectRequest.setReplaceMetadata需与CopyObjectRequest.setNewObjectMetadata配合使用。

限定条件复制

复制对象时，可以指定一个或多个限定条件，满足限定条件时则进行复制，否则抛出异常，复制对象失败。

您可以使用的限定条件如下：

参数	作用	OBS Java SDK对应方法
Copy-Source-If-Modified-Since	如果源对象的修改时间晚于该参数值指定的时间，则进行复制，否则抛出异常。	CopyObjectRequest.setIfModifiedSince
Copy-Source-If-Unmodified-Since	如果源对象的修改时间早于该参数值指定的时间，则进行复制，否则抛出异常。	CopyObjectRequest.setIfUnmodifiedSince
Copy-Source-If-Match	如果源对象的ETag值与该参数值相同，则进行复制，否则抛出异常。	CopyObjectRequest.setIfMatchTag
Copy-Source-If-None-Match	如果源对象的ETag值与该参数值不相同，则进行复制，否则抛出异常。	CopyObjectRequest.setIfNoneMatchTag

说明

- 源对象的ETag值是指源对象数据的MD5校验值。
- 如果包含Copy-Source-if-Unmodified-Since并且不符合，或者包含Copy-Source-If-Match并且不符合，或者包含Copy-Source-if-Modified-Since并且不符合，或者包含Copy-Source-if-None-Match并且不符合，抛出异常中HTTP状态码为：412 precondition failed。
- Copy-Source-If-Modified-Since和Copy-Source-If-None-Match可以一起使用；Copy-Source-If-Unmodified-Since和Copy-Source-If-Match可以一起使用。

以下代码展示了如何进行限定条件复制：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectRequest request = new CopyObjectRequest("sourcebucketname", "sourceobjectkey",
"destbucketname", "destobjectkey");

request.setIfModifiedSince(new SimpleDateFormat("yyyy-MM-dd").parse("2016-01-01"));
request.setIfNoneMatchTag("none-match-etag");

CopyObjectResult result = obsClient.copyObject(request);
System.out.println("\t" + result.getEtag());

// 关闭obsClient
obsClient.close();
```

重写对象访问权限

以下代码展示了如何在复制对象时重写对象访问权限：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
```

```
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectRequest request = new CopyObjectRequest("sourcebucketname", "sourceobjectkey",
"destbucketname", "destobjectkey");

// 复制时重写对象访问权限为公共读
request.setAcl(AccessControlList.REST_CANNED_PUBLIC_READ);
CopyObjectResult result = obsClient.copyObject(request);
System.out.println("\t" + result.getEtag());

// 关闭obsClient
obsClient.close();
```

10 临时授权访问

10.1 什么是临时授权访问

临时授权访问是指通过访问密钥、请求方法类型、请求参数等信息生成一个临时访问权限的URL，这个URL中会包含鉴权信息，您可以使用该URL进行访问OBS服务进行特定操作。在生成URL时，您需要指定URL的有效期。

临时授权访问支持的操作以及相关信息见下表：

操作名	HTTP请求方法（OBS Java SDK对应值）	特殊操作符（OBS Java SDK对应值）	是否需要桶名	是否需要对象名
创建桶	HttpMethodEnum.PUT	N/A	是	否
获取桶列表	HttpMethodEnum.GET	N/A	否	否
删除桶	HttpMethodEnum.DELETE	N/A	是	否
列举桶内对象	HttpMethodEnum.GET	N/A	是	否
列举桶内多版本对象	HttpMethodEnum.GET	SpecialParamEnum.VERSIONS	是	否
列举分段上传任务	HttpMethodEnum.GET	SpecialParamEnum.UPLOADS	是	否
获取桶元数据	HttpMethodEnum.HEAD	N/A	是	否

操作名	HTTP请求方法（OBS Java SDK对应值）	特殊操作符（OBS Java SDK对应值）	是否需要桶名	是否需要对象名
获取桶区域位置	HttpMethodEnum.GET	SpecialParamEnum.LOCATION	是	否
获取桶存量信息	HttpMethodEnum.GET	SpecialParamEnum.STORAGEINFO	是	否
设置桶配额	HttpMethodEnum.PUT	SpecialParamEnum.QUOTA	是	否
获取桶配额	HttpMethodEnum.GET	SpecialParamEnum.QUOTA	是	否
设置桶存储类型	HttpMethodEnum.PUT	SpecialParamEnum.STORAGEPOLICY	是	否
获取桶存储类型	HttpMethodEnum.GET	SpecialParamEnum.STORAGEPOLICY	是	否
设置桶访问权限	HttpMethodEnum.PUT	SpecialParamEnum.ACL	是	否
获取桶访问权限	HttpMethodEnum.GET	SpecialParamEnum.ACL	是	否
开启/关闭桶日志	HttpMethodEnum.PUT	SpecialParamEnum.LOGGING	是	否
查看桶日志	HttpMethodEnum.GET	SpecialParamEnum.LOGGING	是	否
设置桶策略	HttpMethodEnum.PUT	SpecialParamEnum.POLICY	是	否
查看桶策略	HttpMethodEnum.GET	SpecialParamEnum.POLICY	是	否
删除桶策略	HttpMethodEnum.DELETE	SpecialParamEnum.POLICY	是	否
设置生命周期规则	HttpMethodEnum.PUT	SpecialParamEnum.LIFECYCLE	是	否
查看生命周期规则	HttpMethodEnum.GET	SpecialParamEnum.LIFECYCLE	是	否

操作名	HTTP请求方法（OBS Java SDK对应值）	特殊操作符（OBS Java SDK对应值）	是否需要桶名	是否需要对象名
删除生命周期规则	HttpMethodEnum.DELETE	SpecialParamEnum.LIFECYCLE	是	否
设置托管配置	HttpMethodEnum.PUT	SpecialParamEnum.WEBSITE	是	否
查看托管配置	HttpMethodEnum.GET	SpecialParamEnum.WEBSITE	是	否
清除托管配置	HttpMethodEnum.DELETE	SpecialParamEnum.WEBSITE	是	否
设置桶多版本状态	HttpMethodEnum.PUT	SpecialParamEnum.VERSIONING	是	否
查看桶多版本状态	HttpMethodEnum.GET	SpecialParamEnum.VERSIONING	是	否
设置跨域规则	HttpMethodEnum.PUT	SpecialParamEnum.CORS	是	否
查看跨域规则	HttpMethodEnum.GET	SpecialParamEnum.CORS	是	否
删除跨域规则	HttpMethodEnum.DELETE	SpecialParamEnum.CORS	是	否
设置/关闭事件通知	HttpMethodEnum.PUT	SpecialParamEnum.NOTIFICATION	是	否
查看事件通知	HttpMethodEnum.GET	SpecialParamEnum.NOTIFICATION	是	否
OPTIONS桶	HttpMethodEnum.OPTIONS	N/A	是	否
设置桶标签	HttpMethodEnum.PUT	SpecialParamEnum.TAGGING	是	否
查看桶标签	HttpMethodEnum.GET	SpecialParamEnum.TAGGING	是	否
删除桶标签	HttpMethodEnum.DELETE	SpecialParamEnum.TAGGING	是	否
上传对象	HttpMethodEnum.PUT	N/A	是	是
下载对象	HttpMethodEnum.GET	N/A	是	是

操作名	HTTP请求方法（OBS Java SDK对应值）	特殊操作符（OBS Java SDK对应值）	是否需要桶名	是否需要对象名
复制对象	HttpMethodEnum.PUT	N/A	是	是
删除对象	HttpMethodEnum.DELETE	N/A	是	是
批量删除对象	HttpMethodEnum.POST	SpecialParamEnum.DELETE	是	是
获取对象属性	HttpMethodEnum.GET	N/A	是	是
设置对象访问权限	HttpMethodEnum.PUT	SpecialParamEnum.ACL	是	是
查看对象访问权限	HttpMethodEnum.GET	SpecialParamEnum.ACL	是	是
初始化分段上传任务	HttpMethodEnum.POST	SpecialParamEnum.UPLOADS	是	是
上传段	HttpMethodEnum.PUT	N/A	是	是
复制段	HttpMethodEnum.PUT	N/A	是	是
列举已上传的段	HttpMethodEnum.GET	N/A	是	是
合并段	HttpMethodEnum.POST	N/A	是	是
取消分段上传任务	HttpMethodEnum.DELETE	N/A	是	是
OPTIONS对象	HttpMethodEnum.OPTIONS	N/A	是	是
取回归档存储对象	HttpMethodEnum.POST	SpecialParamEnum.RESTORE	是	是

