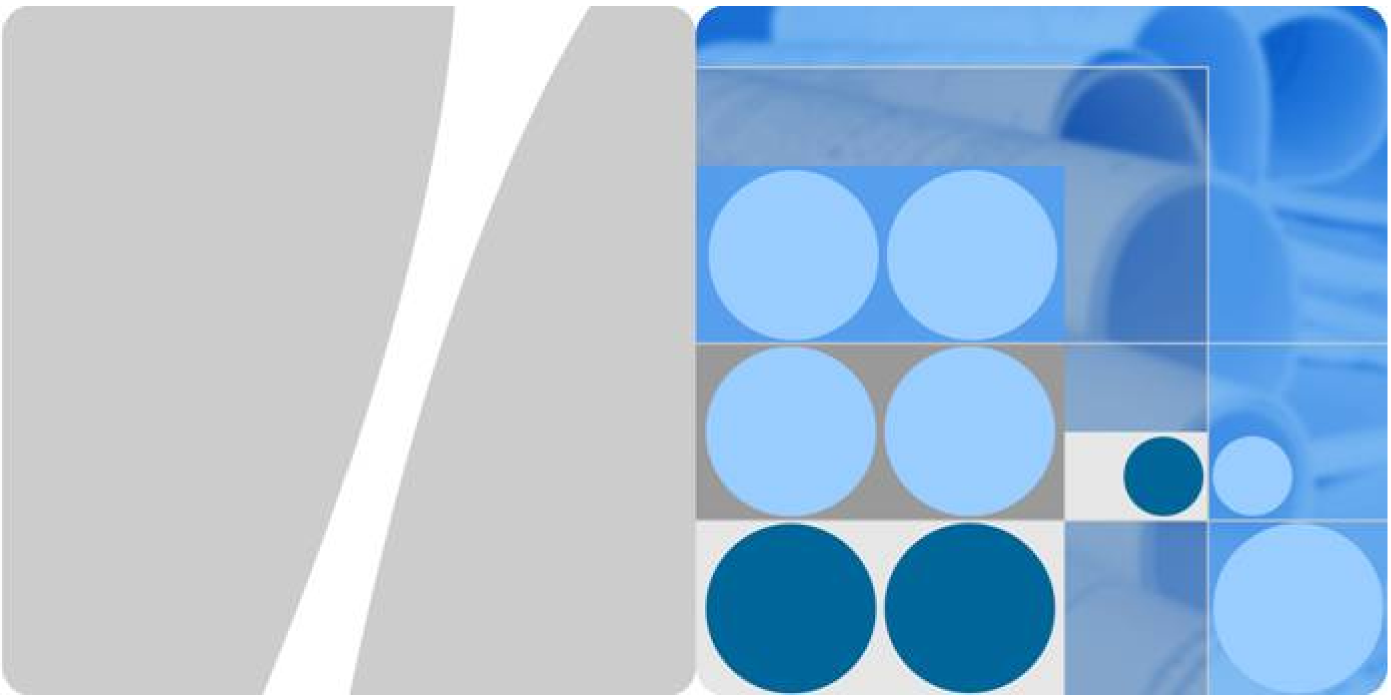

Object Storage Service

Java SDK Developer Guide

Date **2020-02-26**

Contents

1 SDK Download Links

SDK Source Codes and API Documentation

- Latest version of OBS Java SDK: Click **here** to download.
- OBS Java SDK API document: **OBS Java SDK API Reference**

Compatibility

- Recommended JDK versions: 7, 8, 9, and 10
- Third-party dependency: This version is not completely compatible with earlier versions (2.1.*x*). **httpclient4.***x* is replaced with **okhttp3**.
- Namespace: Compatible with earlier versions (2.1.*x*). All external APIs are contained in the **com.obs.services**, **com.obs.services.model**, and **com.obs.services.exception** packages.
- API functions: Compatible with earlier versions (2.1.*x*).

2 Example Programs

OBS Java SDK provides abundant example programs for your reference and direct use. These programs can be obtained from the OBS Java SDK. For example, files in **eSDK_Storage_OBS_**<*VersionId*>**_Java.zip** obtained by decompressing **eSDK_Storage_OBS_**<*VersionId*>**_Java/samples_java** are example programs. Alternatively, you can click code package names provided in the following table to obtain corresponding example programs.

Example programs include:

Sample Code	Description
BucketOperationsSample	How to use bucket-related APIs.
ObjectOperationsSample	How to use object-related APIs.
DownloadSample	How to download an object.
CreateFolderSample	How to create a folder.
DeleteObjectsSample	How to delete objects in a batch.
ListObjectsSample	How to list objects.
ListVersionsSample	How to list versioning objects.
ListObjectsInFolderSample	How to list objects in a folder.
ObjectMetaSample	How to customize object metadata.
SimpleMultipartUploadSample	How to perform a multipart upload.
RestoreObjectSample	How to download Cold objects.
ConcurrentCopyPartSample	How to concurrently copy parts of a large object.
ConcurrentDownloadObjectSample	How to concurrently download parts of a large object.
ConcurrentUploadPartSample	How to concurrently upload parts of a large object.

Sample Code	Description
PostObjectSample	How to perform a browser-based upload.
TemporarySignatureSample	How to use URLs for authorized access.
GetTokenSample	How to obtain the security token.

3 Quick Start

3.1 Before You Start

- Ensure that you are familiar with OBS basic concepts from **Help Center**, such as bucket, object, region, and AK and SK.
- You can see **General Examples of ObsClient** to understand how to call OBS Java SDK APIs in a general manner.
- After an API calling is complete using an instance of **ObsClient**, view whether an exception is thrown. If no, the return value is valid. If yes, the operation fails and you can obtain the error information from an instance of **ObsException**.
- After an API is successfully called by an instance of **ObsClient**, an instance of **ResponseHeader** containing the response headers will be returned.
- Some features are available only in some regions. If the HTTP status code of an API is 405, check whether the region supports this feature.

3.2 Creating Access Keys

OBS uses AKs and SKs in user accounts for signature verification to ensure that only authorized accounts can access specified OBS resources. Detailed explanations about AK and SK are as follows:

- An access key ID (AK) defines a user who accesses the OBS system. An AK belongs to only one user, but one user can have multiple AKs. The OBS system recognizes the users who access the system by their access key IDs.
- A secret access key (SK) is the key used by users to access OBS. It is the authentication information generated based on the AK and the request header. An SK matches an AK, and they group into a pair.

Access keys are classified into permanent access keys (AK/SK) and temporary access keys (AK/SK and security token). Permanent access keys are valid for a year after creation. Each user can create up to two valid AK/SK pairs. Temporary access keys can be used to access OBS only within the specified validity period. After the temporary access keys expire, they need to be obtained again. For security purposes, you are advised to use temporary access keys to access OBS, or periodically update your access keys if you use permanent access keys. The following describes how to obtain access keys of these two types.

Permanent Access Keys

1. Log in to OBS Console.
2. In the upper right corner of the page, hover the cursor over the username and choose **My Credentials**.
3. On the **My Credentials** page, select **Access Keys** in the navigation pane on the left.
4. On the **Access Keys** page, click **Create Access Key**.
5. In the **Create Access Key** dialog box that is displayed, enter the password and verification code.

   NOTE

   - If you have not bound an email address or mobile number, enter only the password.
   - If you have bound an email address and a mobile number, you can select the verification by either email or mobile phone.
6. Click **OK**.
7. In the **Download Access Key** dialog box that is displayed, click **OK** to save the access keys to your browser's default download path.
8. Open the downloaded **credentials.csv** file to obtain the access keys (AK and SK).

   NOTE

   - A user can create a maximum of two valid access keys.
   - Keep the access key properly. If you click **Cancel** in the dialog box, the access keys will not be downloaded, and cannot be obtained later. You can re-create access keys if you need to use them.

Temporary Access Keys

The temporary AK/SK and security token are temporary access tokens issued by the system to users. The validity period ranges from 15 minutes to 24 hours which can be set using APIs. After the validity period expires, users need to obtain the access keys again. The temporary AK/SK and security token shall observe the principle of least privilege. When the temporary AK/SK are used for authentication, the temporary AK/SK and security token must be used at the same time.

For details about how to obtain temporary access keys, see **Obtaining a Temporary AK/SK**.

For details about how to use temporary access keys, see **4.2 Creating an Instance of ObsClient**.

3.3 Preparing a Development Environment

- Download a recommended version of JDK from the **Oracle's official website** and install it.

- The latest version of Eclipse IDE for Java Developers is required and can be downloaded from the **Eclipse's official website**.

3.4 Installing the SDK

Import the JAR files in the Eclipse Java project as follows:

Step 1 **Download** the OBS Java SDK.

Step 2 Decompress the SDK.

Step 3 Copy all JAR files in the decompressed libs folder to your project.

Step 4 On Eclipse, select the project and choose **Properties** > **Java Build Path** > **Add JARs**.

Step 5 Select all JAR files that have been copied in step 3, click **OK** to finish importing JAR files.

----End

3.5 Obtaining Endpoints

- You can click **here** to view the endpoints and regions enabled for OBS.

NOTICE

The SDK allows you to pass endpoints with or without the protocol name. Suppose the endpoint you obtained is **your-endpoint**. The endpoint passed when initializing an instance of **ObsClient** can be **http://*your-endpoint***, **https://*your-endpoint***, or ***your-endpoint***.

3.6 Initializing an Instance of ObsClient

Each time you want to send an HTTP/HTTPS request to OBS, you must create an instance of **ObsClient**. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Use the instance to access OBS.

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

NOTE

For more information, see chapter "Initialization."

3.7 Creating a Bucket

A bucket is a global namespace of OBS and is a data container. It functions as a root directory of a file system and can store objects. The following code shows how to create a bucket:

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

NOTE

- Bucket names are globally unique. Ensure that the bucket you create is named differently from any other bucket.
- A bucket name must comply with the following rules:
  - Contains 3 to 63 characters, chosen from lowercase letters, digits, hyphens (-), and periods (.), and starts with a digit or letter.
  - Cannot be an IP-like address.
  - Cannot start or end with a hyphen (-) or period (.)
  - Cannot contain two consecutive periods (.), for example, **my..bucket**.
  - Cannot contain periods (.) and hyphens (-) adjacent to each other, for example, **my-.bucket** or **my.-bucket**.
- If you create buckets of the same name, no error will be reported and the bucket properties comply with those set in the first creation request.
- For more information, see **6.1 Creating a Bucket**.

3.8 Uploading an Object

Sample code:

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

NOTE

For more information, see **7.1 Object Upload Overview**.

3.9 Downloading an Object

Sample code:

```
ObsObject obsObject = obsClient.getObject("bucketname", "objectname");
InputStream content = obsObject.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();
}
```

NOTE

- When you call **ObsClient.getObject**, an instance of **ObsObject** will be returned. This instance contains the contents and properties of the object.
- When you call **ObsObject.getObjectContent** to obtain an object input stream, you can read the input stream to obtain its contents. Close the input stream after use.
- For more information, see **8.1 Object Download Overview**.

3.10 Listing Objects

After objects are uploaded, you may want to view the objects contained in a bucket. Sample code is as follows:

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

NOTE

- When you call **ObsClient.listObjects**, an instance of **ObjectListing** will be returned. This instance contains the response of the **listObject** request. You can use **ObjetListing.getObjects** to obtain description of all of the listed objects.
- In the previous sample code, 1000 objects will be listed, by default.
- For more information, see **Listing Objects**.

3.11 Deleting an Object

Sample code:

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

3.12 General Examples of ObsClient

After an API calling is complete using an instance of ObsClient, view whether an exception is thrown. If no, the return value is valid and an instance of the **HeaderResponse** class (or of its sub-class) is returned. If yes, obtain the error information from the instance of **ObsException**.

Sample code:

```
// You can reserve only one global instance of ObsClient in your project.
// ObsClient is thread-safe and can be simultaneously used by multiple threads.
ObsClient obsClient = null;
try
{
    String endPoint = "https://your-endpoint";
    String ak = "*** Provide your Access Key ***";
    String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
    obsClient = new ObsClient(ak, sk, endPoint);
    // Call APIs to perform related operations, for example, uploading an object.
    HeaderResponse response = obsClient.putObject("bucketname", "objectname", new File("localfile"));  //
localfile indicates the path of the local file to be uploaded. You need to specify the file name.
    System.out.println(response);
}
```

```
catch (ObsException e)
{
    System.out.println("HTTP Code: " + e.getResponseCode());
    System.out.println("Error Code:" + e.getErrorCode());
    System.out.println("Error Message: " + e.getErrorMessage());

    System.out.println("Request ID:" + e.getErrorRequestId());
    System.out.println("Host ID:" + e.getErrorHostId());
}finally{
    // Close the instance of ObsClient. If this instance is a global one, you do not need to close it every time
you complete calling a method.
    // After you call the ObsClient.close method to close an instance of ObsClient, the instance cannot be
used any more.
    if(obsClient != null){
        try
        {
            // obsClient.close();
        }
        catch (IOException e)
        {
        }
    }
}
```

4 Initialization

4.1 Configuring the AK and SK

To use OBS, you need a valid pair of AK and SK for signature authentication. For details, see **3.2 Creating Access Keys**.

After obtaining the AK and SK, you can start initialization.

4.2 Creating an Instance of ObsClient

ObsClient functions as the Java client for accessing OBS. It offers callers a series of APIs for interaction with OBS and is used for managing and performing operations on resources, such as buckets and objects, stored in OBS. To use OBS Java SDK to send a request to OBS, you need to initialize an instance of ObsClient and modify the default configurations in ObsConfiguration based on actual needs.

- If you use the endpoint to create an instance of ObsClient, all parameters are in their default values and cannot be modified.
  - Sample code for creating an instance of ObsClient using permanent access keys (AK/SK):
    ```
    String endPoint = "https://your-endpoint";
    String ak = "*** Provide your Access Key ***";
    String sk = "*** Provide your Secret Key ***";
    // Create an instance of ObsClient.
    ObsClient obsClient = new ObsClient(ak, sk, endPoint);
    // Use the instance to access OBS.
    // Close ObsClient.
    obsClient.close();
    ```
  - Sample code for creating an instance of ObsClient using temporary access keys (AK/SK and security token):
    ```
    String endPoint = "https://your-endpoint";
    String ak = "*** Provide your Access Key ***";
    String sk = "*** Provide your Secret Key ***";
    String securityToken = "*** Provide your Security Token ***";
    // Create an instance of ObsClient.
    ObsClient obsClient = new ObsClient(ak, sk, securityToken, endPoint);
    // Use the instance to access OBS.
    // Close ObsClient.
    obsClient.close();
    ```

NOTE

For details about how to obtain and use temporary AK/SK and security token, see **2 Example Programs**.

- Sample code for creating an instance of ObsClient using **BasicCredentialsProvider**:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(new BasicObsCredentialsProvider(ak, sk), endPoint);
// Use the instance to access OBS.
// Close ObsClient.
obsClient.close();
```

- Sample code for creating an instance of ObsClient using **EnvironmentVariableObsCredentialsProvider**:

```
String endPoint = "https://your-endpoint";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(new EnvironmentVariableObsCredentialsProvider(),
endPoint);
// Use the instance to access OBS.
// Close ObsClient.
obsClient.close();
```

NOTE

In the preceding code, the access keys are found in the system environment variables. You need to define **OBS_ACCESS_KEY_ID** and **OBS_SECRET_ACCESS_KEY** in the system environment variables to represent the permanent AK and SK respectively.

- Sample code for creating an instance of ObsClient using **EcsObsCredentialsProvider**:

```
String endPoint = "https://your-endpoint";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(new EcsObsCredentialsProvider(), endPoint);
// Use the instance to access OBS.
// Close ObsClient.
obsClient.close();
```

NOTE

When an application is deployed on an ECS, the instance of ObsClient created using the preceding methods automatically obtains the temporary access keys from the ECS and updates them periodically.

NOTICE

Ensure that the UTC time of the server is the same as that of the environment where the application is deployed. Otherwise, the temporary access keys may fail to be updated in time.

- In addition to the preceding methods, you can also search in sequence to obtain the corresponding access keys from the environment variables and ECSs.
  - Sample code for creating an instance of ObsClient using the access keys obtained by searching in sequence:

```
String endPoint = "https://your-endpoint";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(new OBSCredentialsProviderChain(), endPoint);
```

```
// Use the instance to access OBS.
// Close ObsClient.
obsClient.close();
```

NOTE

The preceding method specifies that the access keys are searched from the predefined list in sequence. By default, the system provides two predefined search methods: obtaining the access keys from the environment variables and obtaining from ECSs. ObsClient searches for the access keys from the environment variables first and then from ECSs. In this case, ObsClient is created using the first pair of access keys obtained in the search.

- If you use ObsConfiguration to create an instance of ObsClient, you can set configuration parameters as needed during the creation. After the instance is created, the parameters cannot be modified. For parameter details, see **4.3 Configuring an Instance of ObsClient**. The preceding methods of creating an instance of ObsClient support ObsConfiguration. The sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an ObsConfiguration instance.
ObsConfiguration config = new ObsConfiguration();
config.setEndPoint(endPoint);
config.setSocketTimeout(30000);
config.setMaxErrorRetry(1);

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, config);

// Create an instance of ObsClient using Provider.
// ObsClient obsClient = new ObsClient(new EnvironmentVariableObsCredentialsProvider(), config);
// ObsClient obsClient = new ObsClient(new EcsObsCredentialsProvider(), config);

// Use the instance to access OBS.

// Close ObsClient.
obsClient.close();
```

NOTE

- The project can contain one or more instances of ObsClient.
- ObsClient is thread-safe and can be simultaneously used by multiple threads.

NOTICE

After you call **ObsClient.close** to close an instance of ObsClient, the instance cannot be used any more.

4.3 Configuring an Instance of ObsClient

When you call the **ObsConfiguration** configuration class to create an instance of **ObsClient**, you can configure the agent, timeout duration, maximum allowed number of connections, and some other parameters listed in the following table.