临时授权访问分为V2和V4两种方式，分别基于V2临时授权请求鉴权和V4临时授权请求鉴权。

10.2 临时授权访问

您可以通过ObsClient.createSignedUrl生成临时授权访问的URL。以下代码展示了如何生成常用操作的URL，包括：创建桶、上传对象、下载对象、列举对象、删除对象。

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// URL有效期，3600秒
long expireSeconds = 60 * 60;
Date expireDate = new Date(System.currentTimeMillis() + expireSeconds * 1000);

// 创建桶
String createBucketUrl = obsClient.createSignedUrl(HttpMethodEnum.PUT, "bucketname", null, null,
expireDate, null, null);
System.out.println("\t" + createBucketUrl);

// 上传对象
Map<String, String> headers = new HashMap<String, String>();
headers.put("Content-Type", "text/plain");
String putObjectUrl = obsClient.createSignedUrl(HttpMethodEnum.PUT, "bucketname", "objectkey",
null, expireDate, headers, null);
System.out.println("\t" + putObjectUrl);

// 下载对象
String getObjectUrl = obsClient.createSignedUrl(HttpMethodEnum.GET, "bucketname", "objectkey",
null, expireDate, null, null);
System.out.println("\t" + getObjectUrl);

// 列举对象
String listObjectsUrl = obsClient.createSignedUrl(HttpMethodEnum.GET, "bucketname", null, null,
expireDate, null, null);
System.out.println("\t" + listObjectsUrl);

// 删除对象
String deleteObjectUrl = obsClient.createSignedUrl(HttpMethodEnum.DELETE, "bucketname",
"objectkey", null, expireDate, null, null);
System.out.println("\t" + deleteObjectUrl);

// 关闭obsClient
obsClient.close();
```

说明

- HttpMethodEnum是OBS Java SDK定义的枚举类型，代表请求方法类型。
- 完整的临时授权访问代码示例参见V2TemporarySignatureSample.zip。

10.3 V4 临时授权访问

您可以通过ObsClient.createV4TemporarySignature生成V4临时授权访问的URL。以下代码展示了如何生成常用操作的URL，包括：创建桶、上传对象、下载对象、列举对象、删除对象。

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// URL有效期，3600秒
long expireSeconds = 60 * 60;

// 创建桶
V4TemporarySignatureRequest v4Request = new V4TemporarySignatureRequest(HttpMethodEnum.PUT,
expireSeconds);
v4Request.setBucketName("bucketname");
V4TemporarySignatureResponse createBucket = obsClient.createV4TemporarySignature(v4Request);
System.out.println("\t" + createBucket.getSignedUrl());
```

```
// 上传对象
Map<String, Object> headers = new HashMap<String, Object>();
headers.put("Content-Type", "text/plain");
v4Request = new V4TemporarySignatureRequest(HttpMethodEnum.PUT, expireSeconds);
v4Request.setBucketName("bucketname");
v4Request.setObjectKey("objectkey");
v4Request.setHeaders(headers);
V4TemporarySignatureResponse putObjectResponse = obsClient.createV4TemporarySignature(v4Request);
System.out.println("\t" + putObjectResponse.getSignedUrl());

// 下载对象
v4Request = new V4TemporarySignatureRequest(HttpMethodEnum.GET, expireSeconds);
v4Request.setBucketName("bucketname");
v4Request.setObjectKey("objectkey");
V4TemporarySignatureResponse getObjectResponse = obsClient.createV4TemporarySignature(v4Request);
System.out.println("\t" + getObjectResponse.getSignedUrl());

// 列举对象
v4Request = new V4TemporarySignatureRequest(HttpMethodEnum.GET, expireSeconds);
v4Request.setBucketName("bucketname");
V4TemporarySignatureResponse listObjectResponse = obsClient.createV4TemporarySignature(v4Request);
System.out.println("\t" + listObjectResponse.getSignedUrl());

// 删除对象
v4Request = new V4TemporarySignatureRequest(HttpMethodEnum.DELETE, expireSeconds);
v4Request.setBucketName("bucketname");
v4Request.setObjectKey("objectkey");
V4TemporarySignatureResponse deleteObjectResponse =
obsClient.createV4TemporarySignature(v4Request);
System.out.println("\t" + deleteObjectResponse.getSignedUrl());

// 关闭obsClient
obsClient.close();
```

说明

完整的V4临时授权访问代码示例参见V4TemporarySignatureSample.zip。

11 多版本控制

OBS支持保存一个对象的多个版本，使您更方便地检索和还原各个版本，在意外操作或应用程序故障时快速恢复数据。

更多关于多版本控制的内容请参见**多版本控制**。

11.1 设置桶多版本状态

您可以通过ObsClient.setBucketVersioning设置桶的多版本状态。OBS中的桶支持两种多版本状态：

多版本状态	说明	OBS Java SDK对应值
启用状态	1. 上传对象时，系统为每一个对象创建一个唯一版本号，上传同名的对象将不再覆盖旧的对象，而是创建新的不同版本号的同名对象。 2. 可以指定版本号下载对象，不指定版本号默认下载最新对象。 3. 删除对象时可以指定版本号删除，不带版本号删除对象仅产生一个带唯一版本号的删除标记，并不删除对象。 4. 列出桶内对象列表（ObsClient.listObjects）时默认列出最新对象列表，可以指定列出桶内所有版本对象列表（ObsClient.listVersions）。 5. 除了删除标记外，每个版本的对象存储均需计费。	BucketVersioningConfiguration.ENABLED
暂停状态	1. 旧的版本数据继续保留。 2. 上传对象时创建对象的版本号为null，上传同名的对象将覆盖原有同名的版本号为null的对象。 3. 可以指定版本号下载对象，不指定版本号默认下载最新对象。 4. 删除对象时可以指定版本号删除，不带版本号删除对象将产生一个版本号为null的删除标记，并删除版本号为null的对象。 5. 除了删除标记外，每个版本的对象存储均需计费。	BucketVersioningConfiguration.SUSPENDED

以下代码展示了如何设置桶的多版本状态：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 启用桶多版本状态
obsClient.setBucketVersioning("bucketname", BucketVersioningConfiguration.ENABLED);

// 暂停桶多版本状态
obsClient.setBucketVersioning("bucketname", BucketVersioningConfiguration.SUSPENDED);
```

```
// 关闭obsClient
obsClient.close();
```

11.2 查看桶多版本状态

您可以通过ObsClient.getBucketVersioning查看桶的多版本状态。以下代码展示了如何查看桶的多版本状态：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketVersioningConfiguration status = obsClient.getBucketVersioning("bucketname");
System.out.println("\t" + status.getStatus());

// 关闭obsClient
obsClient.close();
```

11.3 获取多版本对象

您可以通过ObsClient.getObject接口传入版本号（versionId）来获取多版本对象，示例代码如下：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 设置versionId获取多版本对象
S3Object s3Object = obsClient.getObject("bucketname", "objectkey", "versionid");
s3Object.getObjectContent().close();

// 关闭obsClient
obsClient.close();
```

说明

如果版本号为空则默认下载最新版本的对象。

11.4 复制多版本对象

您可以通过ObsClient.copyObject接口传入版本号（versionId）来复制多版本对象，示例代码如下：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectRequest request = new CopyObjectRequest();
request.setSourceBucketName("sourebucketname");
request.setSourceObjectKey("sourceobjectkey");
// 设置要复制对象的版本号
request.setVersionId("versionid");
request.setDestinationBucketName("destbucketname");
request.setDestinationObjectKey("destobjectkey");
obsClient.copyObject(request);
```

```
// 关闭obsClient
obsClient.close();
```