Parameter	Description	Method	Recommended Value
connectionTimeout	Timeout period for establishing an HTTP/HTTPS connection, in ms. The default value is **60,000**.	ObsConfiguration.setConnectionTimeout	[10000, 60000]
socketTimeout	Timeout duration for transmitting data at the Socket layer, in ms. The default value is **60,000**.	ObsConfiguration.setSocketTimeout	[10000, 60000]
idleConnectionTime	Allowed connection idle time, in ms. If a connection exceeds the specified value, the connection will be closed. The default value is **30,000**.	ObsConfiguration.setIdleConnectionTime	Default
maxIdleConnections	Maximum number of allowed idle connections in the connection pool. The default value is **1000**.	ObsConfiguration.setMaxIdleConnections	N/A
maxConnections	Maximum number of concurrent HTTP requests. The default value is **1000**.	ObsConfiguration.setMaxConnections	Default

Parameter	Description	Method	Recommended Value
maxErrorRetry	Maximum number of retry attempts (caused by abnormal requests, **500**, **503**, and other errors). The default value is **3**. **NOTE** This parameter is invalid in object upload and download APIs if an interruption occurs after an upload or download task enters the data flow processing phase. In this case, no retry is performed.	ObsConfiguration.setMaxErrorRetry	[0, 5]
endPoint	Endpoint for accessing OBS, which contains the protocol type, domain name (or IP address), and port number. For example, https://your-endpoint:443.	ObsConfiguration.setEndPoint	N/A
httpProxy	HTTP proxy configuration. This parameter is left blank by default.	ObsConfiguration.setHttpProxy	N/A
validateCertificate	Whether to verify the server certificate. The default value is **false**.	ObsConfiguration.setValidateCertificate	N/A
verifyResponseContentType	Whether to verify **ContentType** of the response header. The default value is **true**.	ObsConfiguration.setVerifyResponseContentType	Default

Parameter	Description	Method	Recommended Value
uploadStreamRetryBufferSize	Size of the cache used for uploading a stream object, in bytes. The default size is **512 KB**.	ObsConfiguration.setUploadStreamRetryBufferSize	N/A
readBufferSize	Cache size for downloading the object from socket streams, in bytes. Value **-1** indicates that cache is not configured. The default value is **-1**.	ObsConfiguration.setReadBufferSize	N/A
writeBufferSize	Cache size for uploading the object to socket streams, in bytes. Value **-1** indicates that cache is not configured. The default value is **-1**.	ObsConfiguration.setWriteBufferSize	N/A
socketWriteBufferSize	Buffer size for sending a socket, in bytes. This parameter corresponds to **java.net.SocketOptions.SO_SNDBUF**. The default value is **-1**, which indicates no limitation.	ObsConfiguration.setSocketWriteBufferSize	Default value
socketReadBufferSize	Buffer size for receiving a socket, in bytes. This parameter corresponds to **java.net.SocketOptions.SO_RCVBUF**. The default value is **-1**, which indicates no limitation.	ObsConfiguration.setSocketReadBufferSize	Default value

Parameter	Description	Method	Recommended Value
keyManagerFactory	Factory used for generating **javax.net.ssl.KeyManager**. This parameter is left blank by default.	ObsConfiguration.setKeyManagerFactory	N/A
trustManagerFactory	Factory used for generating **javax.net.ssl.TrustManager**. This parameter is left blank by default.	ObsConfiguration.setTrustManagerFactory	N/A
isStrictHostnameVerification	Whether to strictly verify the server-side host name. The default value is **false**.	ObsConfiguration.setIsStrictHostnameVerification	N/A
keepAlive	Whether to use persistent connections to access OBS. The default value is **true**.	ObsConfiguration.setKeepAlive	N/A
cname	Whether to use self-defined domain name to access OBS. The default value is **false**.	ObsConfiguration.setCname	N/A
sslProvider	Provider of **SSLContext**. The **SSLContext** provided by JDK is used by default.	ObsConfiguration.setSslProvider	N/A
httpProtocolType	HTTP protocol type used for accessing OBS servers. The default protocol is HTTP 1.1.	ObsConfiguration.setHttpProtocolType	N/A
httpDispatcher	Customize a dispatcher.	ObsConfiguration.setHttpDispatcher	N/A

NOTE

- Parameters whose recommended value is **N/A** need to be set according to the actual conditions.
- To improve the upload and download performance of files in the case that the network bandwidth meets the requirements, you can tune the **socketWriteBufferSize**, **sockeReadBufferSize**, **readBufferSize**, and **writeBufferSize** parameters.
- If the network is unstable, you are advised to set larger values for **connectionTimeout** and **socketTimeout**.
- If the value of **endPoint** does not contain any protocol, HTTPS is used by default.
- For the sake of high DNS resolution performance and OBS reliability, you can set **endPoint** only to the domain name of OBS, instead of the IP address.

4.4 Configuring SDK Logging

OBS Java SDK provides the logging function, based on the Apache Log4j2 open library. The SDK records WARN logs to the path represented by the JDK system variable **user.dir**, by default. You can modify log configuration files to configure logging based on your needs. The procedure is as follows:

Step 1 Find the **log4j2.xml** file in the OBS Java SDK development package.

Step 2 Modify log levels and save paths in the **log4j2.xml** file based on actual needs.

Step 3 Save the **log4j2.xml** file to the **classpath** root directory, or call **Log4j2Configurator.setLogConfig** to specify the save path of **log4j2.xml** directly.

----End

NOTE

- For details about SDK logs, see **17.5 Log Analysis**.

4.5 Configuring Server-Side Certificate Verification

OBS Java SDK supports server-side certificate verification to ensure that OBS is provided by trusted servers. The following details how to configure server certificate verification in Windows. (In Linux, replace **%JAVA_HOME%** with **$JAVA_HOME**.)

NOTE

If the root certificate on the OBS server is issued by an authoritative CA, skip steps 1 to 3. (Root certificates issued by authoritative CAs are in the certificate library of JDK.)

Step 1 Obtain the root certificate of the OBS server (for example, open Internet Explorer and choose **Internet Options** > **Content** > **Certificates** to export the certificate) and save it by the name of **obs.cer**.

Step 2 Run the **%JAVA_HOME%/bin/keytool -import -alias obs -file obs.cer -storepass changeit -keystore %JAVA_HOME%/jre/lib/security/cacerts** command to import the certificate.

Step 3 Run the **%JAVA_HOME%/bin/keytool -list -v -alias obs -storepass changeit -keystore %JAVA_HOME%/jre/lib/security/cacerts** command to view whether the certificate is successfully imported.

Step 4 Enable server certificate verification (**ObsConfiguration.setValidateCertificate(true)**).

----End

4.6 Transparently Transferring the AK and SK

OBS Java SDK provides **SecretFlexibleObsClient** that supports transparent transfer of AKs and SKs in API functions. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
// Create an ObsConfiguration instance.
ObsConfiguration config = new ObsConfiguration();
config.setEndPoint(endPoint);

// Create a SecretFlexibleObsClient instance.
SecretFlexibleObsClient obsClient = new SecretFlexibleObsClient(config);
// Use the instance to access 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);

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

NOTE

SecretFlexibleObsClient is inherited from **ObsClient** and can be used as **ObsClient**.

5 Fault Locating

5.1 Methods

If problems occur when using the OBS Java SDK, you can perform the following steps to analyze and locate the problems.

Step 1 Make sure that the latest version of OBS Java SDK is used. Click **here** to download the latest version.

Step 2 Make sure that the logging function of OBS Java SDK is enabled. For details about how to enable the function, see the **Log Analysis** section. The recommended log level is WARN.

Step 3 Make sure that the program code of the OBS Java SDK complies with **General Examples of ObsClient**. All ObsClient APIs are processed with exception handling. The following is an example code of uploading an object:

```
ObsClient obsClient = null;
try
{
    String endPoint = "https://your-endpoint";
    String ak = "*** Provide your Access Key ***";
    String sk = "*** Provide your Secret Key ***";
    obsClient = new ObsClient(ak, sk, endPoint);
    HeaderResponse response = obsClient.putObject("bucketname", "objectname", new
ByteArrayInputStream("Hello OBS".getBytes()));
    // Optional: After the API is successfully called, record the HTTP status code and request ID returned by
the server.
    System.out.println(response.getStatusCode());
    System.out.println(response.getRequestId());
}
catch (ObsException e)
{
    // Recommended: When an exception occurs, record the HTTP status code, server-side error code, and
request ID returned by the server.
    System.out.println("HTTP Code: " + e.getResponseCode());
    System.out.println("Error Code:" + e.getErrorCode());
    System.out.println("Request ID:" + e.getErrorRequestId());
    // Recommended: When an exception occurs, record the stack information.
    e.printStackTrace(System.out);
}
```

NOTE

You can click **here** to view the details about **ObsException**.

Step 4 If an exception occurs when an ObsClient API is called, obtain the **HTTP status code** and **OBS server-side error code** from **ObsException** or log file, and compare them to locate the exception cause.

Step 5 If the exception cause cannot be found in step 4, obtain the request ID returned by the OBS server from **ObsException** or log file and contact the OBS server O&M team to locate the cause.

Step 6 If the request ID is unable to be obtained, collect the stack information of **ObsException** and contact the OBS client O&M team to locate the cause.

----End

5.2 Notable Issues

SignatureDoesNotMatch

```
HTTP Code: 403
Error Code: SignatureDoesNotMatch
```

Possible causes are as follows:

1. The SK input into ObsClient initialization is incorrect. Solution: Make sure that the SK is correct.
2. This problem is caused by a bug in the OBS Java SDK of an earlier version. Solution: Upgrade the SDK to the latest version.
3. OBS Java SDK 2.1.*x* versions are incompatible with the dependent library Apache HttpClient. Solution: Use the libraries of fixed versions: httpcore-4.4.4 and httpclient-4.5.3.

MethodNotAllowed

```
HTTP Code: 405
Error Code: MethodNotAllowed
```

This error occurs because a feature on which the ObsClient API depends has not been rolled out on the requested OBS server. Contact the OBS O&M team for further confirmation.

BucketAlreadyOwnedByYou

```
HTTP Code: 409
Error Code: BucketAlreadyOwnedByYou
```

In OBS, a bucket name must be globally unique. Solution: If this error occurs when the **ObsClient.createBucket** is called, check whether the bucket exists. You can use either of the following methods to check whether a bucket exists:

Method 1 (recommended): Call **ObsClient.listBuckets** to query the list of all buckets that you own and check whether the bucket exists.

Method 2: Call **ObsClient.headBucket** to check whether the bucket exists.

NOTE

ObsClient.headBucket can query only buckets in the current region, while **ObsClient.listBuckets** can query buckets in all regions.

BucketAlreadyExists

```
HTTP Code: 409
Error Code: BucketAlreadyExists
```

In OBS, a bucket name must be globally unique. Solution: If this error occurs when **ObsClient.createBucket** is called, it indicates that the bucket has been created by another user. Use another bucket name and try again.

Connection Timeout

```
HTTP Code: 408
Caused by: java.net.ConnectException: Connection timed out: connect
    at java.net.DualStackPlainSocketImpl.waitForConnect(Native Method)
    at java.net.DualStackPlainSocketImpl.socketConnect(DualStackPlainSocketImpl.java:85)
```

Possible causes are as follows:

1. The endpoint input into ObsClient initialization is incorrect. Solution: Verify to make sure that the endpoint is correct.
2. The network between the OBS client and OBS server is abnormal. Solution: Check the health status of the network.
3. The OBS domain name resolved by DNS is inaccessible. Solution: Contact the OBS O&M team.

Read/Write Timeout

```
HTTP Code: 408
Error Code:RequestTimeOut
Caused by: java.net.SocketTimeoutException: timeout
    at okio.Okio$4.newTimeoutException(Okio.java:232)
    at okio.AsyncTimeout.exit(AsyncTimeout.java:285)
    at okio.AsyncTimeout$2.read(AsyncTimeout.java:241)
```

Possible causes are as follows:

1. The network latency between the OBS client and OBS server is too long. Solution: Check the health status of the network.
2. The network between the OBS client and OBS server is abnormal. Solution: Check the health status of the network.

Abnormal Returned Value -1

```
HTTP Code: -1
```

Possible causes are as follows:

1. The OBS Java SDK of an earlier version is used and a connection timeout or read/write timeout occurs. Solution: See the solutions for **connection timeout** and **read/write timeout**.
2. This problem is caused by a bug in the OBS Java SDK of an earlier version. Solution: Download the latest SDK from **here**.
3. The server returns an abnormal result. As a result, an unexpected error occurs when the SDK resolves the returned result. Solution: Obtain the request ID returned by OBS server from the log and contact the OBS O&M team.

An Error Occurs During Program Startup After SDK Integration

Possible causes are as follows:

1. If the error ClassNotFoundException occurs during the program startup, it is usually caused by the missing of a third-party dependent library. Solution: Add the required third-party dependent library of the OBS Java SDK. See the following table.

Library Name	Version ID	Description
okhttp	3.11.0	Component for sending HTTP requests
okio	1.14.0	Component of okhttp
java-xmlbuilder	1.1	Component for constructing and parsing XML files
jackson-core	2.9.9	Component for constructing and parsing JSON files
jackson-databind	2.9.9	Component of jackson-core
jackson-annotations	2.9.9	Component of jackson-core

2. If the error NoClassDefFoundError occurs during the startup, it is usually caused by Java class conflict. Solution: a) Check whether a third-party library in the running environment contains multiple versions. b) Check whether the running environment contains the OBS Java SDK software package (esdk-obs-java-3.*x.x*.jar) of multiple versions.

Unable to Obtain Error Codes from ObsException

Possible causes are as follows:

1. An error is reported when **ObsClient.getBucketMetadata** or **ObsClient.getObjectMetadata** is called. In this scenario, the server does not return an error code because the request method used in the background is HEAD. Solution: Call **ObsException.getResponseCode** to obtain the HTTP status code to analyze the possible cause. For example, 403 indicates that the user does not have the access permission, and 404 indicates that the bucket or object does not exist. If the cause cannot be located, obtain the request ID returned by the OBS server from **ObsException** and contact the OBS O&M team.
2. The IP address of the endpoint obtained after DNS resolution during ObsClient initialization is not a valid IP address of the OBS server. Solution: Check whether the endpoint configuration is correct. If the endpoint configuration is correct, contact the OBS O&M team.

UnknownHostException

```
Caused by: java.net.UnknownHostException: bucketname.unknowndomain.com
    at java.net.Inet6AddressImpl.lookupAllHostAddr(Native Method)
    at java.net.InetAddress$1.lookupAllHostAddr(InetAddress.java:901)
    at java.net.InetAddress.getAddressesFromNameService(InetAddress.java:1293)
```

Possible causes are as follows:

1. The endpoint input during ObsClient initialization is incorrect. Solution: Verify to make sure that the endpoint is correct.
2. DNS cannot resolve the OBS domain name. Solution: Contact the OBS O&M team.

NullPointException

```
Exception in thread "main" java.lang.NullPointerException
    at com.obs.services.internal.RestStorageService.isCname(RestStorageService.java:1213)
    at com.obs.services.ObsClient.doActionWithResult(ObsClient.java:2805)
```

Possible causes are as follows:

1. **ObsClient.close** is called to close ObsClient and then another ObsClient API is called. Solution: Call **ObsClient.close** to release resources only before exiting the application.
2. This problem is caused by a bug in the OBS Java SDK of an earlier version. Solution: Download the latest SDK from **here**.

Connection Leakage

```
A connection to xxx was leaked. Did you forget to close a response body?
```

This error occurs when **ObsClient.getObject** is not properly closed after it is called to obtain the data flow of the object to be downloaded. Solution: Make sure that the **ObsObject.getObjectContent.close** method is called in the finally statement block to close the connection.

Problem in SDK Version Upgrade

The third-party dependent library of the SDK of an earlier version (2.1.*x*) is not completely compatible with that of the new version SDK (3.*x*). If a program startup error occurs after the earlier version is upgraded to the new version, see **An Error Occurs During Program Startup After SDK Integration**. If the problem persists, contact the OBS O&M team.

Others

For details, see **FAQs**.

6 Bucket Management

6.1 Creating a Bucket

You can call **ObsClient.createBucket** to create a bucket.

Creating a Bucket in Simple Mode

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Create a bucket.
try{
    // The bucket is successfully created.
    HeaderResponse response = obsClient.createBucket("bucketname");
    System.out.println(response.getRequestId());
}
catch (ObsException e)
{
    // Failed to create a bucket.
    System.out.println("HTTP Code: " + e.getResponseCode());
    System.out.println("Error Code:" + e.getErrorCode());
    System.out.println("Error Message: " + e.getErrorMessage());

    System.out.println("Request ID:" + e.getErrorRequestId());
    System.out.println("Host ID:" + e.getErrorHostId());
}
```

NOTE

- Bucket names are globally unique. Ensure that the bucket you create is named differently from any other bucket.
- A bucket name must comply with the following rules:
  - Contains 3 to 63 characters, chosen from lowercase letters, digits, hyphens (-), and periods (.), and starts with a digit or letter.
  - Cannot be an IP-like address.
  - Cannot start or end with a hyphen (-) or period (.)
  - Cannot contain two consecutive periods (.), for example, **my..bucket**.
  - Cannot contain periods (.) and hyphens (-) adjacent to each other, for example, **my-.bucket** or **my.-bucket**.
- If you create buckets of the same name in a region, no error will be reported and the bucket properties comply with those set in the first creation request.
- The bucket created in the previous example is of the default **ACL** (**private**), in the OBS Standard storage class, and in the default location where the global domain resides.

Creating a Bucket with Parameters Specified

When creating a bucket, you can specify the ACL, storage class, and location for the bucket. OBS provides three storage classes for buckets. For details, see **6.11 Setting or Obtaining the Storage Class of a Bucket**. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObsBucket obsBucket = new ObsBucket();
obsBucket.setBucketName("bucketname");
// Set the access permission for the bucket to public-read-write. (The default value is private.)
obsBucket.setAcl(AccessControlList.REST_CANNED_PUBLIC_READ);
// Set the storage class to OBS Cold.

obsBucket.setBucketStorageClass(StorageClassEnum.COLD);
// Set the location.
obsBucket.setLocation("bucketlocation");
// Create a bucket.
try{
    // The bucket is successfully created.
    HeaderResponse response = obsClient.createBucket(obsBucket);
    System.out.println(response.getRequestId());
}
catch (ObsException e)
{
    // Failed to create a bucket.
    System.out.println("HTTP Code: " + e.getResponseCode());
    System.out.println("Error Code:" + e.getErrorCode());
    System.out.println("Error Message: " + e.getErrorMessage());

    System.out.println("Request ID:" + e.getErrorRequestId());
    System.out.println("Host ID:" + e.getErrorHostId());
}
```

6.2 Listing Buckets

You can call **ObsClient.listBuckets** to list buckets. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// List buckets.
ListBucketsRequest request = new ListBucketsRequest();
request.setQueryLocation(true);
List<ObsBucket> buckets = obsClient.listBuckets(request);
for(ObsBucket bucket : buckets){
    System.out.println("BucketName:" + bucket.getBucketName());
    System.out.println("CreationDate:" + bucket.getCreationDate());
    System.out.println("Location:" + bucket.getLocation());
}
```

NOTE

- Obtained bucket names are listed in the lexicographical order.
- Set **ListBucketsRequest.setQueryLocation** to **true** and then you can query the bucket location when listing buckets.

6.3 Deleting a Bucket

You can call **ObsClient.deleteBucket** to delete a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Delete a bucket.
obsClient.deleteBucket("bucketname");
```

NOTE

- Only empty buckets (without objects and part fragments) can be deleted.
- Bucket deletion is a non-idempotence operation and an error will be reported if the to-be-deleted bucket does not exist.

6.4 Identifying Whether a Bucket Exists

You can call **ObsClient.headBucket** to identify whether a bucket exists. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

6.5 Obtaining Bucket Metadata

You can call **ObsClient.getBucketMetadata** to obtain the metadata of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketMetadataInfoRequest request = new BucketMetadataInfoRequest("bucketname");
request.setOrigin("http://www.a.com");
// Obtain the bucket metadata.
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());
```

NOTE

For details about values of methods, such as **BucketMetadataInfoResult.getAllowMethods**, see the **CORS** configurations of the bucket.

6.6 Managing Bucket ACLs

A bucket **ACL** can be configured in three modes:

1. Specify a pre-defined access control policy during bucket creation.
2. Call **ObsClient.setBucketAcl** to specify a pre-defined access control policy.
3. Call **ObsClient.setBucketAcl** to set the ACL directly.

The following table lists the five permission types supported by OBS.

Permission	Description	Value in OBS Java SDK
READ	A grantee with this permission for a bucket can obtain the list of objects in and metadata of the bucket. A grantee with this permission for an object can obtain the object content and metadata.	Permission.PERMISSION_READ
WRITE	A grantee with this permission for a bucket can upload, overwrite, and delete any object in the bucket. This permission is not applicable to objects.	Permission.PERMISSION_WRITE
READ_ACP	A grantee with this permission can obtain the ACL of a bucket or object. A bucket or object owner has this permission permanently.	Permission.PERMISSION_READ_ACP

Permission	Description	Value in OBS Java SDK
WRITE_ACP	A grantee with this permission can update the ACL of a bucket or object. A bucket or object owner has this permission permanently. A grantee with this permission can modify the access control policy and thus the grantee obtains full access permissions.	Permission.PERMISSION_WRITE_ACP
FULL_CONTROL	A grantee with this permission for a bucket has **READ**, **WRITE**, **READ_ACP**, and **WRITE_ACP** permissions for the bucket. A grantee with this permission for an object has **READ**, **WRITE**, **READ_ACP**, and **WRITE_ACP** permissions for the object.	Permission.PERMISSION_FULL_CONTROL

There are five access control policies pre-defined in OBS, as described in the following table:

Permission	Description	Value in OBS Java SDK
private	The owner of a bucket or object has the **FULL_CONTROL** permission for the bucket or object. Other users have no permission to access the bucket or object.	AccessControlList.REST_CANNED_PRIVATE
public-read	If this permission is set for a bucket, everyone can obtain the list of objects, multipart uploads, and object versions in the bucket, as well as metadata of the bucket. If this permission is set for an object, everyone can obtain the content and metadata of the object.	AccessControlList.REST_CANNED_PUBLIC_READ

Permission	Description	Value in OBS Java SDK
public-read-write	If this permission is set for a bucket, everyone can obtain the object list in the bucket, multipart uploads in the bucket, metadata of the bucket; upload objects; delete objects; initialize multipart uploads; upload parts; combine parts; copy parts; and abort multipart uploads. If this permission is set for an object, everyone can obtain the content and metadata of the object.	AccessControlList.REST_CANNED_PUBLIC_READ_WRITE
public-read-delivered	If this permission is set for a bucket, everyone can obtain the object list, multipart uploads, and bucket metadata in the bucket, and obtain the content and metadata of the objects in the bucket. This permission cannot be set for objects.	AccessControlList.REST_CANNED_PUBLIC_READ_DELIVERED
public-read-write-delivered	If this permission is set for a bucket, everyone can obtain the object list in the bucket, multipart uploads in the bucket, metadata of the bucket; upload objects; delete objects; initialize multipart uploads; upload parts; combine parts; copy parts; abort multipart uploads; obtain content and metadata of objects in the bucket. This permission cannot be set for objects.	AccessControlList.REST_CANNED_PUBLIC_READ_WRITE_DELIVERED

Specifying a Pre-defined Access Control Policy During Bucket Creation

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObsBucket obsBucket = new ObsBucket();
obsBucket.setBucketName("bucketname");
// Set the bucket ACL to public-read-write.
obsBucket.setAcl(AccessControlList.REST_CANNED_PUBLIC_READ_WRITE);
```

```
// Create a bucket.
obsClient.createBucket(obsBucket);
```

Setting a Pre-defined Access Control Policy for a Bucket

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Set the bucket ACL to private.
obsClient.setBucketAcl("bucketname", AccessControlList.REST_CANNED_PRIVATE);
```

Directly Setting a Bucket ACL

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
owner.setId("ownerid");
acl.setOwner(owner);
// Grant the FULL_CONTROL permission to a specified user.
acl.grantPermission(new CanonicalGrantee("userid"), Permission.PERMISSION_FULL_CONTROL);
// Grant the READ permission to all users.
acl.grantPermission(GroupGrantee.ALL_USERS, Permission.PERMISSION_READ);
// Directly set the bucket ACL.
obsClient.setBucketAcl("bucketname", acl);
```

NOTE

The owner or grantee ID needed in the ACL indicates the account ID, which can be viewed on the **My Credential** page of OBS Console.

Obtaining a Bucket ACL

You can call **ObsClient.getBucketAcl** to obtain the bucket ACL. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

6.7 Managing Bucket Policies

Besides bucket ACLs, bucket owners can use bucket policies to centrally control access to buckets and objects in buckets.

For more information, see **Bucket Policy Overview**.

Setting a Bucket Policy

You can call **ObsClient.setBucketPolicy** to set a bucket policy. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
obsClient.setBucketPolicy("bucketname", "your policy");
```

NOTE

For details about the format (JSON character string) of bucket policies, see the *Object Storage Service API Reference*.

Obtaining a Bucket Policy

You can call **ObsClient.getBucketPolicy** to obtain a bucket policy. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

String policy = obsClient.getBucketPolicy("bucketname");
System.out.println("\t" + policy);
```

Deleting a Bucket Policy

You can call **ObsClient.deleteBucketPolicy** to delete a bucket policy. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

6.8 Obtaining a Bucket Location

You can call **ObsClient.getBucketLocation** to obtain the location of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

NOTE

When creating a bucket, you can specify its location. For details, see **Creating a Bucket**.

6.9 Obtaining Storage Information About a Bucket

The storage information about a bucket includes the used capacity of and the number of objects in the bucket. You can call **ObsClient.getBucketStorageInfo** to obtain the bucket storage information. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

6.10 Setting or Obtaining a Bucket Quota

Setting a Bucket Quota

You can call **ObsClient.setBucketQuota** to set the bucket quota. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Set the bucket quota to 100 MB.
BucketQuota quota = new BucketQuota(1024 * 1024 * 100l);
obsClient.setBucketQuota("bucketname", quota);
```

NOTE

A bucket quota must be a non-negative integer expressed in bytes. The maximum value is 2^{63} - 1.

Obtaining a Bucket Quota

You can call **ObsClient.getBucketQuota** to obtain the bucket quota. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

6.11 Setting or Obtaining the Storage Class of a Bucket

OBS allows you to set storage classes for buckets. The storage class of an object defaults to be that of its residing bucket. Different storage classes meet different needs for storage performance and costs. There are three types of storage class for buckets, as described in the following table:

Storage Class	Description	Value in OBS Java SDK
OBS Standard	Features low access latency and high throughput and is applicable to storing frequently-accessed (multiple times per month) hotspot or small objects (< 1 MB) requiring quick response.	StorageClassEnum.STANDARD
OBS Warm	Is applicable to storing semi-frequently accessed (less than 12 times a year) data requiring quick response.	StorageClassEnum.WARM
OBS Cold	Is applicable to archiving rarely-accessed (once a year) data.	StorageClassEnum.COLD

For more information, see **Bucket Storage Classes**.

NOTE

The bucket storage class is independent from the storage classes of objects in the bucket. If the object storage class is not set during object upload, the object storage class is the same as that of the bucket. However, if the storage class of the bucket is changed, the storage class of the objects in the bucket does not change accordingly. If the storage class of an object in a bucket is changed, the storage class of the bucket does not change either.

Setting the Storage Class for a Bucket

You can call **ObsClient.setBucketStoragePolicy** to set the storage class for a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Set the storage class to OBS .
BucketStoragePolicyConfiguration storgePolicy = new BucketStoragePolicyConfiguration();
storgePolicy.setBucketStorageClass(StorageClassEnum.WARM);
obsClient.setBucketStoragePolicy("bucketname", storgePolicy);
```

Obtaining the Storage Class of a Bucket

You can call **ObsClient.getBucketStoragePolicy** to obtain the storage class of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
```

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

7 Object Upload

7.1 Object Upload Overview

In OBS, objects are basic data units that users can perform operations on. OBS Java SDK provides abundant APIs for object upload in the following methods:

- **7.2 Performing a Streaming Upload**
- **7.3 Performing a File-Based Upload**
- **7.7 Performing a Multipart Upload**
- **7.9 Performing an Appendable Upload**
- **7.10 Performing a Resumable Upload**
- **7.11 Performing a Browser-Based Upload**

The SDK supports the upload of objects whose size ranges from 0 KB to 5 GB. For streaming upload, appendable upload, and file-based upload, data to be uploaded cannot be larger than 5 GB. If the file is larger than 5 GB, multipart upload (where each part is smaller than 5 GB) is suitable. Browser-based upload allows files to be uploaded through a browser.

If the uploaded object can be read by anonymous users. After the upload succeeds, anonymous users can access the object data through the object URL. The object URL is in the format of **https://*bucket name.domain name*/*directory level*/*object name***. If the object resides in the root directory of the bucket, its URL does not contain directory levels.

7.2 Performing a Streaming Upload

Streaming upload uses **java.io.InputStream** as the data source of an object. You can call **ObsClient.putObject** to upload the data streams to OBS. Sample code is as follows:

Uploading a Character String (Byte Array)

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

Uploading a Network Stream

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

Uploading a File Stream

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

FileInputStream fis = new FileInputStream(new File("localfile"));  // localfile indicates the path of the local file to be uploaded. You need to specify the file name.
obsClient.putObject("bucketname", "objectname", fis);
```

NOTICE

- To upload a local file, you are advised to use **file-based upload**.
- To upload a large file, you are advised to use **multipart upload**.
- The content to be uploaded cannot exceed 5 GB.

7.3 Performing a File-Based Upload

File-based upload uses local files as the data source of objects. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.putObject("bucketname", "objectname", new File("localfile")); // localfile indicates the path of the local file to be uploaded. You need to specify the file name.
```

NOTE

The content to be uploaded cannot exceed 5 GB.

7.4 Obtaining Upload Progresses

You can call **PutObjectRequest.setProgressListener** to configure the data transmission API to obtain upload progresses. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

PutObjectRequest request = new PutObjectRequest("bucketname", "objectname");
request.setFile(new File("localfile"));
request.setProgressListener(new ProgressListener() {

    @Override
    public void progressChanged(ProgressStatus status) {
        // Obtain the average upload rate.
        System.out.println("AverageSpeed:" + status.getAverageSpeed());
        // Obtain the upload progress in percentage.
        System.out.println("TransferPercentage:" + status.getTransferPercentage());
    }
});
// Refresh the upload progress each time 1 MB data is uploaded.
request.setProgressInterval(1024 * 1024L);
obsClient.putObject(request);
```

NOTE

- You can query the upload progress when uploading an object in streaming, file-based, multipart, appendable, or resumable mode.
- If the value of **ProgressStatus.getTransferPercentage()** is **-1**, the content is uploaded in streaming mode. In this case, you must set the object length (**Content-Length**) in the object property.

7.5 Creating a Folder

There is no folder concept in OBS. All elements in buckets are objects. To create a folder in OBS is essentially to create an object whose size is 0 and whose name ends with a slash (/). Such objects have no difference from other objects and can be downloaded and deleted, except that they are displayed as folders in OBS Console.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

// In the folder, create an object.
obsClient.putObject("bucketname", keySuffixWithSlash + "objectname", new ByteArrayInputStream("Hello
OBS".getBytes()));
```

NOTE

- To create a folder in OBS is to create an object whose size is 0 and whose name ends with a slash (/), in essential.
- To create a multi-level folder, you only need to create the folder with the last level. For example, if you want to create a folder named **src1/src2/src3/**, create it directly, no matter whether the **src1/** and **src1/src2/** folders exist.

7.6 Setting Object Properties

You can set properties for an object when uploading it. Object properties include the object length, MIME type, MD5 value (for verification), storage class, and customized metadata. You can set properties for an object that is being uploaded in streaming, file-based, or multipart mode or when **copying the object**.

The following table describes object properties.

Property Name	Description	Default Value
Content-Length	Indicates the object length. If the object length exceeds the flow or file length, the object will be truncated.	Actual length of the stream or file
Content-Type	Indicates the MIME type of the object, which defines the type and network code of the object as well as in which mode and coding will the browser read the object.	application/octet-stream
Content-MD5	Indicates the base64-encoded digest of the object data. It is provided to the OBS server to verify data integrity.	None
Storage Class	Indicates the storage class of the object. Different storage classes meet different needs for storage performance and costs. The value defaults to be the same as the object's residing bucket and can be changed.	None
Customized metadata	Indicates the user-defined description of properties of the object uploaded to the bucket. It is used to facilitate the customized management on the object.	None

Setting the Length for an Object

You can call **ObjectMetadata.setContentLength** to set the length for an object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
```

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

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

Setting the MIME Type for an Object

You can call **ObjectMetadata.setContentType** to set the MIME type for an object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Upload an image.
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentType("image/jpeg");
obsClient.putObject("bucketname", "objectname.jpg", new File("localimage.jpg"), metadata);
```

NOTE

If this property is not specified, the SDK will automatically identify the MIME type according to the name suffix of the uploaded object. For example, if the name suffix of an object is **.xml** (**.html**), the object will be identified as an application/xml (text/html) file.

Setting the MD5 Value for an Object

You can call **ObjectMetadata.setContentMd5** to set the MD5 value for an object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Upload an image.
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentMd5("your md5 which should be encoded by base64");
obsClient.putObject("bucketname", "objectname", new File("localimage.jpg"), metadata);
```

NOTE

- The MD5 value of an object must be a base64-encoded digest.
- The OBS server will compare this MD5 value with the MD5 value obtained by object data calculation. If the two values are not the same, the upload fails with HTTP status code **400** returned.
- If the MD5 value is not specified, the OBS server will skip MD5 value verification.
- You can call **ObsClient.base64Md5** to calculate the **Content-MD5** header directly.

Setting the Storage Class for an Object

You can call **ObjectMetadata.setObjectStorageClass** to set the storage class for an object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
```

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

ObjectMetadata metadata = new ObjectMetadata();
// Set the storage class of the object to OBS Warm.
metadata.setObjectStorageClass(StorageClassEnum.WARM);
obsClient.putObject("bucketname", "objectname", new File("localfile"), metadata);
```

NOTE

- If you have not set the storage class for an object, the storage class of the object will be the same as that of its residing bucket.
- OBS provides objects with three storage classes which are consistent with the **storage classes** provided for buckets.
- Before downloading a Cold object, you must restore it.

Customizing Metadata for an Object

You can call **ObjectMetadata.addUserMetadata** to customize metadata for an object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
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", "objectname", new File("localfile"), metadata);
```

NOTE

- In the preceding code, two pieces of metadata named **property1** and **property2** are customized and their respective values are set to **property-value1** and **property-value2**.
- An object can have multiple pieces of metadata. The total metadata size cannot exceed 8 KB.
- The customized object metadata can be obtained by using **ObsClient.getObjectMetadata**. For details, see **Obtaining Object Metadata**.
- When you call **ObsClient.getObject** to download an object, its customized metadata will also be downloaded.

7.7 Performing a Multipart Upload

To upload a large file, multipart upload is recommended. Multipart upload is applicable to many scenarios, including:

- Files to be uploaded are larger than 100 MB.
- The network condition is poor. Connection to the OBS server is constantly down.
- Sizes of files to be uploaded are uncertain.

Multipart upload has the following advantages:

- Improving throughput: You can upload parts in parallel to improve throughput.

- Quick recovery from any network failures: Small-size parts can minimize the impact of failed uploading caused by network errors.
- Convenient suspension and resuming of object uploading: You can upload parts at any time. A multipart upload does not have a validity period. You must explicitly complete or cancel the multipart upload.
- Starting uploading before knowing the size of an object: You can upload an object while creating it.

Multipart upload consists of three phases:

Step 1 Initialize a multipart upload (**ObsClient.initiateMultipartUpload**).

Step 2 Upload parts one by one or concurrently (**ObsClient.uploadPart**).

Step 3 Combine parts (**ObsClient.completeMultipartUpload**) or abort the multipart upload (**ObsClient.abortMultipartUpload**).

----End

Initializing a Multipart Upload

Before upload, you need to notify OBS of initializing a multipart upload. This operation will return an upload ID (globally unique identifier) created by the OBS server to identify the multipart upload. You can use this upload ID to initiate related operations, such as aborting a multipart upload, listing multipart uploads, and listing uploaded parts.

You can call **ObsClient.initiateMultipartUpload** to initialize a multipart upload.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest("bucketname",
"objectname");
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);
```

NOTE

- Call **InitiateMultipartUploadRequest** to specify the name and owning bucket of the uploaded object.
- In **InitiateMultipartUploadRequest**, you can specify the MIME type, storage class, and customized metadata for the object.
- The upload ID of the multipart upload returned by **InitiateMultipartUploadResult.getUploadId** will be used in follow-up operations.

Uploading a Part

After initializing a multipart upload, you can specify the object name and upload ID to upload a part. Each upload part has a part number (ranging from 1 to

10000). For parts with the same upload ID, their part numbers are unique and identify their comparative locations in the object. If you use the same part number to upload two parts, the later one being uploaded will overwrite the former. Except for the part last uploaded whose size ranges from 0 to 5 GB, sizes of the other parts range from 100 KB to 5 GB. Parts are uploaded in random order and can be uploaded through different processes or machines. OBS will combine them into the object based on their part numbers.