11.5 取回多版本归档存储对象

您可以通过ObsClient.restoreObject接口传入版本号（versionId）来取回多版本归档存储对象，示例代码如下：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

RestoreObjectRequest request = new RestoreObjectRequest("bucketname", "objectkey", 1);
request.setVersionId("versionid");
obsClient.restoreObject(request);

// 关闭obsClient
obsClient.close();
```

11.6 列举多版本对象

您可以通过ObsClient.listVersions列举多版本对象。

该接口可设置的参数如下：

参数	作用
bucketName	桶名。
prefix	限定返回的对象名必须带有prefix前缀。
keyMarker	列举多版本对象的起始位置，返回的对象列表将是对象名按照字典序排序后该参数以后的所有对象。
maxKeys	列举多版本对象的最大数目，取值范围为1~1000，当超出范围时，按照默认的1000进行处理。
delimiter	用于对对象名进行分组的字符。对于对象名中包含delimiter的对象，其对象名（如果请求中指定了prefix，则此处的对象名需要去掉prefix）中从首字符至第一个delimiter之间的字符串将作为一个分组并作为commonPrefix返回。
versionIdMarker	与keyMarker配合使用，返回的对象列表将是对象名和版本号按照字典序排序后该参数以后的所有对象。

说明

- 如果versionIdMarker不是keyMarker的一个版本号，则该参数无效。
- ObsClient.listVersions返回结果包含多版本对象和对象删除标记。

简单列举

以下代码展示如何简单列举多版本对象，最多返回1000个对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsResult result = obsClient.listVersions("bucketname");

for(VersionOrDeleteMarker v : result.getVersions()){
    System.out.println("\t" + v.getKey());
    System.out.println("\t" + v.getOwner());
    System.out.println("\t" + v.isDeleteMarker());
}

// 关闭obsClient
obsClient.close();
```

说明

- 每次至多返回1000个多版本对象，如果指定桶包含的对象数量大于1000，则返回结果中ListVersionsResult.isTruncated为true表明本次没有返回全部对象，并可通过ListVersionsResult .getNextKeyMarker和ListVersionsResult.getNextVersionIdMarker获取下次列举的起始位置。
- 如果想获取指定桶包含的所有多版本对象，可以采用分页列举的方式。

指定数目列举

以下代码展示如何指定数目列举多版本对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsResult result = obsClient.listVersions("bucketname", 100);
for(VersionOrDeleteMarker v : result.getVersions()){
    System.out.println("\t" + v.getKey());
    System.out.println("\t" + v.getOwner());
    System.out.println("\t" + v.isDeleteMarker());
}

// 关闭obsClient
obsClient.close();
```

指定前缀列举

以下代码展示如何指定前缀列举多版本对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 设置列举带有prefix前缀的100个多版本对象
```

```
ListVersionsResult result = obsClient.listVersions("bucketname", "prefix", null, null, null, 100);
for(VersionOrDeleteMarker v : result.getVersions()){
    System.out.println("\t" + v.getKey());
    System.out.println("\t" + v.getOwner());
    System.out.println("\t" + v.isDeleteMarker());
}

// 关闭obsClient
obsClient.close();
```

指定起始位置列举

以下代码展示如何指定起始位置列举多版本对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 设置列举对象名字典序在"test"之后的100个多版本对象
ListVersionsResult result = obsClient.listVersions("bucketname", null, null, "test", null, 100);

for(VersionOrDeleteMarker v : result.getVersions()){
    System.out.println("\t" + v.getKey());
    System.out.println("\t" + v.getOwner());
    System.out.println("\t" + v.isDeleteMarker());
}

// 关闭obsClient
obsClient.close();
```

分页列举全部多版本对象

以下代码展示分页列举全部多版本对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsResult result;
String keyMarker = null;
String versionIdMarker = null;

do{
    result = obsClient.listVersions("bucketname", null, null, keyMarker, versionIdMarker, 100);
    for(VersionOrDeleteMarker v : result.getVersions()){
        System.out.println("\t" + v.getKey());
        System.out.println("\t" + v.getOwner());
        System.out.println("\t" + v.isDeleteMarker());
    }

    keyMarker = result.getKeyMarker();
    versionIdMarker = result.getNextVersionIdMarker();

}while(result.isTruncated());

// 关闭obsClient
obsClient.close();
```

列举文件夹中的所有多版本对象

OBS本身是没有文件夹的概念的，桶中存储的元素只有对象。文件夹对象实际上是一个大小为0且对象名以“/”结尾的对象，将这个文件夹对象名作为前缀，即可模拟列举文件夹中对象的功能。以下代码展示如何列举文件夹中的多版本对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsResult result;
String keyMarker = null;
String versionIdMarker = null;

do{
    // 设置文件夹对象名"dir/"为前缀
    result = obsClient.listVersions("bucketname", "dir/", null, keyMarker, versionIdMarker, 1000);
    for(VersionOrDeleteMarker v : result.getVersions()){
        System.out.println("\t" + v.getKey());
        System.out.println("\t" + v.getOwner());
        System.out.println("\t" + v.isDeleteMarker());
    }
    keyMarker = result.getKeyMarker();
    versionIdMarker = result.getNextVersionIdMarker();

}while(result.isTruncated());

// 关闭obsClient
obsClient.close();
```

按文件夹分组列举所有多版本对象

以下代码展示如何按文件夹分组，列举桶内所有多版本对象：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsResult result = obsClient.listVersions("bucketname", null, "/", null, null, 1000);
System.out.println("根目录中的对象：");
for(VersionOrDeleteMarker v : result.getVersions()){
    System.out.println("\t" + v.getKey());
    System.out.println("\t" + v.getOwner());
    System.out.println("\t" + v.isDeleteMarker());
}

listVersionsByPrefix(obsClient, result);

// 关闭obsClient
obsClient.close();
```

递归列出子文件夹中多版本对象的函数*listVersionsByPrefix*的示例代码如下：

```
static void listVersionsByPrefix(ObsClient obsClient, ListVersionsResult result) throws
ObsException{
    for(String prefix : result.getCommonPrefixes()){
         System.out.println("文件夹 [" + prefix + "]中的对象:");
         result = obsClient.listVersions("bucketname", prefix, "/", null, null, 1000);
         for(VersionOrDeleteMarker v : result.getVersions()){
             System.out.println("\t" + v.getKey());
             System.out.println("\t" + v.getOwner());
             System.out.println("\t" + v.isDeleteMarker());
         }
         listVersionsByPrefix(obsClient, result);
     }
}
```

说明

- 以上代码示例没有考虑文件夹中多版本对象数超过1000个的情况。
- 由于是需要列举出文件夹中的对象和子文件夹，且文件夹对象总是以“/”结尾，因此delimiter总是为“/”。
- 每次递归的返回结果中ListVersionsResult.getVersions包含的是文件夹中的多版本对象；ListVersionsResult.getCommonPrefixes包含的是文件夹的子文件夹。

11.7 多版本对象权限

设置多版本对象访问权限

您可以通过ObsClient.setObjectAcl接口传入版本号（versionId）设置多版本对象的访问权限，示例代码如下：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 通过预定义访问策略设置多版本对象访问权限为公共读
obsClient.setObjectAcl("bucketname", "objectkey", null, AccessControlList.REST_CANNED_PUBLIC_READ,
"versionid");

AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
owner.setId("ownerid");
acl.setOwner(owner);
// 为所有用户设置读权限
acl.grantPermission(GroupGrantee.ALL_USERS, Permission.PERMISSION_READ);
// 为授权用户设置写权限
acl.grantPermission(GroupGrantee.AUTHENTICATED_USERS, Permission.PERMISSION_WRITE);
// 直接设置多版本对象访问权限
obsClient.setObjectAcl("bucketname", "objectkey", null, null, "versionid");

// 关闭obsClient
obsClient.close();
```

说明

ACL中需要填写的所有者（Owner）或者被授权用户（Grantee）的ID，是指用户的帐户ID，可通过OBS控制台“我的凭证”页面查看。

获取多版本对象访问权限

您可以通过ObsClient.getObjectAcl接口传入版本号（versionId）获取多版本对象的访问权限，示例代码如下：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AccessControlList acl = obsClient.getObjectAcl("bucketname", "objectkey", "versionid");
System.out.println(acl);

// 关闭obsClient
obsClient.close();
```

11.8 删除多版本对象

删除单个多版本对象

您可以通过ObsClient.deleteObject接口传入版本号（versionId）删除多版本对象，示例代码如下：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
obsClient.deleteObject("bucketname", "objectkey", "versionid");

// 关闭obsClient
obsClient.close();
```

批量删除多版本对象

您可以通过ObsClient.deleteObjects接口传入每个待删除对象的版本号（versionId）批量删除多版本对象，示例代码如下：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

DeleteObjectsRequest request = new DeleteObjectsRequest("bucketname");
request.setQuiet(false);
List<KeyAndVersion> toDelete = new ArrayList<KeyAndVersion>();
toDelete.add(new KeyAndVersion("objectkey1", "versionid1"));
toDelete.add(new KeyAndVersion("objectkey2", "versionid2"));
toDelete.add(new KeyAndVersion("objectkey3", "versionid3"));
request.setKeyAndVersions(toDelete.toArray(new KeyAndVersion[toDelete.size()]));
DeleteObjectsResult result = obsClient.deleteObjects(request);

System.out.println("\t" + result.getDeletedObjectResults());
System.out.println("\t" + result.getErrorResults());

// 关闭obsClient
obsClient.close();
```

12 生命周期管理

OBS允许您对桶设置生命周期规则，以自动淘汰过期的对象，节省存储空间。针对不同前缀的对象，您可以同时设置多条规则。一条规则包含：

- 规则ID，用于标识一条规则，不能重复。
- 受影响的对象前缀，此规则只作用于符合前缀的对象。
- 最新版本对象过期时间，指定方式为：
  a. 指定满足前缀的对象创建后第几天时过期。
  b. 直接指定满足前缀的对象过期日期。
- 历史版本对象过期时间，指定方式为：
  a. 指定满足前缀的对象成为历史版本后第几天时过期。
- 是否生效标识。

更多关于生命周期的内容请参考**生命周期管理**。

说明

- 对象过期后会被OBS服务端自动删除。
- 桶必须开启多版本状态，历史版本对象过期时间配置才能生效。

12.1 设置生命周期规则

您可以通过ObsClient.setBucketLifecycleConfiguration设置桶的生命周期规则。以下代码展示了如何设置桶的生命周期规则：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

LifecycleConfiguration config = new LifecycleConfiguration();
```

```
Rule rule = config.new Rule();
rule.setEnabled(true);
rule.setId("rule1");
rule.setPrefix("prefix");
Expiration expiration = config.new Expiration();
// 指定满足前缀的对象创建10天后过期
expiration.setDays(10);
// 直接指定满足前缀的对象过期时间
// expiration.setDate(new SimpleDateFormat("yyyy-MM-dd").parse("2018-12-31"));
rule.setExpiration(expiration);

NoncurrentVersionExpiration noncurrentVersionExpiration = config.new NoncurrentVersionExpiration();
// 指定满足前缀的对象成为历史版本20天后过期
noncurrentVersionExpiration.setDays(20);
rule.setNoncurrentVersionExpiration(noncurrentVersionExpiration);
config.addRule(rule);

obsClient.setBucketLifecycleConfiguration("bucketname", config);

// 关闭obsClient
obsClient.close();
```

12.2 查看生命周期规则

您可以通过ObsClient.getBucketLifecycleConfiguration查看桶的生命周期规则。以下代码展示了如何查看桶的生命周期规则：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

LifecycleConfiguration config = obsClient.getBucketLifecycleConfiguration("bucketname");

for (Rule rule : config.getRules()) {
    System.out.println(rule.getId());
    System.out.println(rule.getPrefix());
    System.out.println(rule.getExpiration() != null ? rule.getExpiration().getDays() : "");
    System.out.println(rule.getNoncurrentVersionExpiration() != null ?
rule.getNoncurrentVersionExpiration().getDays() : "");
}

// 关闭obsClient
obsClient.close();
```

12.3 删除生命周期规则

您可以通过ObsClient.deleteBucketLifecycleConfiguration删除桶的生命周期规则。以下代码展示了如何删除桶的生命周期规则：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.deleteBucketLifecycleConfiguration("bucketname");

// 关闭obsClient
obsClient.close();
```

13 跨域资源共享

跨域资源共享（CORS）允许Web端的应用程序访问不属于本域的资源。OBS提供接口方便开发者控制跨域访问的权限。

更多关于跨域资源共享的内容请参考**跨域资源访问**。

13.1 设置跨域规则

您可以通过ObsClient.setBucketCors设置桶的跨域规则，如果原规则存在则覆盖原规则。以下代码展示了如何设置跨域规则：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

S3BucketCors cors = new S3BucketCors();

List<BucketCorsRule> rules = new ArrayList<BucketCorsRule>();
BucketCorsRule rule = new BucketCorsRule();

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

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

ArrayList<String> allowedHeader = new ArrayList<String>();
// 控制在OPTIONS预取指令中Access-Control-Request-Headers头中指定的header是否被允许使用
allowedHeader.add("x-obs-header");
rule.setAllowedHeader(allowedHeader);
```

```
ArrayList<String> exposeHeader = new ArrayList<String>();
// 指定允许用户从应用程序中访问的header
exposeHeader.add("x-obs-expose-header");
rule.setExposeHeader(exposeHeader);

// 指定浏览器对特定资源的预取(OPTIONS)请求返回结果的缓存时间,单位为秒
rule.setMaxAgeSecond(10);
rules.add(rule);
cors.setRules(rules);

obsClient.setBucketCors("bucketname", cors);

// 关闭obsClient
obsClient.close();
```

说明

AllowedOrigins、AllowedMethods、AllowedHeaders都能够最多支持一个“*”通配符。“*”表示对于所有的域来源、操作或者头域都满足。

13.2 查看跨域规则

您可以通过ObsClient.getBucketCors查看桶的跨域规则。以下代码展示了如何查看跨域规则：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);


S3BucketCors cors = obsClient.getBucketCors("bucketname");
for(BucketCorsRule rule : cors.getRules()){
    System.out.println("\t" + rule.getId());
    System.out.println("\t" + rule.getMaxAgeSecond());
    System.out.println("\t" + rule.getAllowedHeader());
    System.out.println("\t" + rule.getAllowedOrigin());
    System.out.println("\t" + rule.getAllowedMethod());
    System.out.println("\t" + rule.getExposeHeader());
}

// 关闭obsClient
obsClient.close();
```

13.3 删除跨域规则

您可以通过ObsClient.deleteBucketCors删除桶的跨域规则。以下代码展示了如何删除跨域规则：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.deleteBucketCors("bucketname");

// 关闭obsClient
obsClient.close();
```

14 设置访问日志

OBS允许您对桶设置访问日志记录，设置之后对于桶的访问会被记录成日志，日志存储在OBS上您指定的目标桶中。

更多关于访问日志的内容请参考**日志记录**。

14.1 开启桶日志

您可以通过ObsClient.setBucketLoggingConfiguration开启桶日志功能。开启桶日志功能需要两个步骤：

步骤1 设置目标桶的日志投递组访问权限（目标桶必须保证日志投递用户组有写和读ACP权限）。

步骤2 设置桶的日志管理配置。

----结束

说明

- 日志目标桶与源桶必须在同一个区域（region）。
- 若桶的存储类型为低频访问存储或归档存储，则不能作为日志目标桶。

直接开启桶日志

以下代码展示了如何直接开启桶日志：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketLoggingConfiguration config = new BucketLoggingConfiguration();

config.setTargetBucketName("targetbucketname");
config.setLogfilePrefix("targetprefix");
```

```
// 设置桶的日志配置，第三个参数为true时表明自动配置目标桶的日志投递组权限
obsClient.setBucketLoggingConfiguration("bucketname", config, true);