You can call **ObsClient.uploadPart** to upload a part.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

List<PartEtag> partEtags = new ArrayList<PartEtag>();
// Upload the first part.
UploadPartRequest request = new UploadPartRequest("bucketname", "objectname");
// Set an upload ID.
request.setUploadId(uploadId);
// Set a part number, which ranges from 1 to 10000.
request.setPartNumber(1);
// Set the large file to be uploaded.
request.setFile(new File("localfile"));

// Set the part size.
request.setPartSize(5 * 1024 * 1024L);
UploadPartResult result = obsClient.uploadPart(request);
partEtags.add(new PartEtag(result.getEtag(), result.getPartNumber()));

// Upload the second part.
request = new UploadPartRequest("bucketname", "objectname");
// Set an upload ID.
request.setUploadId(uploadId);
// Set the part number.
request.setPartNumber(2);
// Set the large file to be uploaded.
request.setFile(new File("localfile"));
// Set the offset of the second part.
request.setOffset(5 * 1024 * 1024L);
// Set the part size.
request.setPartSize(5 * 1024 * 1024L);
result = obsClient.uploadPart(request);
partEtags.add(new PartEtag(result.getEtag(), result.getPartNumber()));
```

NOTE

- Except the part last uploaded, other parts must be larger than 100 KB. Part sizes will not be verified during upload because which one is last uploaded is not identified until parts are combined.
- OBS will return ETags (MD5 values) of the received parts to users.
- To ensure data integrity, set **UploadPartRequest.setAttachMd5** to **true** to make the SDK automatically calculate the MD5 value (valid only when the data source is a local file) of each part and add the MD5 value to the **Content-MD5** request header. The OBS server will compare the MD5 value contained by each part and that calculated by the SDK to verify the data integrity.
- You can call **UploadPartRequest.setContentMd5** to set the MD5 value of the uploaded data directly. If this value is set, the **UploadPartRequest.setAttachMd5** parameter becomes ineffective.
- Part numbers range from 1 to 10000. If the part number you set is out of this range, OBS will return error **400 Bad Request**.
- The minimum part size supported by an OBS 3.0 bucket is 100 KB, and the minimum part size supported by an OBS 2.0 bucket is 5 MB.

Combining Parts

After all parts are uploaded, call the API for combining parts to generate the object. Before this operation, valid part numbers and ETags of all parts must be sent to OBS. After receiving this information, OBS verifies the validity of each part one by one. After all parts pass the verification, OBS combines these parts to form the final object.

You can call **ObsClient.completeMultipartUpload** to combine parts.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

List<PartEtag> partEtags = new ArrayList<PartEtag>();
// First part
PartEtag part1 = new PartEtag();
part1.setPartNumber(1);
part1.seteTag("etag1");
partEtags.add(part1);

// Second part
PartEtag part2 = new PartEtag();
part2.setPartNumber(2);
part2.setEtag("etag2");
partEtags.add(part2);

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

obsClient.completeMultipartUpload(request);
```

NOTE

- In the preceding code, partEtags indicates the list of part numbers and ETags of uploaded parts.
- Part numbers can be inconsecutive.

Concurrently Uploading Parts

Multipart upload is mainly used for large file upload or when the network condition is poor. The following sample code shows how to concurrently upload parts in a multipart upload:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
final String bucketName = "bucketname";
final String objectKey = "objectname";
// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Initialize the thread pool.
ExecutorService executorService = Executors.newFixedThreadPool(20);
final File largeFile = new File("localfile");

// Initialize the multipart upload.
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(bucketName, objectKey);
InitiateMultipartUploadResult result = obsClient.initiateMultipartUpload(request);

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

// Set the part size to 100 MB.
long partSize = 100 * 1024 * 1024L;
long fileSize = largeFile.length();

// Calculate the number of parts need to be uploaded.
long partCount = fileSize % partSize == 0 ? fileSize / partSize : fileSize / partSize + 1;

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

// Start uploading parts concurrently.
for (int i = 0; i < partCount; i++)
{
    // Start position of parts in the file
    final long offset = i * partSize;
    // Part size
    final long currPartSize = (i + 1 == partCount) ? fileSize - offset : partSize;
    // Part number
    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();
            }
        }
    });
}

// Wait until the upload is complete.
executorService.shutdown();
while (!executorService.isTerminated())
{
    try
    {
        executorService.awaitTermination(5, TimeUnit.SECONDS);
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
}
// Combine parts.
CompleteMultipartUploadRequest completeMultipartUploadRequest = new
CompleteMultipartUploadRequest(bucketName, objectKey, uploadId, partEtags);
obsClient.completeMultipartUpload(completeMultipartUploadRequest);
```

NOTE

When uploading a large file, use **UploadPartRequest.setOffset** and **UploadPartRequest.setPartSize** to determine the start and end positions of each part.

Aborting a Multipart Upload

After a multipart upload is aborted, you cannot use its upload ID to perform any operation and the uploaded parts will be deleted by OBS.

When an object is being uploaded in multi-part mode or an object fails to be uploaded, parts are generated in the bucket. These parts occupy your storage space. You can cancel the multi-part uploading task to delete unnecessary parts, thereby saving the storage space.

You can call **ObsClient.abortMultipartUpload** to abort a multipart upload.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

obsClient.abortMultipartUpload(request);
```

Listing Uploaded Parts

You can call **ObsClient.listParts** to list successfully uploaded parts of a multipart upload.

The following table describes the parameters involved in this API.

Parameter	Description	Method in OBS Java SDK
bucketName	Bucket name	ListPartsRequest.setBucketName
key	Object name	ListPartsRequest.setKey
uploadId	Upload ID, which globally identifies a multipart upload. The value is in the returned result of **ObsClient.initiateMultipartUpload**.	ListPartsRequest.setUploadId
maxParts	Maximum number of parts that can be listed per page.	ListPartsRequest.setMaxParts
partNumberMarker	Part number after which listing uploaded parts begins. Only parts whose part numbers are larger than this value will be listed.	ListPartsRequest.setPartNumberMarker

- Listing parts in simple mode

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

//List the uploaded parts. uploadId is obtained frominitiateMultipartUpload.
ListPartsRequest request = new ListPartsRequest("bucketname", "objectname");
request.setUploadId(uploadId);
ListPartsResult result = obsClient.listParts(request);

for(Multipart part : result.getMultipartList()){
    // Part number, specified when uploading
   System.out.println("\t"+part.getPartNumber());
    // Part size
   System.out.println("\t"+part.getSize());
    // Part ETag
   System.out.println("\t"+part.getEtag());
    // Time when the part was last uploaded
   System.out.println("\t"+part.getLastModified());
}
```

NOTE

- Information about a maximum of 1,000 parts can be listed each time. If a task of the specific upload ID contains more than 1,000 parts and **ListPartsResult.isTruncated** is **true** in the returned result, not all parts are returned. In such cases, you can use **ListPartsResult.getNextPartNumberMarker** to obtain the start position for next listing.
- If you want to obtain all parts involved in a specific upload ID, you can use the paging mode for listing.

- Listing all parts

If the number of parts of a multipart upload is larger than 1,000, you can use the following sample code to list all parts.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// List the uploaded parts. uploadId is obtained from initiateMultipartUpload.
ListPartsRequest request = new ListPartsRequest("bucketname", "objectname");
request.setUploadId(uploadId);
ListPartsResult result;

do{
    result = obsClient.listParts(request);
    for(Multipart part : result.getMultipartList()){
    // Part number, specified when uploading
       System.out.println("\t"+part.getPartNumber());
    // Part size
       System.out.println("\t"+part.getSize());
    // Part ETag
       System.out.println("\t"+part.getEtag());
    // Time when the part was last uploaded
       System.out.println("\t"+part.getLastModified());
    }
    request.setPartNumberMarker(Integer.parseInt(result.getNextPartNumberMarker()));
}while(result.isTruncated());
```

Listing Multipart Uploads

You can call **ObsClient.listMultipartUploads** to list multipart uploads. The following table describes parameters involved in **ObsClient.listMultipartUploads**.

Parameter	Description	Method in OBS Java SDK
bucketName	Bucket name	ListMultipartUploadsRequest.setBucketName
prefix	Prefix that the object names in the multipart uploads to be listed must contain	ListMultipartUploadsRequest.setPrefix
delimiter	Character used to group object names involved in multipart uploads. If the object name contains the **delimiter** parameter, the character string from the first character to the first delimiter in the object name is grouped under a single result element, **commonPrefix**. (If a prefix is specified in the request, the prefix must be removed from the object name.)	ListMultipartUploadsRequest.setDelimiter

Parameter	Description	Method in OBS Java SDK
maxUploads	Maximum number of returned multipart uploads. The value ranges from 1 to 1000. If the value is not in this range, 1,000 multipart uploads are returned by default.	ListMultipartUploadsRequest.setMaxUploads
keyMarker	Object name to start with when listing multipart uploads	ListMultipartUploadsRequest.setKeyMarker
uploadIdMarker	Upload ID after which the multipart upload listing begins. It is effective only when used with **keyMarker** so that multipart uploads after **uploadIdMarker** of **keyMarker** will be listed.	ListMultipartUploadsRequest.setUploadIdMarker

- Listing multipart uploads in simple mode

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
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());
}
```

NOTE

- Information about a maximum of 1,000 multipart uploads can be listed each time. If a bucket contains more than 1,000 multipart uploads and **MultipartUploadListing.isTruncated** is **true**, not all uploads are listed. In such cases, you can use **MultipartUploadListing.getNextKeyMarker** and **MultipartUploadListing.getNextUploadIdMarker** to obtain the start position for next listing.
- If you want to obtain all multipart uploads in a bucket, you can list them in paging mode.

- Listing all multipart uploads in paging mode

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
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());
```

7.8 Configuring Lifecycle Management

When uploading an object or initializing a multipart upload, you can directly set the expiration time for the object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

PutObjectRequest request = new PutObjectRequest ("bucketname", "objectkey");
request.setFile(new File("localfile"));  // localfile indicates the path of the local file to be uploaded. You
need to specify the file name.
// When uploading an object, set the object to expire after 30 days.
request.setExpires(30);
obsClient.putObject(request);


InitiateMultipartUploadRequest request2 = new InitiateMultipartUploadRequest("bucketname",
"objectname");
// When initializing a multipart upload, set the object to expire 60 days after combination.
request2.setExpires(60);
obsClient.initiateMultipartUpload(request);
```

NOTE

- The previous mode specifies the time duration in days after which an object will expire. The OBS server automatically clears expired objects.
- The object expiration time set in the preceding method takes precedence over the bucket lifecycle rule.

7.9 Performing an Appendable Upload

Appendable upload allows you to upload an object in appendable mode and then append data to the object. You can call **ObsClient.appendObject** to perform an appendable upload. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Upload an object in appendable mode.
AppendObjectRequest request = new AppendObjectRequest();
request.setBucketName("bucketname");
request.setObjectKey("objectname");
request.setPosition(0);
request.setInput(new ByteArrayInputStream("Hello OBS".getBytes()));
AppendObjectResult result = obsClient.appendObject(request);

// Append data to the object.
```

```
request.setPosition(result.getNextPosition());
request.setInput(new ByteArrayInputStream("Hello OBS Again".getBytes()));
result = obsClient.appendObject(request);

System.out.println("NextPosition:" + result.getNextPosition());
System.out.println("Etag:" + result.getEtag());
// Use the API for obtaining object properties to get the start position for next appending.
ObjectMetadata metadata = obsClient.getObjectMetadata("bucketname", "objectname");
System.out.println("NextPosition from metadata:" + metadata.getNextPosition());
```

NOTE

- Objects uploaded using **ObsClient.putObject**, referred to as normal objects, can overwrite objects uploaded using **ObsClient.appendObject**, referred to as appendable objects. Data cannot be appended to an appendable object anymore once the object has been overwritten by a normal object.
- When you upload an object for the first time in appendable mode, an exception will be thrown (status code **409**) if a normal object with the same name exists.
- The ETag returned for an appendable upload is the ETag for the uploaded content, rather than that of the whole object.
- Data appended each time can be up to 5 GB, and 10,000 times of appendable uploads can be performed on a single object.
- After an appendable upload is successful, you can call **AppendObjectResult.getNextPosition** or use the **ObsClient.getObjectMetadata** API to get the start position for next appending.

7.10 Performing a Resumable Upload

Uploading large files often fails due to poor network conditions or program breakdowns. It is a waste of resources to restart the upload process upon an upload failure, and the restarted upload process may still suffer from the unstable network. To resolve such issues, you can use the API for resumable upload, whose working principle is to divide the to-be-uploaded file into multiple parts and upload them separately. The upload result of each part is recorded in a checkpoint file in real time. Only when all parts are successfully uploaded, the result indicating a successful upload will be returned. Otherwise, an exception is thrown to remind you of calling the API again for re-uploading. Based on the upload status of each part recorded in the checkpoint file, the re-uploading will upload the parts failed to be uploaded previously, instead of uploading all parts. By virtue of this, resources are saved and efficiency is improved.

You can call **ObsClient.uploadFile** to perform a resumable upload. The following table describes the parameters involved in this API.

Parameter	Description	Method in OBS Java SDK
bucketName	(Mandatory) Bucket name	UploadFileRequest.setBucketName
objectKey	(Mandatory) Object name	UploadFileRequest.setObjectKey
uploadFile	(Mandatory) Local file to be uploaded	UploadFileRequest.setUploadFile

Parameter	Description	Method in OBS Java SDK
partSize	Part size, in bytes. The value ranges from **100 KB** to **5 GB** and defaults to **9 MB**.	UploadFileRequest.setPartSize
taskNum	Maximum number of parts that can be concurrently uploaded. The default value is **1**.	UploadFileRequest.setTaskNum
enableCheckpoint	Whether to enable the resumable upload mode. The default value is **false**, which indicates that this mode is disabled.	UploadFileRequest.setEnableCheckpoint
checkpointFile	File used to record the upload progress. This parameter is effective only in the resumable upload mode. If the value of this parameter is **null**, the file will be in the same directory as the local file to be uploaded.	UploadFileRequest.setCheckpointFile
objectMetadata	Object properties	UploadFileRequest.setObjectMetadata
enableCheckSum	Whether to verify the content of the to-be-uploaded file. This parameter is effective only in the resumable upload mode. The default value is **false,** which indicates that the content will not be verified.	UploadFileRequest.setEnableCheckSum
progressListener	Configure the data transmission listener to obtain upload progresses.	UploadFileRequest.setProgressListener

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

UploadFileRequest request = new UploadFileRequest("bucketname", "obsjectKey");
// Set the large file to be uploaded. localfile is the path of the local file to be uploaded. You need to specify
the file name.
request.setUploadFile("localfile");
// Set the maximum number of parts that can be concurrently uploaded.
request.setTaskNum(5);
// Set the part size to 10 MB.
request.setPartSize(10 * 1024 * 1024);
// Enable resumable upload.
request.setEnableCheckpoint(true);
try{
    // Perform a resumable upload.
    CompleteMultipartUploadResult result = obsClient.uploadFile(request);
}catch (ObsException e) {
   // When an exception occurs, you can call the API for resumable upload again to perform re-uploading.
}
```

NOTE

- The API for resumable upload, which is implemented based on **multipart upload**, is an encapsulated and enhanced version of multipart upload.
- This API saves resources and improves efficiency upon the re-upload, and speeds up the upload process by concurrently uploading parts. Because this API is invisible to users, users are unaware of internal service details, such as the creation and deletion of checkpoint files, division of objects, and concurrent upload of parts.
- The default value of the **enableCheckpoint** parameter is **false**, which indicates that the resumable upload mode is disabled. In such cases, this API degrades to the simple encapsulation of multipart upload, and no checkpoint file will be generated.
- **checkpointFile** and **enableCheckSum** are effective only when **enableCheckpoint** is **true**.

7.11 Performing a Browser-Based Upload

Performing a browser-based upload is to upload objects to a specified bucket in HTML form. The maximum size of an object is 5 GB.

You can call **ObsClient.createPostSignature** to generate request parameters for a browser-based upload. You can use code to simulate a browser-based upload. For details, see **PostObjectSample**. You can also perform a browser-based upload as follows: The procedure is as follows:

Step 1 Call **ObsClient.createPostSignature** to generate request parameters for authentication.

Step 2 Prepare an HTML form page.

Step 3 Enter the request parameters in the HTML page.

Step 4 Select a local file to and upload it in browser-based mode.

----End

NOTE

There are two request parameters generated:

- **policy**, which corresponds to the **policy** field in the form
- **signature**, which corresponds to the **signature** field in the form

The following sample code shows how to generate the request parameters in a browser-based upload.

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

PostSignatureRequest request = new PostSignatureRequest();
// Fill in parameters in the form.
Map<String, Object> formParams = new HashMap<String, Object>();
// Set the object ACL to public-read.
```

```
formParams.put("x-obs-acl", "public-read");
// Set the MIME type for the object.
formParams.put("content-type", "text/plain");

request.setFormParams(formParams);
// Set the validity period for the browser-based upload request, in seconds.
request.setExpires(3600);
PostSignatureResponse response = obsClient.createPostSignature(request);

// Obtain the request parameters.
System.out.println("\t" + response.getPolicy());
System.out.println("\t" + response.getSignature());
```

Code of an HTML form example is as follows:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>

<form action="http://bucketname.your-endpoint/" method="post" enctype="multipart/form-data">
Object key
<!-- Object name -->
<input type="text" name="key" value="objectname" />
<p>
ACL
<!-- Object ACL -->
<input type="text" name="x-obs-acl" value="public-read" />
<p>
Content-Type
<!-- Object MIME type -->
<input type="text" name="content-type" value="text/plain" />
<p>
<!-- Base64 code of the policy -->
<input type="hidden" name="policy" value="*** Provide your policy ***" />
<!-- AK -->
<input type="hidden" name="AccessKeyId" value="*** Provide your access key ***"/>
<!-- Signature information -->
<input type="hidden" name="signature" value="*** Provide your signature ***"/>

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

NOTE

- Values of **policy** and **signature** in the HTML form are obtained from the returned result of **ObsClient.createPostSignature**.
- You can directly download the HTML form example: **PostDemo**.

8 Object Download

8.1 Object Download Overview

OBS Java SDK provides abundant APIs for object download in the following methods:

- **8.2 Performing a Streaming Download**
- **8.3 Performing a Partial Download**
- **8.9 Performing a Resumable Download**

You can call **ObsClient.getObject** to download an object.

8.2 Performing a Streaming Download

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObsObject obsObject = obsClient.getObject("bucketname", "objectname");

// Read the object contents.
System.out.println("Object content:");
InputStream input = obsObject.getObjectContent();
byte[] b = new byte[1024];
ByteArrayOutputStream bos = new ByteArrayOutputStream();
int len;
while ((len=input.read(b)) != -1){
      bos.write(b, 0, len);
}

System.out.println(new String(bos.toByteArray()));
bos.close();
input.close();
```

NOTE

- After **ObsClient.getObject** is called, an instance of **ObsObject** will be returned. This instance contains the residing bucket, name, properties, and input streams of the object.
- You can perform operations on the input streams of an object to read and write the object contents to a local file or to the memory.

NOTICE

Object input streams obtained by **ObsObject.getObjectContent** must be closed explicitly. Otherwise, resource leakage occurs.

8.3 Performing a Partial Download

When only partial data of an object is required, you can download data falling within a specific range. If the specified range is 0 to 1000, data at the 0th to the 1000th bytes, 1001 bytes in total, will be returned. If the specified range is invalid, data of the whole object will be returned. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

GetObjectRequest request = new GetObjectRequest("bucketname", "objectname");
// Specify the start and end positions.
request.setRangeStart(0l);
request.setRangeEnd(1000l);
ObsObject obsObject = obsClient.getObject(request);

// Obtain data.
byte[] buf = new byte[1024];
InputStream in = obsObject.getObjectContent();
for (int n = 0; n != -1; ) {
    n = in.read(buf, 0, buf.length);
}

in.close();
```

NOTE

- If the specified range is invalid (because the start or end position is set to a negative integer or the range is larger than the object length), data of the whole object will be returned.
- This download method also can be used to concurrently download parts of a large object. For details about the sample code, see **ConcurrentDownloadObjectSample**.