// 关闭obsClient
obsClient.close();
```

说明

ObsClient.setBucketLoggingConfiguration第三个参数表示是否由OBS Java SDK自动设置目标桶的日志投递组写权限和读ACP权限。

分步开启桶日志

以下代码展示了如何分步开启桶日志：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

String targetBucket = "targetbucketname";

// 设置目标桶日志投递组用户写权限和读ACP权限
obsClient.setBucketAcl(targetBucket, AccessControlList.REST_CANNED_LOG_DELIVERY_WRITE);

// 设置桶的日志配置
BucketLoggingConfiguration config = new BucketLoggingConfiguration();
config.setTargetBucketName(targetBucket);
config.setLogfilePrefix("prefix");

obsClient.setBucketLoggingConfiguration("bucketname", config);

// 关闭obsClient
obsClient.close();
```

为日志对象设置权限

以下代码展示了如何为日志对象设置权限：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

String targetBucket = "targetbucketname";

// 设置桶的日志配置
BucketLoggingConfiguration config = new BucketLoggingConfiguration();
config.setTargetBucketName(targetBucket);
config.setLogfilePrefix("prefix");

// 为所有用户设置对日志对象的读权限
GrantAndPermission grant1 = new GrantAndPermission(GroupGrantee.ALL_USERS,
Permission.PERMISSION_READ);
// 为OBS授权用户设置对日志文件的全部权限
GrantAndPermission grant2 = new GrantAndPermission(GroupGrantee.AUTHENTICATED_USERS,
Permission.PERMISSION_FULL_CONTROL);
config.setTargetGrants(new GrantAndPermission[]{grant1, grant2});

obsClient.setBucketLoggingConfiguration("bucketname", config, true);

// 关闭obsClient
obsClient.close();
```

14.2 查看桶日志

您可以通过ObsClient.getBucketLoggingConfiguration查看桶日志配置。以下代码展示了如何查看桶日志配置：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketLoggingConfiguration config = obsClient.getBucketLoggingConfiguration("bucketname");
System.out.println("\t" + config.getTargetBucketName());
System.out.println("\t" + config.getLogfilePrefix());

// 关闭obsClient
obsClient.close();
```

14.3 关闭桶日志

关闭桶日志功能实际上就是调用ObsClient.setBucketLoggingConfiguration将日志配置清空，以下代码展示了如何关闭桶日志：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 对桶设置空的日志配置
obsClient.setBucketLoggingConfiguration("bucketname", new BucketLoggingConfiguration());

// 关闭obsClient
obsClient.close();
```

15 静态网站托管

您可以将静态网站文件上传至OBS的桶中作为对象，并对这些对象赋予公共读权限，然后将该桶配置成静态网站托管模式，以实现在OBS上托管静态网站的目的。第三方用户在访问您网站的时候，实际上是在访问OBS的桶中的对象。在使用静态网站托管功能时，OBS还支持配置请求重定向，通过重定向配置您可以将特定的请求或所有请求实施重定向。

更多关于静态网站托管的内容请参考**静态网站托管**。

15.1 网站文件托管

您可通过以下步骤实现网站文件托管：

步骤1 将网站文件上传至OBS的桶中，并设置对象MIME类型。

步骤2 设置对象访问权限为公共读。

步骤3 通过浏览器访问对象。

----结束

以下代码展示了如何实现网站文件托管：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// 上传对象，设置对象MIME类型
PutObjectRequest request = new PutObjectRequest();
request.setBucketName("bucketname");
request.setObjectKey("test.html");
request.setFile(new File("localfile.html"));
ObjectMetadata metadata = new ObjectMetadata();
```

```
metadata.setContentType("text/html");
request.setMetadata(metadata);
obsClient.putObject(request);

// 设置对象访问权限为公共读
obsClient.setObjectAcl("bucketname", "test.html", AccessControlList.REST_CANNED_PUBLIC_READ);

obsClient.close();
```

说明

上例中您可以使用两种方式在浏览器直接访问托管的文件

1. http://obs.myhwclouds.com/bucketname/test.html（路径访问方式）
2. http://bucketname.obs.myhwclouds.com/test.html （虚拟主机访问方式）

15.2 设置托管配置

您可以通过ObsClient.setBucketWebsiteConfiguration设置桶的托管配置。

配置默认主页和错误页面

以下代码展示了如何配置默认主页和错误页面：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

WebsiteConfiguration config = new WebsiteConfiguration();
// 配置默认主页
config.setSuffix("index.html");
// 配置错误页面
config.setKey("error.html");
obsClient.setBucketWebsiteConfiguration("bucketname", config);

obsClient.close();
```

配置重定向规则

以下代码展示了如何配置重定向规则：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

WebsiteConfiguration config = new WebsiteConfiguration();
// 配置默认主页
config.setSuffix("index.html");
// 配置错误页面
config.setKey("error.html");

RouteRule rule = new RouteRule();
Redirect r = new Redirect();
r.setHostName("www.example.com");
r.setHttpRedirectCode("305");
r.setProtocol("http");
r.setReplaceKeyPrefixWith("replacekeyprefix");
rule.setRedirect(r);
RouteRuleCondition condition = new RouteRuleCondition();
condition.setHttpErrorCodeReturnedEquals("404");
condition.setKeyPrefixEquals("keyprefix");
rule.setCondition(condition);
```

```
config.getRouteRules().add(rule);

obsClient.setBucketWebsiteConfiguration("bucketname", config);

obsClient.close();
```

配置所有请求重定向

以下代码展示了如何配置所有请求重定向：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

WebsiteConfiguration config = new WebsiteConfiguration();
RedirectAllRequest redirectAll = new RedirectAllRequest();
redirectAll.setHostName("www.example.com");
// 支持http和https
redirectAll.setProtocol("http");
config.setRedirectAllRequestsTo(redirectAll);

obsClient.setBucketWebsiteConfiguration("bucketname", config);

obsClient.close();
```

15.3 查看托管配置

您可以通过ObsClient.getBucketWebsiteConfiguration查看桶的托管配置。以下代码展示了如何查看托管配置：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

WebsiteConfiguration config = obsClient.getBucketWebsiteConfiguration("bucketname");
System.out.println("\t" + config.getKey());
System.out.println("\t" + config.getSuffix());
for(RouteRule rule : config.getRouteRules()){
    System.out.println("\t" +rule);
}

obsClient.close();
```

15.4 清除托管配置

您可以通过ObsClient.deleteBucketWebsiteConfiguration清除桶的托管配置。以下代码展示了如何清除托管配置：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.deleteBucketWebsiteConfiguration("bucketname");

obsClient.close();
```

16 标签管理

16.1 设置桶标签

您可以通过ObsClient.setBucketTagging设置桶标签。以下代码展示了如何设置桶标签：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketTagInfo bucketTagInfo = new BucketTagInfo();

TagSet tagSet = new TagSet();

tagSet.addTag("tag1", "value1");
tagSet.addTag("tag2", "value2");

bucketTagInfo.setTagSet(tagSet);
obsClient.setBucketTagging("bucketname", bucketTagInfo);

// 关闭obsClient
obsClient.close();
```

说明

- 要使用桶标签功能，首先需要确保OBS服务端支持桶标签特性。
- 每个桶支持最多10个标签。
- 标签的Key和Value支持Unicode。

16.2 查看桶标签

您可以通过ObsClient.getBucketTagging查看桶标签。以下代码展示了如何查看桶标签：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
```

```
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketTagInfo bucketTagInfo = obsClient.getBucketTagging("bucketname");
for(Tag tag : bucketTagInfo.getTagSet().getTags()){
    System.out.println("\t" + tag.getKey() + ":" + tag.getValue());
}

// 关闭obsClient
obsClient.close();
```

16.3 删除桶标签

您可以通过ObsClient.deleteBucketTagging删除桶标签。以下代码展示了如何删除桶标签：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.deleteBucketTagging("bucketname");
// 关闭obsClient
obsClient.close();
```

17 事件通知

OBS提供了事件通知功能，当启用通知功能后，桶上所发生的指定操作都会通过消息通知服务（SMN）发送给您。事件通知配置包含：

- 配置ID，若用户未指定，则OBS自动分配一个ID。
- 事件通知主题的URN（Topic），当OBS检测到桶中特定的事件后，将会发布通知消息至该主题。
- 需要发布通知消息的事件类型列表。
- 过滤规则配置，用以匹配对象名前缀或者后缀。

更多关于事件通知的内容请参考**事件通知**。

17.1 设置事件通知

您可以通过ObsClient.setBucketNotification设置桶的事件通知。以下代码展示了如何设置桶的事件通知：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketNotificationConfiguration config = new BucketNotificationConfiguration();

config.setId("id1");
config.setTopic("urn:smn:cn-north-1:xxxx:topic1");
config.getEvents().add("s3:ObjectCreated:*");

Filter f = new Filter();
f.getFilterRules().add(new FilterRule("prefix", "smn"));
f.getFilterRules().add(new FilterRule("suffix", ".jpg"));
config.setFilter(f);

obsClient.setBucketNotification("bucketname", config);
// 关闭obsClient
obsClient.close();
```

17.2 查看事件通知

您可以通过ObsClient.getBucketNotification查看桶的事件通知。以下代码展示了如何查看桶的事件通知：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketNotificationConfiguration config = obsClient.getBucketNotification("bucketname");

System.out.println(config);

// 关闭obsClient
obsClient.close();
```

17.3 关闭事件通知

关闭桶事件通知实际上就是调用ObsClient.setBucketNotification将事件通知配置清空，以下代码展示了如何清除桶的事件通知：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.setBucketNotification("bucketname", new BucketNotificationConfiguration());

// 关闭obsClient
obsClient.close();
```

18 服务端加密

OBS支持服务端加密功能，使对象加密的行为在OBS服务端进行。

更多关于服务端加密的内容请参考**服务端加密**。

18.1 加密说明

OBS Java SDK支持服务端加密的接口见下表：

OBS Java SDK接口方法	描述	支持加密类型
ObsClient.putObject	上传对象时设置加密算法、密钥，对对象启用服务端加密。	SSE-KMS SSE-C
ObsClient.getObject	下载对象时设置解密算法、密钥，用于解密对象。	SSE-C
ObsClient.copyObject	1. 复制对象时设置源对象的解密算法、密钥，用于解密源对象。 2. 复制对象时设置目标对象的加密算法、密钥，对目标对象启用加密算法。	SSE-KMS SSE-C
ObsClient.getObjectMetadata	获取对象元数据时设置解密算法、密钥，用于解密对象。	SSE-C
ObsClient.initiateMultipartUpload	初始化分段上传任务时设置加密算法、密钥，对分段上传任务最终生成的对象启用服务端加密。	SSE-KMS SSE-C

OBS Java SDK接口方法	描述	支持加密类型
ObsClient.uploadPart	上传段时设置加密算法、密钥，对分段数据启用服务端加密。	SSE-C
ObsClient.copyPart	1. 复制段时设置源对象的解密算法、密钥，用于解密源对象。 2. 复制段时设置目标段的加密算法、密钥，对目标段启用加密算法。	SSE-C

18.2 加密示例

上传对象加密

以下代码展示了在上传对象时使用服务端加密功能：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

PutObjectRequest request = new PutObjectRequest();
request.setBucketName("bucketname");
request.setObjectKey("objectkey");
request.setFile(new File("localfile"));
// 设置SSE-C加密算法
SseCHeader sseCHeader = new SseCHeader();
sseCHeader.setAlgorithm(ServerAlgorithm.AES256);
sseCHeader.setSseCKey("your sse-c key generated by AES-256 algorithm");

request.setSseCHeader(sseCHeader);
obsClient.putObject(request);

// 关闭obsClient
obsClient.close();
```

下载对象解密

以下代码展示了在下载对象时使用服务端解密功能：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

GetObjectRequest request = new GetObjectRequest("bucketname", "objectkey");
// 设置SSE-C加密算法
SseCHeader sseCHeader = new SseCHeader();
sseCHeader.setAlgorithm(ServerAlgorithm.AES256);
// 此处的密钥必须和上传对象加密时使用的密钥一致
sseCHeader.setSseCKey("your sse-c key generated by AES-256 algorithm");

request.setSseCHeader(sseCHeader);
S3Object s3Object = obsClient.getObject(request);
```

```
s3Object.getObjectContent().close();