8.4 Obtaining Download Progresses

You can call **GetObjectRequest.setProgressInterval** to configure the data transmission interface to obtain download progresses. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
```

```
// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

GetObjectRequest request = new GetObjectRequest("bucketname", "objectname");
request.setProgressListener(new ProgressListener() {

    @Override
    public void progressChanged(ProgressStatus status) {
        // Obtain the average download rate.
        System.out.println("AverageSpeed:" + status.getAverageSpeed());
        // Obtain the download progress in percentage.
        System.out.println("TransferPercentage:" + status.getTransferPercentage());
    }
});
// Refresh the upload progress each time 1 MB data is uploaded.
request.setProgressInterval(1024 * 1024L);
ObsObject obsObject = obsClient.getObject(request);

// Read the object contents.
System.out.println("Object content:");
InputStream input = obsObject.getObjectContent();
byte[] b = new byte[1024];
ByteArrayOutputStream bos = new ByteArrayOutputStream();
int len;
while ((len=input.read(b)) != -1){
    bos.write(b, 0, len);
}

System.out.println(new String(bos.toByteArray()));
bos.close();
input.close();
```

NOTE

You can obtain the download progress when downloading an object in streaming, partial, or resumable mode.

8.5 Performing a Conditioned Download

When downloading an object, you can specify one or more conditions. Only when the conditions are met, the object will be downloaded. Otherwise, an exception will be thrown and the download will fail.

You can set the following conditions.

Parameter	Description	Method in OBS Java SDK
If-Modified-Since	Returns the object if it is modified after the time specified by this parameter; otherwise, an exception is thrown.	GetObjectRequest.setIfModifiedSince
If-Unmodified-Since	Returns the object if it remains unchanged since the time specified by this parameter; otherwise, an exception is thrown.	GetObjectRequest.setIfUnmodifiedSince

Parameter	Description	Method in OBS Java SDK
If-Match	Returns the source object if its ETag is the same as the one specified by this parameter; otherwise, an exception is thrown.	GetObjectRequest.setIfMatchTag
If-None-Match	Returns the source object if its ETag is different from the one specified by this parameter; otherwise, an exception is thrown.	GetObjectRequest.setIfNoneMatchTag

NOTE

- The ETag of an object is the MD5 check value of the object.
- If a request includes **If-Unmodified-Since** or **If-Match** and the specified condition is not met, **412 Precondition Failed** will be returned.
- If a request includes **If-Modified-Since** or **If-None-Match**, and the specified condition is not met, **304 Not Modified** will be returned.

Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

GetObjectRequest request = new GetObjectRequest("bucketname", "objectname");
request.setRangeStart(0l);
request.setRangeEnd(1000l);

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

obsObject.getObjectContent().close();
```

8.6 Rewriting Response Headers

When downloading an object, you can rewrite some HTTP/HTTPS response headers. The following table lists rewritable response headers.

Parameter	Description	Method in OBS Java SDK
contentType	Rewrites **Content-Type** in HTTP/HTTPS responses.	ObjectRepleaceMetadata.setContentType
contentLanguage	Rewrites **Content-Language** in HTTP/HTTPS responses.	ObjectRepleaceMetadata.setContentLanguage
expires	Rewrites **Expires** in HTTP/HTTPS responses.	ObjectRepleaceMetadata.setExpires

Parameter	Description	Method in OBS Java SDK
cacheControl	Rewrites **Cache-Control** in HTTP/HTTPS responses.	ObjectRepleaceMetadata.setCacheControl
contentDisposition	Rewrites **Content-Disposition** in HTTP/HTTPS responses.	ObjectRepleaceMetadata.setContentDisposition
contentEncoding	Rewrites **Content-Encoding** in HTTP/HTTPS responses.	ObjectRepleaceMetadata.setContentEncoding

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

ObsObject obsObject = obsClient.getObject(request);
System.out.println(obsObject.getMetadata().getContentType());

obsObject.getObjectContent().close();
```

8.7 Obtaining Customized Metadata

After an object is successfully downloaded, its customized data is returned. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Upload the object and customize the metadata.
PutObjectRequest request = new PutObjectRequest("bucketname", "objectname");
ObjectMetadata metadata = new ObjectMetadata();
metadata.addUserMetadata("property", "property-value");
request.setMetadata(metadata);
obsClient.putObject(request);

// Download the object and obtain the customized metadata.
ObsObject obsObject = obsClient.getObject("bucketname", "objectname");
System.out.println(obsObject.getMetadata().getUserMetadata("property"));

obsObject.getObjectContent().close();
```

8.8 Downloading a Cold Object

If you want to download a Cold object, you need to restore the object first. Two restore options are supported, as described in the following table:

Option	Description	Value in OBS Java SDK
Expedited	Data can be restored within 1 to 5 minutes.	RestoreTierEnum.EXPEDITED
Standard	Data can be restored within 3 to 5 hours. This is the default option.	RestoreTierEnum.STANDARD

You can call **ObsClient.restoreObject** to restore a Cold object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

RestoreObjectRequest request = new RestoreObjectRequest();
request.setBucketName("bucketname");
request.setObjectKey("objectname");
request.setDays(1);
request.setRestoreTier(RestoreTierEnum.EXPEDITED);
obsClient.restoreObject(request);

// Wait until the object is restored.
Thread.sleep(60 * 6 * 1000);

// Download an object.
ObsObject obsObject = obsClient.getObject("bucketname", "objectname");

obsObject.getObjectContent().close();
```

NOTE

- The object specified in **ObsClient.restoreObject** must be in the OBS Cold storage class. Otherwise, an exception will be thrown when you call this API.
- **RestoreObjectRequest.setDays** specifies the retention period of restored object, ranging from 1 to 30.
- **RestoreObjectRequest.setRestoreTier** specifies the restore option, which indicates the time spent on restoring an object.

8.9 Performing a Resumable Download

Downloading large files often fails due to poor network conditions or program breakdowns. It is a waste of resources to restart the download process upon a download failure, and the restarted download process may still suffer from the unstable network. To resolve such issues, you can use the API for resumable download, whose working principle is to divide the to-be-downloaded file into multiple parts and download them separately. The download result of each part is recorded in a checkpoint file in real time. Only when all parts are successfully downloaded, the result indicating a successful download will be returned. Otherwise, an exception is thrown to remind you of calling the API again for re-downloading. Based on the download status of each part recorded in the checkpoint file, the re-downloading will download the parts failed to be downloaded previously, instead of downloading all parts. By virtue of this, resources are saved and efficiency is improved.

You can call **ObsClient.downloadFile** to perform a resumable download. The following table describes the parameters involved in this API.

Parameter	Description	Method in OBS Java SDK
bucketName	(Mandatory) Bucket name	DownloadFileRequest.setBucketName
objectKey	(Mandatory) Object name	DownloadFileRequest.setObjectKey
downloadFile	Full path of the local directory to which the object is downloaded. If the value of this parameter is **null**, the downloaded object is saved in the directory where the program is executed.	DownloadFileRequest.setDownloadFile
partSize	Part size, in bytes. The value ranges from **100 KB** to **5 GB** and defaults to **9 MB**.	DownloadFileRequest.setPartSize
taskNum	Maximum number of parts that can be concurrently downloaded. The default value is **1**.	DownloadFileRequest.setTaskNum
enableCheckpoint	Whether to enable the resumable download mode. The default value is **false**, which indicates that this mode is disabled.	DownloadFileRequest.setEnableCheckpoint
checkpointFile	File used to record the download progress. This parameter is effective only in the resumable download mode. If the value of this parameter is **null**, the file will be in the same local directory as the downloaded object.	DownloadFileRequest.setCheckpointFile
versionId	Object version	DownloadFileRequest.setVersionId
ifModifiedSince	Returns the object if it is modified after the time specified by this parameter; otherwise, an exception is thrown.	DownloadFileRequest.setIfModifiedSince

Parameter	Description	Method in OBS Java SDK
ifUnmodifiedSince	Returns the object if it remains unchanged since the time specified by this parameter; otherwise, an exception is thrown.	DownloadFileRequest.setIfUnmodifiedSince
ifMatchTag	Returns the source object if its ETag is the same as the one specified by this parameter; otherwise, an exception is thrown.	DownloadFileRequest.setIfMatchTag
ifNoneMatchTag	Returns the source object if its ETag is different from the one specified by this parameter; otherwise, an exception is thrown.	DownloadFileRequest.setIfNoneMatchTag
progressListener	Configure the data transmission listener to obtain download progresses.	DownloadFileRequest.setProgressListener

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
DownloadFileRequest request = new DownloadFileRequest("bucketname", "objectname");
// Set the local path to which the object is downloaded.
request.setDownloadFile("localfile");
// Set the maximum number of parts that can be concurrently downloaded.
request.setTaskNum(5);
// Set the part size to 10 MB.
request.setPartSize(10 * 1024 * 1024);
// Enable resumable download.
request.setEnableCheckpoint(true);
try{
    // Perform a resumable download.
    DownloadFileResult result = obsClient.downloadFile(request);
}catch (ObsException e) {
    // When an exception occurs, you can call the API for resumable download again to perform re-
downloading.
}
```

NOTE

- The API for resumable download, which is implemented based on **partial download**, is an encapsulated and enhanced version of partial download.
- This API saves resources and improves efficiency upon the re-download, and speeds up the download process by concurrently downloading parts. Because this API is invisible to users, users are unaware of internal service details, such as the creation and deletion of checkpoint files, division of objects, and concurrent download of parts.
- The default value of the **enableCheckpoint** parameter is **false**, which indicates that the resumable download mode is disabled. In such cases, this API degrades to the simple encapsulation of partial download, and no checkpoint file will be generated.
- **checkpointFile** is effective only when **enableCheckpoint** is **true**.

9 Object Management

9.1 Obtaining Object Properties

You can call **ObsClient.getObjectMetadata** to obtain properties of an object, including the length, MIME type, customized metadata. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

9.2 Managing Object ACLs

Object ACLs, similar to bucket ACLs, support pre-defined access control policies and direct configuration. For details, see **Managing Bucket ACLs**.

An object **ACL** can be configured in three modes:

1. Specify a pre-defined access control policy during object upload.
2. Call **ObsClient.setObjectAcl** to specify a pre-defined access control policy.
3. Call **ObsClient.setObjectAcl** to set the ACL directly.

Specifying a Pre-defined Access Control Policy During Object Upload

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);
```

```
PutObjectRequest request = new PutObjectRequest();
request.setBucketName("bucketname");
request.setObjectKey("objectname");
request.setFile(new File("localfile"));
// Set the object ACL to public-read.
request.setAcl(AccessControlList.REST_CANNED_PUBLIC_READ);
obsClient.putObject(request);
```

Setting a Pre-defined Access Control Policy for an Object

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Set the object ACL to private.
obsClient.setObjectAcl("bucketname", "objectname", AccessControlList.REST_CANNED_PRIVATE);
```

Directly Setting an Object ACL

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
owner.setId("ownerid");
acl.setOwner(owner);
// Grant the FULL_CONTROL permission to a specified user.
acl.grantPermission(new CanonicalGrantee("userid"), Permission.PERMISSION_FULL_CONTROL);
// Grant the READ permission to all users.
acl.grantPermission(GroupGrantee.ALL_USERS, Permission.PERMISSION_READ);
obsClient.setObjectAcl("bucketname", "objectname", acl);
```

NOTE

The owner or grantee ID needed in the ACL indicates the account ID, which can be viewed on the **My Credentials** page of OBS Console.

Obtaining an Object ACL

You can call **ObsClient.getObjectAcl** to obtain an object ACL. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

9.3 Listing Objects

You can call **ObsClient.listObjects** to list objects in a bucket.

The following table describes the parameters involved in this API.

Parameter	Description	Method in OBS Java SDK
bucketName	Bucket name	ListObjectsRequest.setBucketName
prefix	Name prefix that the objects to be listed must contain	ListObjectsRequest.setPrefix
marker	Object name to start with when listing objects in a bucket. All objects are listed in the lexicographical order.	ListObjectsRequest.setMarker
maxKeys	Maximum number of objects returned in the response. The value ranges from 1 to 1000. If the value is not in this range, 1000 objects are returned by default.	ListObjectsRequest.setMaxKeys
delimiter	Character used to group object names. If the object name contains the **delimiter** parameter, the character string from the first character to the first delimiter in the object name is grouped under a single result element, **commonPrefix**. (If a prefix is specified in the request, the prefix must be removed from the object name.)	ListObjectsRequest.setDelimiter

Listing Objects in Simple Mode

The following sample code shows how to list objects in simple mode. A maximum of 1000 objects can be listed.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObjectListing result = obsClient.listObjects("bucketname");
for(ObsObject obsObject : result.getObjects()){
    System.out.println("\t" + obsObject.getObjectKey());
    System.out.println("\t" + obsObject.getOwner());
}
```

NOTE

- A maximum of 1000 objects can be listed each time. If a bucket contains more than 1000 objects and **ObjectListing.isTruncated** is **true** in the returned result, not all objects are listed. In such cases, you can use **ObjectListing.getNextMarker** to obtain the start position for next listing.
- If you want to obtain all objects in a specified bucket, you can use the paging mode for listing objects.

Listing Objects by Specifying the Number

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// Specify the number of objects to be listed to 100.
request.setMaxKeys(100);
ObjectListing result = obsClient.listObjects(request);
for(ObsObject obsObject : result.getObjects()){
    System.out.println("\t" + obsObject.getObjectKey());
    System.out.println("\t" + obsObject.getOwner());
}
```

Listing Objects by Specifying a Prefix

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// Set the number to 100 and the prefix to prefix.
request.setMaxKeys(100);
request.setPrefix("prefix");
ObjectListing result = obsClient.listObjects(request);
for(ObsObject obsObject : result.getObjects()){
    System.out.println("\t" + obsObject.getObjectKey());
    System.out.println("\t" + obsObject.getOwner());
}
```

Listing Objects by Specifying the Start Position

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// List 100 objects following test in lexicographic order.
request.setMaxKeys(100);
request.setMarker("test");
```

```
ObjectListing result = obsClient.listObjects(request);
for(ObsObject obsObject : result.getObjects()){
    System.out.println("\t" + obsObject.getObjectKey());
    System.out.println("\t" + obsObject.getOwner());
}
```

Listing All Objects in Paging Mode

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// Set the number of objects displayed per page to 100.
request.setMaxKeys(100);

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

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

Listing All Objects in a Folder

There is no folder concept in OBS. All elements in buckets are objects. Folders are actually objects whose sizes are 0 and whose names end with a slash (/). When you set a folder name as the prefix, objects in this folder will be listed. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// Set the prefix of objects in the folder to dir/.
request.setPrefix("dir/");
request.setMaxKeys(1000);

ObjectListing result;

do{
    result = obsClient.listObjects(request);
    for (ObsObject obsObject : result.getObjects())
    {
        System.out.println("\t" + obsObject.getObjectKey());
        System.out.println("\t" + obsObject.getOwner());
    }
    request.setMarker(result.getNextMarker());
}while(result.isTruncated());
```

Listing All Objects According to Folders in a Bucket

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
request.setMaxKeys(1000);
// Set folder isolators to slashes.
request.setDelimiter("/");
ObjectListing result = obsClient.listObjects(request);
System.out.println("Objects in the root directory:");
for(ObsObject obsObject : result.getObjects()){
    System.out.println("\t" + obsObject.getObjectKey());
    System.out.println("\t" + obsObject.getOwner());
}
listObjectsByPrefix(obsClient, request, result);
```

The following is the sample code of the **listObjectsByPrefix** function, which is used to recursively list objects in sub-folders.

```
static void listObjectsByPrefix(ObsClient obsClient, ListObjectsRequest request, ObjectListing result) throws
ObsException
{
      for(String prefix : result.getCommonPrefixes()){
         System.out.println("Objects in folder [" + prefix + "]:");
         request.setPrefix(prefix);
         result  = obsClient.listObjects(request);
         for(ObsObject obsObject : result.getObjects()){
             System.out.println("\t" + obsObject.getObjectKey());
             System.out.println("\t" + obsObject.getOwner());
         }
         listObjectsByPrefix(obsClient, request, result);
     }
}
```

NOTE

- The sample code does not apply to scenarios where the number of objects in a folder exceeds 1,000.
- Because objects and sub-folders in a folder are to be listed and all the objects end with a slash (/), **delimiter** is always a slash (/).
- In the returned result of each recursion, **ObjectListing.getObjects** includes the objects in the folder and **ObjectListing.getCommonPrefixes** includes the sub-folders in the folder.

9.4 Deleting Objects

NOTE

Exercise caution when performing this operation. If the versioning function is disabled for the bucket where the object is located, the object cannot be restored after being deleted.

Deleting a Single Object

You can call **ObsClient.deleteObject** to delete a single object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
```

```
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
obsClient.deleteObject("bucketname", "objectname");
```

Deleting Objects in a Batch

You can call **ObsClient.deleteObjects** to delete objects in a batch.

A maximum of 1,000 objects can be deleted each time. Two response modes are supported: verbose (detailed) and quiet (brief).

- In verbose mode (default mode), the returned response includes the deletion result of each requested object.
- In quiet mode, the returned response includes only results of objects failed to be deleted.

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsRequest request = new ListVersionsRequest("bucketname");
// Delete 100 objects at a time.
request.setMaxKeys(100);
ListVersionsResult result;
do {
    result = obsClient.listVersions(request);

    DeleteObjectsRequest deleteRequest = new DeleteObjectsRequest("bucketname");

    for(VersionOrDeleteMarker v  : result.getVersions()) {
        deleteRequest.addKeyAndVersion(v.getKey(), v.getVersionId());
    }

    DeleteObjectsResult deleteResult = obsClient.deleteObjects(deleteRequest);
// Obtain the list of successfully deleted objects.
    System.out.println(deleteResult.getDeletedObjectResults());
// Obtain the list of objects failed to be deleted.
    System.out.println(deleteResult.getErrorResults());

    request.setKeyMarker(result.getNextKeyMarker());
    request.setVersionIdMarker(result.getNextVersionIdMarker());
}while(result.isTruncated());
```

9.5 Copying an Object

The object copy operation creates a copy for an existing object in OBS.

You can call **ObsClient.copyObject** to copy an object. When copying an object, you can rewrite properties and ACL for it, as well as set restriction conditions.

Constraints

- The user has the read permission on the source object to be copied.
- Cross-region replication is not supported.

- The source object to be copied cannot be larger than 5 GB. If the size is less than 1 GB, you are advised to copy it directly. If the size is greater than 1 GB, you are advised to perform a multipart copy.
- If the source object to be copied is in the Cold storage class, you must restore it first.

Copying an Object Directly

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

Rewriting Object Properties

The following sample code shows how to rewrite object properties.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectRequest request = new CopyObjectRequest("sourcebucketname", "sourceobjectname",
"destbucketname", "destobjectname");
// Rewrite object properties.
request.setReplaceMetadata(true);
ObjectMetadata newObjectMetadata = new ObjectMetadata();
newObjectMetadata.setContentType("image/jpeg");
newObjectMetadata.addUserMetadata("property", "property-value");
newObjectMetadata.setObjectStorageClass(StorageClassEnum.WARM);
request.setNewObjectMetadata(newObjectMetadata);
CopyObjectResult result = obsClient.copyObject(request);
System.out.println("\t" + result.getEtag());
```

NOTE

CopyObjectRequest.setReplaceMetadata and **CopyObjectRequest.setNewObjectMetadata** must be used together.

Copying an Object by Specifying Conditions

When copying an object, you can specify one or more restriction conditions. If the conditions are met, the object will be copied. Otherwise, an exception will be thrown and the copy will fail.

You can set the following conditions.

Parameter	Description	Method in OBS Java SDK
Copy-Source-If-Modified-Since	Copies the source object if it is changed after the time specified by this parameter; otherwise, an exception is thrown.	CopyObjectRequest.setIfModifiedSince
Copy-Source-If-Unmodified-Since	Copies the source object if it is changed before the time specified by this parameter; otherwise, an exception is thrown.	CopyObjectRequest.setIfUnmodifiedSince
Copy-Source-If-Match	Copies the source object if its ETag is the same as the one specified by this parameter; otherwise, an exception is thrown.	CopyObjectRequest.setIfMatchTag
Copy-Source-If-None-Match	Copies the source object if its ETag is different from the one specified by this parameter; otherwise, an exception is thrown.	CopyObjectRequest.setIfNoneMatchTag

NOTE

- The ETag of the source object is the MD5 check value of the source object.
- If **Copy-Source-If-Unmodified-Since**, **Copy-Source-If-Match**, **Copy-Source-If-Modified-Since**, or **Copy-Source-If-None-Match** is included and its specified condition is not met, an exception, whose HTTP status code is **412 Precondition Failed**, will be thrown.
- **Copy-Source-If-Modified-Since** and **Copy-Source-If-None-Match** can be used together, and so do **Copy-Source-If-Unmodified-Since** and **Copy-Source-If-Match**.

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectRequest request = new CopyObjectRequest("sourcebucketname", "sourceobjectname",
"destbucketname", "destobjectname");

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());
```

Rewriting an Object ACL

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectRequest request = new CopyObjectRequest("sourcebucketname", "sourceobjectname",
"destbucketname", "destobjectname");

// Modify the Object ACL to public-read.
request.setAcl(AccessControlList.REST_CANNED_PUBLIC_READ);
CopyObjectResult result = obsClient.copyObject(request);
System.out.println("\t" + result.getEtag());
```

Performing a Multipart Copy

As a special case of multipart upload, multipart copy implements multipart upload by copying the whole or part of an object in a bucket. You can call **ObsClient.copyPart** to copy parts. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

final String destBucketName = "destbucketname";
final String destObjectKey = "destobjectname";
final String sourceBucketName = "sourcebucketname";
final String sourceObjectKey = "sourceobjectname";
// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Initialize the thread pool.
ExecutorService executorService = Executors.newFixedThreadPool(20);

// Initialize the multipart upload.
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(destBucketName,
destObjectKey);
InitiateMultipartUploadResult result = obsClient.initiateMultipartUpload(request);

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

// Obtain information about the large object.
ObjectMetadata metadata = obsClient.getObjectMetadata(sourceBucketName, sourceObjectKey);
// Set the part size to 100 MB.
long partSize = 100 * 1024 * 1024L;
long objectSize = metadata.getContentLength();

// Calculate the number of parts need to be copied.
long partCount = objectSize % partSize == 0 ? objectSize / partSize : objectSize / partSize + 1;

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

// Start copying parts concurrently.
for (int i = 0; i < partCount; i++)
{
// Start position for copying parts
    final long rangeStart = i * partSize;
// End position for copying parts
    final long rangeEnd = (i + 1 == partCount) ? objectSize - 1 : rangeStart + partSize - 1;
    // Part number
    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();
            }
        }
    });
}

// Wait until the copy is complete.
executorService.shutdown();
while (!executorService.isTerminated())
{
    try
    {
        executorService.awaitTermination(5, TimeUnit.SECONDS);
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
}

// Combine parts.
CompleteMultipartUploadRequest completeMultipartUploadRequest = new
CompleteMultipartUploadRequest(destBucketName, destObjectKey, uploadId, partEtags);
obsClient.completeMultipartUpload(completeMultipartUploadRequest);
```

10 Authorized Access

10.1 Using a URL for Authorized Access

ObsClient allows you to create a URL whose **Query** parameters are carried with authentication information by specifying the AK and SK, HTTP method, and request parameters. You can provide other users with this URL for temporary access. When generating a URL, you need to specify the validity period of the URL to restrict the access duration of visitors.

If you want to grant other users the permission to perform other operations on buckets or objects (for example, upload or download objects), generate a URL with the corresponding request (for example, to upload an object using the URL that generates the PUT request) and provide the URL for other users.

The following table lists operations can be performed through a signed URL.

Operation	HTTP Request Method (Value in OBS Java SDK)	Special Operator (Value in OBS Java SDK)	Bucket Name Required	Object Name Required
PUT Bucket	HttpMethodEnum.PUT	N/A	Yes	No
GET Buckets	HttpMethodEnum.GET	N/A	No	No
DELETE Bucket	HttpMethodEnum.DELETE	N/A	Yes	No
GET Objects	HttpMethodEnum.GET	N/A	Yes	No
GET Object versions	HttpMethodEnum.GET	SpecialParamEnum.VERSIONS	Yes	No

Operation	HTTP Request Method (Value in OBS Java SDK)	Special Operator (Value in OBS Java SDK)	Bucket Name Required	Object Name Required
List Multipart uploads	HttpMethodEnum.GET	SpecialParamEnum.UPLOADS	Yes	No
Obtain Bucket Metadata	HttpMethodEnum.HEAD	N/A	Yes	No
GET Bucket location	HttpMethodEnum.GET	SpecialParamEnum.LOCATION	Yes	No
GET Bucket storageinfo	HttpMethodEnum.GET	SpecialParamEnum.STORAGEINFO	Yes	No
PUT Bucket quota	HttpMethodEnum.PUT	SpecialParamEnum.QUOTA	Yes	No
GET Bucket quota	HttpMethodEnum.GET	SpecialParamEnum.QUOTA	Yes	No
PUT Bucket storagePolicy	HttpMethodEnum.PUT	SpecialParamEnum.STORAGEPOLICY	Yes	No
GET Bucket storagePolicy	HttpMethodEnum.GET	SpecialParamEnum.STORAGEPOLICY	Yes	No
PUT Bucket acl	HttpMethodEnum.PUT	SpecialParamEnum.ACL	Yes	No
GET Bucket acl	HttpMethodEnum.GET	SpecialParamEnum.ACL	Yes	No
PUT Bucket logging	HttpMethodEnum.PUT	SpecialParamEnum.LOGGING	Yes	No

Operation	HTTP Request Method (Value in OBS Java SDK)	Special Operator (Value in OBS Java SDK)	Bucket Name Required	Object Name Required
GET Bucket logging	HttpMethodEnum.GET	SpecialParamEnum.LOGGING	Yes	No
PUT Bucket policy	HttpMethodEnum.PUT	SpecialParamEnum.POLICY	Yes	No
GET Bucket policy	HttpMethodEnum.GET	SpecialParamEnum.POLICY	Yes	No
DELETE Bucket policy	HttpMethodEnum.DELETE	SpecialParamEnum.POLICY	Yes	No
PUT Bucket lifecycle	HttpMethodEnum.PUT	SpecialParamEnum.LIFECYCLE	Yes	No
GET Bucket lifecycle	HttpMethodEnum.GET	SpecialParamEnum.LIFECYCLE	Yes	No
DELETE Bucket lifecycle	HttpMethodEnum.DELETE	SpecialParamEnum.LIFECYCLE	Yes	No
PUT Bucket website	HttpMethodEnum.PUT	SpecialParamEnum.WEBSITE	Yes	No
GET Bucket website	HttpMethodEnum.GET	SpecialParamEnum.WEBSITE	Yes	No
DELETE Bucket website	HttpMethodEnum.DELETE	SpecialParamEnum.WEBSITE	Yes	No
PUT Bucket versioning	HttpMethodEnum.PUT	SpecialParamEnum.VERSIONING	Yes	No

Operation	HTTP Request Method (Value in OBS Java SDK)	Special Operator (Value in OBS Java SDK)	Bucket Name Required	Object Name Required
GET Bucket versioning	HttpMethodEnum.GET	SpecialParamEnum.VERSIONING	Yes	No
PUT Bucket cors	HttpMethodEnum.PUT	SpecialParamEnum.CORS	Yes	No
GET Bucket cors	HttpMethodEnum.GET	SpecialParamEnum.CORS	Yes	No
DELETE Bucket cors	HttpMethodEnum.DELETE	SpecialParamEnum.CORS	Yes	No
PUT Bucket notification	HttpMethodEnum.PUT	SpecialParamEnum.NOTIFICATION	Yes	No
GET Bucket notification	HttpMethodEnum.GET	SpecialParamEnum.NOTIFICATION	Yes	No
PUT Object	HttpMethodEnum.PUT	N/A	Yes	Yes
Append Object	HttpMethodEnum.POST	SpecialParamEnum.APPEND	Yes	Yes
GET Object	HttpMethodEnum.GET	N/A	Yes	Yes
PUT Object - Copy	HttpMethodEnum.PUT	N/A	Yes	Yes
DELETE Object	HttpMethodEnum.DELETE	N/A	Yes	Yes
DELETE Objects	HttpMethodEnum.POST	SpecialParamEnum.DELETE	Yes	Yes
Obtain Object Metadata	HttpMethodEnum.HEAD	N/A	Yes	Yes

Operation	HTTP Request Method (Value in OBS Java SDK)	Special Operator (Value in OBS Java SDK)	Bucket Name Required	Object Name Required
PUT Object acl	HttpMethodEnum.PUT	SpecialParamEnum.ACL	Yes	Yes
GET Object acl	HttpMethodEnum.GET	SpecialParamEnum.ACL	Yes	Yes
Initiate Multipart Upload	HttpMethodEnum.POST	SpecialParamEnum.UPLOADS	Yes	Yes
PUT Part	HttpMethodEnum.PUT	N/A	Yes	Yes
PUT Part - Copy	HttpMethodEnum.PUT	N/A	Yes	Yes
List Parts	HttpMethodEnum.GET	N/A	Yes	Yes
Complete Multipart Upload	HttpMethodEnum.POST	N/A	Yes	Yes
DELETE Multipart Upload	HttpMethodEnum.DELETE	N/A	Yes	Yes
POST Object restore	HttpMethodEnum.POST	SpecialParamEnum.RESTORE	Yes	Yes

To use a URL for authorized access, perform the following two steps:

Step 1 Call **ObsClient.createTemporarySignature** to generate a signed URL.

Step 2 Use any HTTP library to make an HTTP/HTTPS request to OBS.

----End

The following code provides an example showing how to use a URL for authorized access, including bucket creation, as well as object upload, download, listing, and deletion.

Creating a Bucket

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;

TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.PUT,
expireSeconds);
request.setBucketName("bucketname");
TemporarySignatureResponse response = obsClient.createTemporarySignature(request);
System.out.println("Creating bucket using temporary signature url:");
System.out.println("\t" + response.getSignedUrl());

Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
      builder.header(entry.getKey(), entry.getValue());
}
// Make a PUT request to create a bucket.
String location = "your bucket location";
Request httpRequest = builder.url(response.getSignedUrl()).put(RequestBody.create(null,
"<CreateBucketConfiguration><LocationConstraint>" + location + "</LocationConstraint></
CreateBucketConfiguration>".getBytes())).build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
             .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
System.out.println("\tStatus:" + res.code());
if (res.body() != null) {
      System.out.println("\tContent:" + res.body().string() + "\n");
}
res.close();
```

Uploading an Object

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;

Map<String, String> headers = new HashMap<String, String>();
String contentType = "text/plain";
headers.put("Content-Type", contentType);

TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.PUT,
expireSeconds);
request.setBucketName("bucketname");
request.setObjectKey("objectname");
request.setHeaders(headers);

TemporarySignatureResponse response = obsClient.createTemporarySignature(request);

System.out.println("Creating object using temporary signature url:");
System.out.println("\t" + response.getSignedUrl());
Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
      builder.header(entry.getKey(), entry.getValue());
}
```

```
//Make a PUT request to upload an object.
Request httpRequest =
builder.url(response.getSignedUrl()).put(RequestBody.create(MediaType.parse(contentType), "Hello
OBS".getBytes("UTF-8"))).build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
            .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
System.out.println("\tStatus:" + res.code());
if (res.body() != null) {
      System.out.println("\tContent:" + res.body().string() + "\n");
}
res.close();
```

Downloading an Object

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;


TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.GET,
expireSeconds);
request.setBucketName("bucketname");
request.setObjectKey("objectname");

TemporarySignatureResponse response = obsClient.createTemporarySignature(request);

System.out.println("Getting object using temporary signature url:");
System.out.println("\t" + response.getSignedUrl());
Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
      builder.header(entry.getKey(), entry.getValue());
}

//Make a GET request to download an object.
Request httpRequest = builder.url(response.getSignedUrl()).get().build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
            .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
System.out.println("\tStatus:" + res.code());
if (res.body() != null) {
      System.out.println("\tContent:" + res.body().string() + "\n");
}
res.close();
```

Listing Objects

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;


TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.GET,
expireSeconds);
request.setBucketName("bucketname");

TemporarySignatureResponse response = obsClient.createTemporarySignature(request);

System.out.println("Getting object list using temporary signature url:");
System.out.println("\t" + response.getSignedUrl());
Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
       builder.header(entry.getKey(), entry.getValue());
}

//Make a GET request to obtain the object list.
Request httpRequest = builder.url(response.getSignedUrl()).get().build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
             .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
System.out.println("\tStatus:" + res.code());
if (res.body() != null) {
       System.out.println("\tContent:" + res.body().string() + "\n");
}
res.close();
```

Deleting an Object

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;


TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.DELETE,
expireSeconds);
request.setBucketName("bucketname");
request.setObjectKey("objectname");

TemporarySignatureResponse response = obsClient.createTemporarySignature(request);

System.out.println("Deleting object using temporary signature url:");
System.out.println("\t" + response.getSignedUrl());
Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
       builder.header(entry.getKey(), entry.getValue());
}

//Make a DELETE request to delete an object.
Request httpRequest = builder.url(response.getSignedUrl()).delete().build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
             .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
System.out.println("\tStatus:" + res.code());
if (res.body() != null) {
       System.out.println("\tContent:" + res.body().string() + "\n");
}
res.close();
```

Initiating Multipart Upload

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;

TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.POST,
expireSeconds);
request.setBucketName("bucketname");
request.setObjectKey("objectname");
request.setSpecialParam(SpecialParamEnum.UPLOADS);

TemporarySignatureResponse response = obsClient.createTemporarySignature(request);

System.out.println("initiate multipart upload using temporary signature url:");
System.out.println("\t" + response.getSignedUrl());

Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
      builder.header(entry.getKey(), entry.getValue());
}

// POST a request to initialize a multipart upload.
Request httpRequest = builder.url(response.getSignedUrl()).post(RequestBody.create(null, "")).build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
            .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
System.out.println("\tStatus:" + res.code());
if (res.body() != null) {
      System.out.println("\tContent:" + res.body().string() + "\n");
}
res.close();
```

Uploading Parts

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;

TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.POST,
expireSeconds);
request.setBucketName("bucketname");
request.setObjectKey("objectname");

Map<String, Object> queryParams = new HashMap<String, Object>();
// Set the partNumber parameter, for example, queryParams.put("partNumber", "1").
queryParams.put("partNumber", "partNumber");
queryParams.put("uploadId", "your uploadId");

request.setQueryParams(queryParams);

TemporarySignatureResponse response = obsClient.createTemporarySignature(request);

System.out.println("upload part using temporary signature url:");
System.out.println("\t" + response.getSignedUrl());
```

```
Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
      builder.header(entry.getKey(), entry.getValue());
}

// PUT a request to upload object parts.
Request httpRequest = builder.url(response.getSignedUrl()).put(RequestBody.create(null, new byte[6 * 1024
* 1024])).build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
            .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
System.out.println("\tStatus:" + res.code());
if (res.body() != null) {
      System.out.println("\tContent:" + res.body().string() + "\n");
}
res.close();
```

Listing Uploaded Parts

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;

TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.GET,
expireSeconds);
request.setBucketName("bucketname");
request.setObjectKey("objectname");

Map<String, Object> queryParams = new HashMap<String, Object>();
queryParams.put("uploadId", "your uploadId");
request.setQueryParams(queryParams);

TemporarySignatureResponse response = obsClient.createTemporarySignature(request);

System.out.println("list parts using temporary signature url:");
System.out.println("\t" + response.getSignedUrl());

Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
      builder.header(entry.getKey(), entry.getValue());
}

// Make a GET request to list uploaded parts.
Request httpRequest = builder.url(response.getSignedUrl()).get().build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
            .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
System.out.println("\tStatus:" + res.code());
if (res.body() != null) {
      System.out.println("\tContent:" + res.body().string() + "\n");
}
res.close();
```

Merging Uploaded Parts

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
```

```
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;

TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.POST,
expireSeconds);
request.setBucketName("bucketname");
request.setObjectKey("objectname");

Map<String, String> headers = new HashMap<String, String>();
String contentType = "application/xml";
headers.put("Content-Type", contentType);
request.setHeaders(headers);

Map<String, Object> queryParams = new HashMap<String, Object>();
queryParams.put("uploadId", "your uploadId");
request.setQueryParams(queryParams);

TemporarySignatureResponse response = obsClient.createTemporarySignature(request);

System.out.println("complete multipart upload using temporary signature url:");
System.out.println("\t" + response.getSignedUrl());

Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
      builder.header(entry.getKey(), entry.getValue());
}

// The following content is an example code. You need to assemble the following content by listing the
response results of the uploaded parts.
String content = "<CompleteMultipartUpload>";
content += "<Part>";
content += "<PartNumber>1</PartNumber>";
content += "<ETag>da6a0d097e307ac52ed9b4ad551801fc</ETag>";
content += "</Part>";
content += "<Part>";
content += "<PartNumber>2</PartNumber>";
content += "<ETag>da6a0d097e307ac52ed9b4ad551801fc</ETag>";
content += "</Part>";
content += "</CompleteMultipartUpload>";

// POST a request to merge uploaded parts.
Request httpRequest =
builder.url(response.getSignedUrl()).post(RequestBody.create(MediaType.parse(contentType),
content.getBytes("UTF-8"))).build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
             .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
System.out.println("\tStatus:" + res.code());
if (res.body() != null) {
      System.out.println("\tContent:" + res.body().string() + "\n");
}
res.close();
```

NOTE

- **HttpMethodEnum** is an enumeration function defined in OBS Java SDK, whose value indicates the request method types.