// 关闭obsClient
obsClient.close();
```

19 异常处理

19.1 OBS 服务端错误码

在向OBS服务端发出请求后，如果遇到错误，会在响应中包含响应的错误码描述错误信息。详细的错误码及其对应的描述和HTTP状态码见下表：

错误码	描述	HTTP状态码
AccessDenied	拒绝访问。	403 Forbidden
AccessForbidden	权限不足。	403 Forbidden
AccountProblem	用户的帐户出现异常（过期、冻结等），不能成功地完成操作。	403 Forbidden
AllAccessDisabled	用户无权限执行某操作。	403 Forbidden

错误码	描述	HTTP状态码
AmbiguousGrantByEmailAddress	用户提供的Email地址关联的帐户超过了1个。	400 Bad Request
BadDigest	客户端指定的对象内容的MD5值与系统接收到的内容MD5值不一致。	400 Bad Request
BadDomainName	域名不合法。	400 Bad Request
BadRequest	请求参数不合法。	400 Bad Request
BucketAlreadyExists	请求的桶名已经存在。桶的命名空间是系统中所有用户共用的，选择一个不同的桶名再重试一次。	409 Conflict
BucketAlreadyOwnedByYou	发起该请求的用户已经创建过了这个名字的桶，并拥有这个桶。	409 Conflict
BucketNotEmpty	用户尝试删除的桶不为空。	409 Conflict
CredentialsNotSupported	该请求不支持证书验证。	400 Bad Request
CustomDomainAreadyExist	配置了已存在的域。	400 Bad Request
CustomDomainNotExist	操作的域不存在。	400 Bad Request
DeregisterUserId	用户已经注销。	403 Forbidden
EntityTooSmall	用户试图上传的对象大小小于系统允许的最小大小。	400 Bad Request
EntityTooLarge	用户试图上传的对象大小超过了系统允许的最大大小。	400 Bad Request
FrozenUserId	用户被冻结。	403 Forbidden
IllegalVersioningConfiguration Exception	请求中的版本配置无效。	400 Bad Request
IllegalLocationConstraintException	配置了与所在Region不匹配的区域限制。	400 Bad Request
InArrearOrInsufficientBalance	因为ACL而没有权限进行某种操作。	403 Forbidden
IncompleteBody	请求体不完整。	400 Bad Request
IncorrectNumberOfFilesInPost Request	每个POST请求都需要带一个上传的文件。	400 Bad Request
InlineDataTooLarge	Inline Data超过了允许的最大长度。	400 Bad Request
InsufficientStorageSpace	存储空间不足。	403 Forbidden

错误码	描述	HTTP状态码
InternalError	系统遇到内部错误，请重试。	500 Internal Server Error
InvalidAccessKeyId	系统记录中不存在客户提供的Access Key Id。	403 Forbidden
InvalidAddressingHeader	用户必须指定匿名角色。	N/A
InvalidArgument	无效的参数。	400 Bad Request
InvalidBucketName	请求中指定的桶名无效。	400 Bad Request
InvalidBucket	请求访问的桶已不存在。	400 Bad Request
InvalidBucketState	无效的桶状态。	409 Conflict
InvalidBucketStoragePolicy	修改桶策略时，提供的新策略不合法。	400 Bad Request
InvalidDigest	HTTP头中指定的Content-MD5值无效。	400 Bad Request
InvalidEncryptionAlgorithmError	错误的加密算法。	400 Bad Request
InvalidLocationConstraint	创建桶时，指定的location不合法。	400 Bad Request
InvalidPart	一个或多个指定的段无法找到。这些段可能没有上传，或者指定的entity tag与段的entity tag不一致。	400 Bad Request
InvalidPartOrder	段列表的顺序不是升序，段列表必须按段号升序排列。	400 Bad Request
InvalidPayer	所有对这个对象的访问已经无效了。	403 Forbidden
InvalidPolicyDocument	表单中的内容与策略文档中指定的条件不一致。	400 Bad Request
InvalidRange	请求的range不可获得。	416 Client Requested Range Not Satisfiable
InvalidRedirectLocation	无效的重定向地址。	400 Bad Request
InvalidRequest	无效请求。	400 Bad Request
InvalidRequestBody	POST请求体无效。	400 Bad Request
InvalidSecurity	提供的安全证书无效。	403 Forbidden
InvalidStorageClass	用户指定的Storage Class无效。	400 Bad Request

错误码	描述	HTTP状态码
InvalidTargetBucketForLogging	delivery group对目标桶无ACL权限。	400 Bad Request
InvalidURI	无法解析指定的URI。	400 Bad Request
KeyTooLong	提供的Key过长。	400 Bad Request
MalformedACLError	提供的XML格式错误，或者不符合我们要求的格式。	400 Bad Request
MalformedError	请求中携带的XML格式不正确。	400 Bad Request
MalformedLoggingStatus	Logging的XML格式不正确。	400 Bad Request
MalformedPolicy	Bucket policy检查不通过。	400 Bad Request
MalformedPOSTRequest	POST请求的请求体不是结构化良好的多段或形式化数据。	400 Bad Request
MalformedQuotaError	Quota的XML格式不正确。	400 Bad Request
MalformedXML	当用户发送了一个配置项的错误格式的XML会出现这样的错误。错误消息是：“The XML you provided was not well-formed or did not validate against our published schema."。	400 Bad Request
MaxMessageLengthExceeded	请求消息过长。	400 Bad Request
MaxPostPreDataLengthExceeded Error	在上传文件前面的POST请求域过大。	400 Bad Request
MetadataTooLarge	元数据消息头超过了允许的最大元数据大小。	400 Bad Request
MethodNotAllowed	指定的方法不允许操作在请求的资源上。	405 Method Not Allowed
MissingContentLength	必须要提供HTTP消息头中的Content-Length字段。	411 Length Required
MissingRegion	请求中缺少Region信息，且系统无默认Region。	400 Bad Request
MissingRequestBodyError	当用户发送一个空的XML文档作为请求时会发生。错误消息是：“Request body is empty.”。	400 Bad Request
MissingRequiredHeader	请求中缺少必要的头域。	400 Bad Request

错误码	描述	HTTP状态码
MissingSecurityHeader	请求缺少一个必须的头。	400 Bad Request
NoSuchBucket	指定的桶不存在。	404 Not Found
NoSuchBucketPolicy	桶policy不存在。	404 Not Found
NoSuchCORSConfiguration	CORS配置不存在。	404 Not Found
NoSuchCustomDomain	请求的用户域不存在。	404 Not Found
NoSuchKey	指定的Key不存在。	404 Not Found
NoSuchLifecycleConfiguration	请求的LifeCycle不存在。	404 Not Found
NoSuchPolicy	给定的policy名字不存在。	404 Not Found
NoSuchUpload	指定的多段上传不存在。Upload ID不存在，或者多段上传已经终止或完成。	404 Not Found
NoSuchVersion	请求中指定的version ID与现存的所有版本都不匹配。	404 Not Found
NoSuchWebsiteConfiguration	请求的Website不存在。	404 Not Found
NotImplemented	用户提供的消息头功能上还没有实现。	501 Not Implemented
NotSignedUp	帐户未在系统中注册，必须先在系统中注册了才能使用该帐户。	403 Forbidden
OperationAborted	另外一个冲突的操作当前正作用在这个资源上，请重试。	409 Conflict
PermanentRedirect	尝试访问的桶必须使用指定的结点，请将以后的请求发送到这个结点。	301 Moved Permanently
PreconditionFailed	用户指定的先决条件中至少有一项没有包含。	412 Precondition Failed
Redirect	临时重定向。	307 Moved Temporarily
RequestIsNotMultiPartContent	桶POST必须是闭式的多段/表单数据。	400 Bad Request
RequestTimeout	用户与Server之间的socket连接在超时时间内没有进行读写操作。	400 Bad Request
RequestTimeTooSkewed	请求的时间与服务器的时间相差太大。	403 Forbidden

错误码	描述	HTTP状态码
RequestTorrentOfBucketError	不允许请求桶的torrent文件。	400 Bad Request
ServiceNotImplemented	请求的方法服务端没有实现。	501 Not Implemented
ServiceNotSupported	请求的方法服务端不支持。	409 Conflict
ServiceUnavailable	服务器过载或者内部错误异常。	503 Service Unavailable
SignatureDoesNotMatch	请求中带的签名与系统计算得到的签名不一致。检查您的访问密钥（AK和SK）和签名计算方法。	403 Forbidden
SlowDown	请降低请求频率。	503 Service Unavailable
System Capacity Not enough	系统空间不足异常 。	403 Forbidden
TooManyCustomDomains	配置了过多的用户域。	400 Bad Request
TemporaryRedirect	当DNS更新时，请求将被重定向到桶。	307 Moved Temporarily
TooManyBuckets	用户拥有的桶的数量达到了系统的上限，并且请求试图创建一个新桶。	400 Bad Request
TooManyObjectCopied	用户单个对象被拷贝的数量超过系统上限。	400 Bad Request
TooManyWrongSignature	因高频错误请求被拒绝服务。	400 Bad Request
UnexpectedContent	该请求不支持带内容字段。	400 Bad Request
UnresolvableGrantByEmailAddress	用户提供的Email与记录中任何帐户的都不匹配。	400 Bad Request
UserKeyMustBeSpecified	请求中缺少用户的AK信息。	400 Bad Request
WebsiteRedirect	Website请求缺少bucketName。	301 Moved Permanently
KMS.DisabledException	SSE-KMS加密方式下，主密钥被禁用。	400 Bad Request
KMS.NotFoundException	SSE-KMS加密方式下，主密钥不存在。	400 Bad Request
RestoreAlreadyInProgress	对象正在取回，请求冲突。	409 Conflict
ObjectHasAlreadyRestored	已经取回的对象，禁止缩短取回保存时间。	409 Conflict

错误码	描述	HTTP状态码
InvalidObjectState	取回对象不是归档存储对象。	403 Forbidden
InvalidTagError	配置桶标签时，提供了无效的Tag。	400 Bad Request
NoSuchTagSet	指定的桶没有设置标签。	404 Not Found

19.2 SDK 自定义异常

SDK自定义异常（ObsException）是由ObsClient统一抛出的异常，继承自java.lang.RuntimeException类。通常是OBS服务端错误，包含**OBS错误码**、错误信息等，便于用户定位问题，并做出适当的处理。

ObsException通常包含以下错误信息：

- ObsException.getResponseCode：HTTP状态码。
- ObsException.getErrorCode：OBS服务端错误码。
- ObsException.getErrorMessage：OBS服务端错误描述。
- ObsException.getErrorRequestId：OBS服务端返回的请求ID。
- ObsException.getErrorHostId：请求的服务端ID。
- ObsException.getResponseHeaders：HTTP响应头信息。

19.3 SDK 公共响应头

调用ObsClient类的相关接口成功后，均会返回公共响应头类，即HeaderResponse类实例（或其子类实例），该类包含了HTTP/HTTPS的响应头信息。

处理公共响应头的示例代码如下：

```
String endPoint = "obs.myhwclouds.com";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// 创建ObsClient实例
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
HeaderResponse response = obsClient.createBucket("bucketname");

// 从公共响应头中获取request-id
System.out.println("\t" + response.getResponseHeaders().get("request-id"));

obsClient.close();
```

19.4 OBS 客户端通用示例

调用ObsClient类的相关接口时，没有异常抛出，则表明返回值有效，返回**SDK公共响应头**实例或其子类实例；若抛出异常，则说明操作失败，此时应从**SDK自定义异常**实例中获取错误信息。

以下代码展示了使用OBS客户端的通用方式：

```
ObsClient obsClient = null;
try
{
    String endPoint = "obs.myhwclouds.com";
    String ak = "*** Provide your Access Key ***";
    String sk = "*** Provide your Secret Key ***";
    // 创建ObsClient实例
    obsClient = new ObsClient(ak, sk, endPoint);
    // 调用接口进行操作，例如上传对象
    HeaderResponse response = obsClient.putObject("bucketname", "objectkey", new
File("localfile"));
    System.out.println(response);
}
catch (ObsException e)
{
    System.out.println("Response Code: " + e.getResponseCode());
    System.out.println("Error Message: " + e.getErrorMessage());
    System.out.println("Error Code:       " + e.getErrorCode());
    System.out.println("Request ID:      " + e.getErrorRequestId());
    System.out.println("Host ID:           " + e.getErrorHostId());
}finally{
    // 关闭obsClient
    if(obsClient != null){
        try
        {
            obsClient.close();
        }
        catch (IOException e)
        {
        }
    }
}
```

19.5 日志分析

日志路径

OBS Java SDK的日志路径是通过log4j2.xml指定的，默认存放于JDK系统变量user.dir所代表的路径下。通常包含三个日志文件：

文件名	说明
OBS-SDK.interface_north.log	北向日志，OBS Java SDK与用户第三方应用交互的日志记录。
OBS-SDK.interface_south.log	南向日志，OBS Java SDK与OBS服务端交互的日记记录。
OBS-SDK.run.log	OBS客户端运行日志。

日志内容格式

SDK日志格式为：日志时间|线程号|日志级别|日志内容。示例如下：

```
#南向日志
2017-08-21 17:40:07 133|main|INFO |HttpClient cost 157 ms to apply http request
2017-08-21 17:40:07 133|main|INFO |Received expected response code: true
2017-08-21 17:40:07 133|main|INFO |expected code(s): [200, 204].

#北向日志
```

```
2017-08-21 17:40:06 820|main|INFO |Storage|1|HTTP+XML|ObsClient||||2017-08-21 17:40:05|2017-08-21
17:40:06|||0|
2017-08-21 17:40:07 136|main|INFO |Storage|1|HTTP+XML|setObjectAcl||||2017-08-21 17:40:06|
2017-08-21 17:40:07|||0|
2017-08-21 17:40:07 137|main|INFO |ObsClient [setObjectAcl] cost 312 ms
```

日志级别

当系统出现问题需要定位且当前的日志无法满足要求时，可以通过修改日志的级别来获取更多的信息。其中TRACE日志信息最丰富，ERROR日志信息最少。

具体说明如下：

- OFF: 关闭级别，如果设置为这个级别，日志打印功能将被关闭。
- TRACE：跟踪级别，如果设置为这个级别，将打印所有日志信息。通常不建议使用。
- DEBUG：调试级别，如果设置为这个级别，除了打印INFO级别的信息外，还将打印每次HTTP/HTTPS请求和响应的头信息，鉴权算法计算出的StringToSign信息等。
- INFO：信息级别，如果设置为这个级别，除了打印WARN级别的信息外，还将打印HTTP/HTTPS请求的耗时时间，ObsClient接口的耗时时间等。
- WARN：告警级别，如果设置为这个级别，除了打印ERROR级别的信息外，还将打印部分关键事件的信息，如重试请求超过最大次数等。
- ERROR：错误级别，如果设置为这个级别，仅打印发生异常时的错误信息。

设置方式

以下代码分别为南向日志、北向日志和OBS客户端运行日志设置了不同的日志级别（日志配置文件的全部内容参见log4j2.xml配置文件）：

```
<!-- north log -->
<Logger name="com.obs.services.ObsClient" level="INFO" additivity="false">
      <AppenderRef ref="NorthInterfaceLogAppender" />
</Logger>

<!-- south log -->
<Logger name="com.obs.services.internal.RestStorageService" level="WARN" additivity="false">
       <AppenderRef ref="SouthInterfaceLogAppender" />
</Logger>

<!-- running log -->
<Logger name="com.obs.log.RunningLog" level="ERROR" additivity="false">
       <AppenderRef ref="RunningLogAppender" />
</Logger>
```

日志无法生成

当在类路径下加入log4j2.xml配置文件后，仍无法生成日志，可以尝试使用代码直接指定log4j2.xml配置文件的路径。示例代码如下：

```
org.apache.logging.log4j.core.config.ConfigurationSource source = new
org.apache.logging.log4j.core.config.ConfigurationSource(new FileInputStream("*** your log4j2.xml
path ***"));
org.apache.logging.log4j.core.config.Configurator.initialize(null, source);
```

19.6 缺少类错误

使用OBS Java SDK进行二次开发时如果报缺少类错误，请检查类路径下除esdk-obs-java-x.x.x.jar外是否包含以下依赖库：

- java-xmlbuilder-1.1.jar
- jna-4.1.0.jar
- commons-codec-1.9.jar
- commons-logging-1.2.jar
- httpclient-4.5.3.jar
- httpasyncclient-4.1.2.jar
- httpcore-4.4.4.jar
- httpcore-nio-4.4.4.jar
- log4j-api-2.8.2.jar
- log4j-core-2.8.2.jar

说明

OBS Java SDK进行了全面升级，现在已去掉对jets3t-0.9.0.jar，log4j-1.2.17.jar等库的依赖。

19.7 连接超时异常

连接超时异常通常有两种表现形式：

1. 通过ObsException获取到的异常堆栈出现org.apache.http.conn.ConnectTimeoutException或java.net.SocketTimeoutException。
2. 通过ObsException.getResponseCode获取到的错误码为-1。

出现这类异常的原因一般是服务地址（Endpoint）错误或网络不通导致无法连接OBS服务，此时请检查服务地址和网络状况。

19.8 资源无法释放

如果发现使用OBS Java SDK后存在内存泄露或OBS服务端连接未断开等情况，请检查是否正确调用了ObsClient.close以及S3Object.getObjectContent.close释放资源。

19.9 缓存溢出异常

如果上传对象报错，通过ObsException获取到错误信息“Input stream cannot be reset as xxx bytes have been written, exceeding the available buffer size of xxx”。请通过以下两种方法解决该问题：

1. 如果是上传本地文件，请直接使用文件上传而不是流式上传调用上传对象接口。
2. 如果是上传其他流式数据（如网络流），调用ObsConfiguration.setUploadStreamRetryBufferSize设置更大的缓存空间。

19.10 签名不匹配异常

如果从ObsException中获取到HTTP状态码为403，OBS服务端错误码为SignatureDoesNotMatch，请检查AK/SK是否有误。

19.11 永久重定向异常

如果从ObsException中获取到HTTP状态码为301，OBS服务端错误码为PermanentRedirect，表明请求的桶不在当前所访问的域名下。可通过如下两种方式解决：

1. 配置虚拟主机访问方式：初始化OBS客户端时设置ObsConfiguration.setDisableDnsBucket(false)。
2. 修改当前所访问的域名为桶实际所在域名：从ObsException.getResponseHeaders中获取bucket-region头信息的值，确定桶实际所在的区域，再从**服务地址和区域信息**列表中寻找代表该区域的域名。

20 常见问题

更多常见问题参见**对象存储服务常见问题**。

20.1 如何进行分段上传

参见**分段上传**。

20.2 如何创建文件夹

参见**创建文件夹**。

20.3 如何列出所有对象

参见**列举对象**、**列举多版本对象**。

20.4 如何生成临时授权的 URL

参见**临时授权访问**。

20.5 如何使用表单上传

参见**基于表单上传**。

20.6 如何分段下载大对象

参见**范围下载**，或直接下载示例代码：ConcurrentDownloadObjectSample.zip。

20.7 如何验证服务端根证书

参见**配置SDK验证服务端证书**。

20.8 如何使对象可以被匿名用户访问

参见**网站文件托管**。

20.9 如何确定 OBS 服务地址和区域信息

参见**服务地址**。

20.10 如何获取访问密钥

参见**OBS服务环境搭建**。

20.11 如何使用 Maven 下载 SDK

OBS Java SDK暂不支持直接使用Maven下载，仅在SDK软件包提供了pom.xml文件指明依赖的第三方库信息。

A 修订记录

发布日期	修订记录
2017-10-10	第一次正式发布。
2017-11-30	第二次正式发布。 新增章节： 6.10 桶存储类型 7.7 断点续传上传 8.7 断点续传下载 修改章节： 7.4 设置对象属性，新增设置对象存储类型相关内容