11 Versioning Management

11.1 Versioning Overview

OBS can store multiple versions of an object. You can quickly search for and restore different versions as well as restore data in the event of misoperations or application faults.

For details, see **Versioning**.

11.2 Setting Versioning Status for a Bucket

You can call **ObsClient.setBucketVersioning** to set the versioning status for a bucket. OBS supports two versioning statuses.

Versioning Status	Description	Value in OBS Java SDK
Enabled	1. OBS creates a unique version ID for each uploaded object. Namesake objects are not overwritten and are distinguished by their own version IDs. 2. Objects can be downloaded by specifying the version ID. By default, the object of the latest version is downloaded if no version ID is specified. 3. Objects can be deleted by specifying the version ID. If an object is deleted with no version ID specified, the object will generate a delete marker with a unique version ID but is not physically deleted. 4. Objects of the latest version in a bucket are returned by default after **ObsClient.listObjects** is called. You can call **ObsClient.listVersions** to list a bucket's objects with all version IDs. 5. Except for delete markers, storage space occupied by objects with all version IDs is billed.	VersioningStatusE-num.ENABLED
Suspended	1. Noncurrent object versions are not affected. 2. OBS creates version ID **null** to an uploaded object and the object will be overwritten after a namesake one is uploaded. 3. Objects can be downloaded by specifying the version ID. By default, the object of the latest version is downloaded if no version ID is specified. 4. Objects can be deleted by specifying version IDs. If an object is deleted with no version ID specified, the object is only attached with a delete marker whose version ID is **null**. Objects with version ID **null** are physically deleted. 5. Except for delete markers, storage space occupied by objects with all version IDs is billed.	VersioningStatusE-num.SUSPENDED

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Enable versioning for a bucket.
obsClient.setBucketVersioning("bucketname", new
BucketVersioningConfiguration(VersioningStatusEnum.ENABLED));

// Suspend versioning for a bucket.
obsClient.setBucketVersioning("bucketname", new
BucketVersioningConfiguration(VersioningStatusEnum.SUSPENDED));
```

11.3 Viewing Versioning Status of a Bucket

You can call **ObsClient.getBucketVersioning** to view the versioning status of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

11.4 Obtaining a Versioning Object

You can call **ObsClient.getObject** to obtain a versioning object by specifying the version ID (**versionId**). Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Set versionId to obtain a versioning object.
ObsObject obsObject = obsClient.getObject("bucketname", "objectname", "versionid");
obsObject.getObjectContent().close();
```

NOTE

If version ID is **null**, the object of the latest version will be downloaded, by default.

11.5 Copying a Versioning Object

You can call **ObsClient.copyObject** to pass the version ID (**versionId**) to copy a versioning object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
```

```
CopyObjectRequest request = new CopyObjectRequest();
request.setSourceBucketName("sourebucketname");
request.setSourceObjectKey("sourceobjectname");
// Set the version ID of the object to be copied.
request.setVersionId("versionid");
request.setDestinationBucketName("destbucketname");
request.setDestinationObjectKey("destobjectname");
obsClient.copyObject(request);
```

11.6 Restoring a Versioning Cold Object

You can call **ObsClient.restoreObject** to restore a versioning Cold object by specifying the version ID (**versionId**). Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

RestoreObjectRequest request = new RestoreObjectRequest("bucketname", "objectname", 1);
// Restore a versioning object in the Expedited mode.
request.setRestoreTier(RestoreTierEnum.EXPEDITED);
request.setVersionId("versionid");
obsClient.restoreObject(request);
```

11.7 Listing Versioning Objects

You can call **ObsClient.listVersions** to list versioning objects.

The following table describes the parameters involved in this API.

Parameter	Description
bucketName	Bucket name
prefix	Name prefix that the objects to be listed must contain
keyMarker	Object name to start with when listing versioning objects in a bucket. All versioning objects whose names follow this parameter are listed in the lexicographical order.
maxKeys	Maximum number of versioning objects returned. The value ranges from 1 to 1000. If the value is not in this range, 1,000 versioning objects are returned by default.
delimiter	Character used to group object names. If the object name contains the **delimiter** parameter, the character string from the first character to the first delimiter in the object name is grouped under a single result element, **commonPrefix**. (If a prefix is specified in the request, the prefix must be removed from the object name.)

Parameter	Description
versionIdMarker	Indicates the object name to start with when listing objects in a bucket. All objects are listed in the lexicographical order by object name and version ID. This parameter must be used together with **keyMarker**.

NOTE

- If the value of **versionIdMarker** is not a version ID specified by **keyMarker**, **versionIdMarker** is ineffective.
- The returned result of **ObsClient.listVersions** includes the versioning objects and delete markers.

Listing Versioning Objects in Simple Mode

The following sample code shows how to list versioning objects in simple mode. A maximum of 1000 versioning objects can be returned.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
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());
}
```

NOTE

- A maximum of 1,000 versioning objects can be listed each time. If a bucket contains more than 1,000 objects and **ListVersionsResult.isTruncated** is **true** in the returned result, not all versioning objects are listed. In such cases, you can use **ListVersionsResult.getNextKeyMarker** and **ListVersionsResult.getNextVersionIdMarker** to obtain the start position for next listing.
- If you want to obtain all versioning objects in a specified bucket, you can use the paging mode for listing objects.

Listing Versioning Objects by Specifying the Number

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
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());
}
```

Listing Versioning Objects by Specifying a Prefix

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// List 100 objects whose name prefix is prefix.
ListVersionsRequest request = new ListVersionsRequest ("bucketname", 100);
request.setPrefix("prefix");
ListVersionsResult result = obsClient.listVersions(request);
for(VersionOrDeleteMarker v : result.getVersions()){
    System.out.println("\t" + v.getKey());
    System.out.println("\t" + v.getOwner());
    System.out.println("\t" + v.isDeleteMarker());
}
```

Listing Versioning Objects by Specifying the Start Position

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// List 100 versioning objects whose names following test in lexicographic order.
ListVersionsRequest request = new ListVersionsRequest ("bucketname", 100);
request.setKeyMarker("test");
ListVersionsResult result = obsClient.listVersions(request);

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

Listing All Versioning Objects in Paging Mode

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsResult result;
ListVersionsRequest request = new ListVersionsRequest ("bucketname", 100);
do{
    result = obsClient.listVersions(request);
    for(VersionOrDeleteMarker v : result.getVersions()){
        System.out.println("\t" + v.getKey());
        System.out.println("\t" + v.getOwner());
        System.out.println("\t" + v.isDeleteMarker());
    }
    request.setKeyMarker(result.getNextKeyMarker());
    request.setVersionIdMarker(result.getNextVersionIdMarker());
}while(result.isTruncated());
```

Listing All Versioning Objects in a Folder

There is no folder concept in OBS. All elements in buckets are objects. Folders are actually objects whose sizes are 0 and whose names end with a slash (/). When you set a folder name as the prefix, objects in this folder will be listed. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsResult result;
ListVersionsRequest request = new ListVersionsRequest ("bucketname", 100);
// Set the prefix of objects in the folder to dir/.
request.setPrefix("dir/");
do{
    result = obsClient.listVersions(request);
    for(VersionOrDeleteMarker v : result.getVersions()){
        System.out.println("\t" + v.getKey());
        System.out.println("\t" + v.getOwner());
        System.out.println("\t" + v.isDeleteMarker());
    }
    request.setKeyMarker(result.getNextKeyMarker());
    request.setVersionIdMarker(result.getNextVersionIdMarker());
}while(result.isTruncated());
```

Listing All Versioning Objects According to Folders in a Bucket

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
ListVersionsRequest request = new ListVersionsRequest ("bucketname", 1000);
request.setDelimiter("/");
ListVersionsResult result = obsClient.listVersions(request);
System.out.println("Objects in the root directory:");
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);
```

The following is the sample code of the **listVersionsByPrefix** function, which is used to recursively list objects in sub-folders.

```
static void listVersionsByPrefix(ObsClient obsClient, ListVersionsResult result) throws ObsException{
    for(String prefix : result.getCommonPrefixes()){
         System.out.println("Objects in folder [" + prefix + "]:");
         ListVersionsRequest request = new ListVersionsRequest ("bucketname", 1000);
         request.setDelimiter("/");
         request.setPrefix(prefix)
         result = obsClient.listVersions(request);
         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);
     }
}
```

NOTE

- The previous sample code does not include scenarios where the number of objects in a folder exceeds 1000.
- Because objects and sub-folders in a folder are to be listed and all the objects end with a slash (/), **delimiter** is always a slash (/).
- In the returned result of each recursion, **ListVersionsResult.getVersions** includes the versioning objects in the folder and **ListVersionsResult.getCommonPrefixes** includes the sub-folders in the folder.

11.8 Setting or Obtaining a Versioning Object ACL

Directly Setting a Versioning Object ACL

You can call **ObsClient.setObjectAcl** and set the version ID (**versionId**) to specify the ACL for a versioning object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Set the versioning object ACL to public-read by specifying the pre-defined access control policy.
obsClient.setObjectAcl("bucketname", "objectname", AccessControlList.REST_CANNED_PUBLIC_READ,
"versionid");

AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
owner.setId("ownerid");
acl.setOwner(owner);
// Grant the READ permission to all users.
acl.grantPermission(GroupGrantee.ALL_USERS, Permission.PERMISSION_READ);
// Set the ACL for a versioning object.
obsClient.setObjectAcl("bucketname", "objectname", acl, "versionid");
```

NOTE

The owner or grantee ID needed in the ACL indicates the account ID, which can be viewed on the **My Credentials** page of OBS Console.

Obtaining a Versioning Object ACL

You can call **ObsClient.getObjectAcl** to obtain the ACL of a versioning object by specifying the version ID (**versionId**). Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

11.9 Deleting Versioning Objects

Deleting a Single Versioning Object

You can call **ObsClient.deleteObject** to pass the version ID (**versionId**) to delete a versioning object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
obsClient.deleteObject("bucketname", "objectname", "versionid");
```

Deleting Versioning Objects in a Batch

You can call **ObsClient.deleteObjects** to pass the version ID (**versionId**) of each to-be-deleted versioning object to delete them. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
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("objectname1", "versionid1"));
toDelete.add(new KeyAndVersion("objectname2", "versionid2"));
toDelete.add(new KeyAndVersion("objectname3", "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());
```

12 Lifecycle Management

12.1 Lifecycle Management Overview

OBS allows you to set lifecycle rules for buckets to automatically transit the storage class of an object and delete expired objects, so as to effectively use storage features and optimize the storage space. You can set multiple lifecycle rules based on the prefix. A lifecycle rule must contain:

- Rule ID, which uniquely identifies the rule
- Prefix of objects that are under the control of this rule
- Transition policy of an object of the latest version, which can be specified in either mode:
    a. How many days after the object is created
    b. Transition date
- Expiration time of an object of the latest version, which can be specified in either mode:
    a. How many days after the object is created
    b. Expiration date
- Transition policy of a noncurrent object version, which can be specified in the following mode:
    – How many days after the object becomes a noncurrent object version
- Expiration time of a noncurrent object version, which can be specified in the following mode:
    – How many days after the object becomes a noncurrent object version
- Identifier specifying whether the setting is effective

For more information, see **Lifecycle Management**.

NOTE

- An object will be automatically deleted by the OBS server once it expires.
- The time set in the transition policy of an object must be earlier than its expiration time, and the time set in the transition policy of a noncurrent object version must be earlier than its expiration time.
- The configured expiration time and transition policy for a noncurrent object version will take effect only when the versioning is enabled or suspended for a bucket.

12.2 Setting Lifecycle Rules

You can call **ObsClient.setBucketLifecycle** to set lifecycle rules for a bucket.

Setting an Object Transition Policy

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
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");
Transition transition = config.new Transition();
// Specify that objects whose names contain the prefix will be transited 30 days after creation.
transition.setDays(30);
// Specify the storage class of the object after transition.
transition.setObjectStorageClass(StorageClassEnum.WARM);
// Specify a date when the objects whose names contain the prefix will be transited.
// transition.setDate(new SimpleDateFormat("yyyy-MM-dd").parse("2018-10-31"));
rule.getTransitions().add(transition);

NoncurrentVersionTransition noncurrentVersionTransition = config.new NoncurrentVersionTransition();
// Specify that objects whose names contain the prefix will be transited after changing into noncurrent
versions for 30 days.
noncurrentVersionTransition.setDays(30);
// Specify the storage class of the noncurrent object version after transition.
noncurrentVersionTransition.setObjectStorageClass(StorageClassEnum.COLD);
rule.getNoncurrentVersionTransitions().add(noncurrentVersionTransition);

config.addRule(rule);

obsClient.setBucketLifecycle("bucketname", config);
```

Setting an Object Expiration Time

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
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();
// Specify that objects whose names contain the prefix will expire 60 days after creation.
expiration.setDays(60);
// Specify a date when the objects whose names contain the prefix will expire.
// expiration.setDate(new SimpleDateFormat("yyyy-MM-dd").parse("2018-12-31"));
rule.setExpiration(expiration);

NoncurrentVersionExpiration noncurrentVersionExpiration = config.new NoncurrentVersionExpiration();
// Specify that objects whose names contain the prefix will expire after changing into noncurrent versions
for 60 days.
noncurrentVersionExpiration.setDays(60);
rule.setNoncurrentVersionExpiration(noncurrentVersionExpiration);
config.addRule(rule);

obsClient.setBucketLifecycle("bucketname", config);
```

12.3 Viewing Lifecycle Rules

You can call **ObsClient.getBucketLifecycle** to view lifecycle rules of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

for (Rule rule : config.getRules())
{
    System.out.println(rule.getId());
    System.out.println(rule.getPrefix());
    for(Transition transition : rule.getTransitions()){
        System.out.println(transition.getDays());
        System.out.println(transition.getStorageClass());
    }
    System.out.println(rule.getExpiration() != null ? rule.getExpiration().getDays() : "");
    for(NoncurrentVersionTransition noncurrentVersionTransition : rule.getNoncurrentVersionTransitions()){
        System.out.println(noncurrentVersionTransition.getDays());
        System.out.println(noncurrentVersionTransition.getStorageClass());
    }
    System.out.println(rule.getNoncurrentVersionExpiration() != null ?
rule.getNoncurrentVersionExpiration().getDays() : "");
}
```

12.4 Deleting Lifecycle Rules

You can call **ObsClient.deleteBucketLifecycle** to delete lifecycle rules of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

13 CORS

13.1 CORS Overview

Cross-origin resource sharing (CORS) allows web application programs to access resources in other domains. OBS provides developers with APIs for facilitating cross-origin resource access.

For more information, see **CORS**.

13.2 Setting CORS Rules

You can call **ObsClient.setBucketCors** to set CORS rules for a bucket. If the bucket is configured with CORS rules, the newly set ones will overwrite the existing ones. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketCors cors = new BucketCors();

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

ArrayList<String> allowedOrigin = new ArrayList<String>();
// Specify the origin of the cross-origin request.
allowedOrigin.add( "http://www.a.com");
allowedOrigin.add( "http://www.b.com");
rule.setAllowedOrigin(allowedOrigin);

ArrayList<String> allowedMethod = new ArrayList<String>();
// Specify the request method, which can be GET, PUT, DELETE, POST, or HEAD.
allowedMethod.add("GET");
allowedMethod.add("HEAD");
allowedMethod.add("PUT");
rule.setAllowedMethod(allowedMethod);

ArrayList<String> allowedHeader = new ArrayList<String>();
// Specify whether headers specified in Access-Control-Request-Headers in the OPTIONS request can be
used.
```

```
allowedHeader.add("x-obs-header");
rule.setAllowedHeader(allowedHeader);

ArrayList<String> exposeHeader = new ArrayList<String>();
// Specify response headers that users can access using application programs.
exposeHeader.add("x-obs-expose-header");
rule.setExposeHeader(exposeHeader);

// Specify the browser's cache time of the returned results of OPTIONS requests for specific resources, in
seconds.
rule.setMaxAgeSecond(10);
rules.add(rule);
cors.setRules(rules);

obsClient.setBucketCors("bucketname", cors);
```

NOTE

AllowedOrigins and **AllowedHeaders** respectively can contain up to one wildcard character (*). The wildcard character (*) indicates that all origins or headers are allowed.

13.3 Viewing CORS Rules

You can call **ObsClient.getBucketCors** to view CORS rules of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);


BucketCors 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());
}
```

13.4 Deleting CORS Rules

You can call **ObsClient.deleteBucketCors** to delete CORS rules of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

14 Access Logging

14.1 Logging Overview

OBS allows you to configure access logging for buckets. After the configuration, access to buckets will be recorded in the format of logs. These logs will be saved in specific buckets in OBS.

For more information, see **Logging**.

14.2 Enabling Bucket Logging

You can call **ObsClient.setBucketLogging** to enable bucket logging.

NOTICE

The source bucket and target bucket of logging must be in the same region.

NOTE

If the bucket is in the OBS Warm or Cold storage class, it cannot be used as the target bucket.

Enabling Bucket Logging

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketLoggingConfiguration config = new BucketLoggingConfiguration();
config.setAgency("your agency");
config.setTargetBucketName("targetbucketname");
config.setLogfilePrefix("targetprefix");

obsClient.setBucketLogging("bucketname", config);
```

Setting ACLs for Objects to Be Logged

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

String targetBucket = "targetbucketname";

// Configure logging.
BucketLoggingConfiguration config = new BucketLoggingConfiguration();
config.setAgency("your agency");
config.setTargetBucketName(targetBucket);
config.setLogfilePrefix("prefix");

// Grant the READ permission on the objects to be logged to all users.
GrantAndPermission grant1 = new GrantAndPermission(GroupGrantee.ALL_USERS,
Permission.PERMISSION_READ);
config.setTargetGrants(new GrantAndPermission[]{grant1});

obsClient.setBucketLogging("bucketname", config);
```

14.3 Viewing Bucket Logging

You can call **ObsClient.getBucketLogging** to view the logging settings of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

14.4 Disabling Bucket Logging

You can call **ObsClient.setBucketLogging** to clear logging settings of a bucket so as to disable logging of the bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Leave the logging settings in blank.
obsClient.setBucketLogging("bucketname", new BucketLoggingConfiguration());
```

15 Static Website Hosting

15.1 Static Website Hosting Overview

You can upload the content files of the static website to your bucket in OBS as objects and configure the **public-read** permission on the files, and then configure the static website hosting mode for your bucket to host your static websites in OBS. After this, when third-party users access your websites, they actually access the objects in your bucket in OBS. When using static website hosting, you can configure request redirection to redirect specific or all requests.

For more information, see **Static Website Hosting**.

15.2 Website File Hosting

You can perform the following to implement website file hosting:

Step 1 Upload a website file to your bucket in OBS as an object and set the MIME type for the object.

Step 2 Set the ACL of the object to **public-read**.

Step 3 Access the object using a browser.

----End

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Upload objects and set the MIME type for the objects.
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);

// Set the object ACL to public-read.
obsClient.setObjectAcl("bucketname", "test.html", AccessControlList.REST_CANNED_PUBLIC_READ);
```

NOTE

You can use **https://***bucketname.your-endpoint***/test.html** in a browser to access files hosted using the sample code.

15.3 Setting Website Hosting

You can call **ObsClient.setBucketWebsite** to set website hosting for a bucket.

Configuring the Default Homepage and Error Pages

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

WebsiteConfiguration config = new WebsiteConfiguration();
// Configure the default homepage.
config.setSuffix("index.html");
// Configure the error pages.
config.setKey("error.html");
obsClient.setBucketWebsite("bucketname", config);
```

Configuring the Redirection Rules

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

WebsiteConfiguration config = new WebsiteConfiguration();
// Configure the default homepage.
config.setSuffix("index.html");
// Configure the error pages.
config.setKey("error.html");

RouteRule rule = new RouteRule();
Redirect r = new Redirect();
r.setHostName("www.example.com");
r.setHttpRedirectCode("305");
r.setRedirectProtocol(ProtocolEnum.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.setBucketWebsite("bucketname", config);
```

Configuring Redirection for All Requests

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

WebsiteConfiguration config = new WebsiteConfiguration();
RedirectAllRequest redirectAll = new RedirectAllRequest();
redirectAll.setHostName("www.example.com");
redirectAll.setRedirectProtocol(ProtocalEnum.HTTP);
config.setRedirectAllRequestsTo(redirectAll);

obsClient.setBucketWebsite("bucketname", config);
```

15.4 Viewing Website Hosting Settings

You can call **ObsClient.getBucketWebsite** to view the hosting settings of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

15.5 Deleting Website Hosting Settings

You can call **ObsClient.deleteBucketWebsite** to delete the hosting settings of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

16 Event Notification

16.1 Event Notification Overview

The event notification function allows users to be notified of their operations on buckets, ensuring users know events happened on buckets in a timely manner. Currently, OBS supports event notifications through Simple Message Notification (SMN).

For more information, see **Event Notification**.

16.2 Setting Event Notification

You can call **ObsClient.setBucketNotification** to set event notification for a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
BucketNotificationConfiguration bucketNotificationConfig = new BucketNotificationConfiguration();

TopicConfiguration topicConfig = new TopicConfiguration();
topicConfig.setId("id1");
topicConfig.setTopic("your topic");
topicConfig.getEventTypes().add(EventTypeEnum.OBJECT_CREATED_ALL);
Filter topicFilter = new Filter();
topicFilter.getFilterRules().add(new FilterRule("prefix", "smn"));
topicFilter.getFilterRules().add(new FilterRule("suffix", ".jpg"));
topicConfig.setFilter(topicFilter);
bucketNotificationConfig.addTopicConfiguration(topicConfig);


obsClient.setBucketNotification("bucketname", bucketNotificationConfig);
```

16.3 Viewing Event Notification Settings

You can call **ObsClient.getBucketNotification** to view event notification settings of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

System.out.println(config);
```

16.4 Disabling Event Notification

To disable event notification on buckets is to call **ObsClient.setBucketNotification** to clear all event notification settings. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

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

17 Troubleshooting

17.1 HTTP Status Codes

The OBS server complies with the HTTP standard. After an API is called, the OBS server returns a standard HTTP status code. The following tables list the categories of HTTP status codes and the common HTTP status codes in OBS.

- Categories of HTTP status codes

Category	Description
1XX	Informational response. A request is received by the server and the server requires the requester to continue the operation. This category is usually invisible to the client.
2XX	Success. The operation is received and processed successfully.
3XX	Redirection. Further operations to complete the request are required. This category is usually invisible to the client.
4XX	Client errors. The request contains a syntax error, or the request cannot be implemented.
5XX	Server errors. An error occurs when the server is processing the request.

- Common HTTP status codes in OBS and their meanings

HTTP Status Code	Description	Possible Cause
400 Bad Request	The request parameter is incorrect.	● Invalid request parameter. ● The consistency check fails after the client request carries MD5. ● An invalid parameter is transferred when the SDK is used. ● An invalid bucket name is used.
403 Forbidden	The access is denied.	● The signature carried in the request header does not match with the signature calculated by the OBS server. Generally, the error is caused by incorrect AK/SK. ● The account does not have the permission to access the requested resource. ● The account is in arrears. ● The bucket space is insufficient when a quota is set for the bucket. ● Invalid AK ● The time difference between the client and the server is too large. That is, the time of the server where the client is located is not synchronized with the time of the NTP service.
404 Not Found	The requested resource does not exist.	● The bucket does not exist. ● The object does not exist. ● The bucket policy configuration does not exist. For example, the bucket CORS configuration or bucket policy configuration does not exist. ● The multipart upload does not exist.
405 Method Not Allowed	The request method is not supported.	The requested method or feature is not supported in the region where the bucket resides.

HTTP Status Code	Description	Possible Cause
408 Request Timeout	Request timed out.	The Socket connection between the server and client timed out.
409 Conflict	Request conflicts occur.	● Buckets of the same name are created in different regions. ● Deletion of a non-empty bucket is attempted.
500 Internal Server Error	An internal error occurs on the server side.	An internal error occurs on the server side.
503 Service Unavailable	The service is unavailable.	The server cannot be accessed temporarily.

17.2 OBS Server-Side Error Codes

If the OBS server encounters an error when processing a request, a response containing the error code and error description is returned. The following table lists details about each error code and HTTP status code.

HTTP Status Code	Error Code	Error Message	Solution
301 Moved Permanently	PermanentRedirect	The requested bucket can be accessed only through the specified address. Send subsequent requests to the address.	Send the request to the returned redirection address.
301 Moved Permanently	WebsiteRedirect	The website request lacks **bucketName**.	Put the bucket name in the request and try again.
307 Moved Temporarily	TemporaryRedirect	Temporary redirection. If the DNS is updated, the request is redirected to the bucket.	The system automatically redirects the request or sends the request to the redirection address.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	BadDigest	The specified value of **Content-MD5** does not match the value received by OBS.	Check whether the MD5 value carried in the header is the same as that calculated by the message body.
400 Bad Request	BadDomainName	Invalid domain name.	Use a valid domain name.
400 Bad Request	BadRequest	Invalid request parameter.	Modify the parameter according to the error details returned in the message body.
400 Bad Request	CustomDomainAreadyExist	The configured domain already exists.	It has been configured and does not need to be configured again.
400 Bad Request	CustomDomainNotExist	The domain to be deleted does not exist.	The domain is not configured or has been deleted. You do not need to delete it.
400 Bad Request	EntityTooLarge	The size of the object uploaded using the POST method exceeds the upper limit.	Modify the conditions specified in the policy when posting the object or reduce the object size.
400 Bad Request	EntityTooSmall	The size of the object uploaded using the POST method does not reach the lower limit.	Modify the conditions specified in the policy when posting the object or increase the object size.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	IllegalLocation-ConstraintException	A request without **Location** is sent for creating a bucket in a non-default region.	Send the bucket creation request to the default region, or send the request with the **Location** of the non-default region.
400 Bad Request	IncompleteBody	No complete request body is received due to network or other problems.	Upload the object again.
400 Bad Request	IncorrectNumber-OfFilesInPost Request	Each POST request must contain one file to be uploaded.	Carry a file to be uploaded.
400 Bad Request	InvalidArgument	Invalid parameter.	Modify the parameter according to the error details in the message body.
400 Bad Request	InvalidBucket	The bucket to be accessed does not exist.	Try another bucket name.
400 Bad Request	InvalidBucketName	The bucket name specified in the request is invalid, which may have exceeded the maximum length, or contain special characters that are not allowed.	Try another bucket name.
400 Bad Request	InvalidLocation-Constraint	The specified **Location** in the bucket creation request is invalid or does not exist.	Correct the **Location** in the bucket creation request.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	InvalidPart	One or more specified parts are not found. The parts may not be uploaded or the specified entity tags (ETags) do not match the parts' ETags.	Merge the parts correctly according to the ETags.
400 Bad Request	InvalidPartOrder	Parts are not listed in ascending order by part number.	Sort the parts in ascending order and merge them again.
400 Bad Request	InvalidPolicyDocu-ment	The content of the form does not meet the conditions specified in the policy document.	Modify the policy in the constructed form according to the error details in the message body and try again.
400 Bad Request	InvalidRedirectLo-cation	Invalid redirect location.	Specify the correct IP address.
400 Bad Request	InvalidRequest	Invalid request.	Modify the parameter according to the error details returned in the message body.
400 Bad Request	InvalidRequestBod y	The request body is invalid. The request requires a message body but no message body is uploaded.	Upload the message body in the correct format.
400 Bad Request	InvalidTargetBuck-etForLogging	The delivery group has no ACL permission for the target bucket.	Configure the target bucket ACL and try again.
400 Bad Request	KeyTooLongError	The provided key is too long.	Use a shorter key.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	MalformedACLError	The provided XML file is in an incorrect format or does not meet format requirements.	Use the correct XML format to retry.
400 Bad Request	MalformedError	The XML format in the request is incorrect.	Use the correct XML format to retry.
400 Bad Request	MalformedLoggingStatus	The XML format of **Logging** is incorrect.	Use the correct XML format to retry.
400 Bad Request	MalformedPolicy	The bucket policy failed the check.	Modify the bucket policy according to the error details returned in the message body.
400 Bad Request	MalformedQuotaError	The Quota XML format is incorrect.	Use the correct XML format to retry.
400 Bad Request	MalformedXML	An XML file of a configuration item is in incorrect format.	Use the correct XML format to retry.
400 Bad Request	MaxMessageLengthExceeded	Copying an object does not require a message body in the request.	Remove the message body and retry.
400 Bad Request	MetadataTooLarge	The size of the metadata header has exceeded the upper limit.	Reduce the size of the metadata header.
400 Bad Request	MissingRegion	No region contained in the request and no default region defined in the system.	Carry the region information in the request.
400 Bad Request	MissingRequestBodyError	An empty XML file is sent as a request.	Provide the correct XML file.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	MissingRequired-Header	A required header is missing in the request.	Provide the required header.
400 Bad Request	MissingSecurity-Header	A required header is missing in the request.	Provide the required header.
400 Bad Request	TooManyBuckets	You have attempted to create more buckets than allowed.	Delete some buckets and try again.
400 Bad Request	TooManyCustomDomains	Too many user accounts are configured.	Delete some user accounts and try again.
400 Bad Request	TooManyWrongSignature	The request is rejected due to high-frequency errors.	Replace AK and try again.
400 Bad Request	UnexpectedContent	The request requires a message body which is not carried by the client, or the request does not require a message body but the client carries the message body.	Try again according to the instruction.
400 Bad Request	UserKeyMustBeSpecified	This operation is only available to special users.	Contact the technical support.
403 Forbidden	AccessDenied	Access denied, because the request does not carry a date header or the header format is incorrect.	Provide a correct date header in the request.

HTTP Status Code	Error Code	Error Message	Solution
403 Forbidden	AccessForbidden	Insufficient permission. No CORS rule is configured for the bucket or the CORS rule does not match.	Modify the CORS configuration of the bucket or send the matched OPTIONS request based on the CORS configuration of the bucket.
403 Forbidden	AllAccessDisabled	You have no permission to perform the operation. The bucket name is forbidden.	Change the bucket name.
403 Forbidden	DeregisterUserId	The user has been deregistered.	Top up or re-register.
403 Forbidden	InArrearOrInsufficientBalance	The subscriber owes fees or the account balance is insufficient, and the subscriber does not have the permission to perform an operation.	Top up the account.
403 Forbidden	InsufficientStorageSpace	Insufficient storage space.	If the quota is exceeded, increase quota or delete some objects.
403 Forbidden	InvalidAccessKeyId	The access key ID provided by the customer does not exist in the system.	Provide correct access key ID.
403 Forbidden	NotSignedUp	Your account has not been registered with the system. Only a registered account can be used.	Register OBS.

HTTP Status Code	Error Code	Error Message	Solution
403 Forbidden	RequestTimeTooSkewed	The request time and the server's time differ a lot.	Check whether the difference between the client time and the current time is too large.
403 Forbidden	SignatureDoesNotMatch	The provided signature in the request does not match the signature calculated by OBS.	Check your secret access key and signature calculation method.
403 Forbidden	Unauthorized	You have not been authenticated in real name.	Authenticate your real name and try again.
404 Not Found	NoSuchBucket	The specified bucket does not exist.	Create a bucket and perform the operation again.
404 Not Found	NoSuchBucketPolicy	No bucket policy exists.	Configure a bucket policy.
404 Not Found	NoSuchCORSConfiguration	No CORS configuration exists.	Configure CORS first.
404 Not Found	NoSuchCustomDomain	The requested user domain does not exist.	Set a user domain first.
404 Not Found	NoSuchKey	The specified key does not exist.	Upload the object first.
404 Not Found	NoSuchLifecycle-Configuration	The requested lifecycle rule does not exist.	Configure a lifecycle rule first.
404 Not Found	NoSuchUpload	The specified multipart upload does not exist. The upload ID does not exist or the multipart upload job has been aborted or completed.	Use the existing part or reinitialize the part.

HTTP Status Code	Error Code	Error Message	Solution
404 Not Found	NoSuchVersion	The specified version ID does not match any existing version.	Use a correct version ID.
404 Not Found	NoSuchWebsiteConfiguration	The requested website does not exist.	Configure the website first.
405 Method Not Allowed	MethodNotAllowed	The specified method is not allowed against the requested resource.	The method is not allowed.
408 Request Timeout	RequestTimeout	No read or write operation has been performed within the timeout period of the socket connection between the user and the server.	Check the network and try again, or contact technical support.
409 Conflict	BucketAlreadyExists	The requested bucket name already exists. The bucket namespace is shared by all users of OBS. Select another name and retry.	Try another bucket name.
409 Conflict	BucketAlreadyOwnedByYou	Your previous request for creating the named bucket succeeded and you already own it.	You do not need to create the bucket again.
409 Conflict	BucketNotEmpty	The bucket that you tried to delete is not empty.	Delete the objects in the bucket and then delete the bucket.

HTTP Status Code	Error Code	Error Message	Solution
409 Conflict	OperationAborted	A conflicting operation is being performed on this resource. Retry later.	Try again later.
409 Conflict	ServiceNotSupported	The request method is not supported by the server.	Not supported by the server. Contact technical support.
411 Length Required	MissingContentLength	The HTTP header Content-Length is not provided.	Provide the Content-Length header.
412 Precondition Failed	PreconditionFailed	At least one of the specified preconditions is not met.	Modify according to the condition prompt in the returned message body.
416 Client Requested Range Not Satisfiable	InvalidRange	The requested range cannot be obtained.	Retry with the correct range.
500 Internal Server Error	InternalError	An internal error occurs. Retry later.	Contact the technical support.
501 Not Implemented	ServiceNotImplemented	The request method is not implemented by the server.	Not supported currently. Contact the technical support.
503 Service Unavailable	ServiceUnavailable	The server is overloaded or has internal errors.	Try again later or contact the technical support.
503 Service Unavailable	SlowDown	Too frequent requests. Reduce your request frequency.	Reduce your request frequency.

17.3 SDK Custom Exceptions

SDK custom exceptions (**ObsException**), thrown by **ObsClient**, are inherited from class **java.lang.RuntimeException**. Exceptions are usually OBS server errors, including **OBS error codes** and error information. This facilitates users to locate problems and troubleshoot faults.

ObsException contains the following error information:

- **ObsException.getResponseCode**: HTTP status code
- **ObsException.getErrorCode**: Error code returned by the OBS server
- **ObsException.getErrorMessage**: Error description returned by the OBS server
- **ObsException.getErrorRequestId**: Request ID returned by the OBS server
- **ObsException.getErrorHostId**: Requested server ID
- **ObsException.getResponseHeaders:** HTTP response headers

17.4 SDK Common Response Headers

After you call an API in an instance of **ObsClient**, an instance of the **HeaderResponse** class (or its sub-class) will be returned. It contains information about HTTP/HTTPS response headers.

Sample code for processing public response headers:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
HeaderResponse response = obsClient.createBucket("bucketname");

// Obtain the UUID from the public response headers.
System.out.println("\t" + response.getRequestId());

obsClient.close();
```

17.5 Log Analysis

How To Enable Logging

1. Save the **log4j2.xml** file obtained from the OBS Java SDK package to the **classpath** root directory.
2. Call **Log4j2Configurator.setLogConfig** to specify the save path of **log4j2.xml** directly.

NOTE

You can obtain the default log configuration file **log4j2.xml** from the OBS Java SDK package, and then modify to customize the file.

Log Path

The log path of OBS Java SDK is specified in **log4j2.xml**. Logs are saved in the path represented by system variable **user.dir** of JDK by default. In general, there are three logs files as follows:

File Name	Description
OBS-SDK.interface_north.log	Northbound log file, which saves the logs about the communication between OBS Java SDK and third-party applications of users.

File Name	Description
OBS-SDK.interface_south.log	Southbound log file, which saves the logs about the communication between OBS Java SDK and the OBS server.
OBS-SDK.access.log	Run log file of the OBS server.

Log Format

The SDK log format is: *Log time*|*Thread number*|*Log level*|*Log content*. The following are example logs:

```
#Southbound logs
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].

#Northbound logs
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
```

Log Level

When current logs cannot be used to troubleshoot system faults, you can change the log level to obtain more information. You can obtain the most information in **TRACE** logs and the least information in **ERROR** logs.

Log level description:

- **OFF**: Close level. If this level is set, logging will be disabled.
- **TRACE**: Trace level. If this level is set, all log information will be printed. This level is not recommended.
- **DEBUG**: Debugging level. If this level is set, information about logs of the **INFO** level and above, HTTP/HTTPS request and response headers, and **StringToSign** information calculated by authentication algorithm will be printed.
- **INFO**: Information level. If this level is set, information about logs of the **WARN** level and above, time consumed for each HTTP/HTTPS request, and time consumed for calling the ObsClient API will be printed.
- **WARN**: Warning level. If this level is set, information about logs of the **ERROR** level and above, as well as information about some critical events (for example, the number of retry attempts exceeds the upper limit) will be printed.
- **ERROR**: Error level. If this level is set, only error information will be printed.

How to Set

The following sample code shows how to set different levels for the southbound logs, northbound logs, and OBS server run logs. (For details about log configuration, see configuration file **log4j2.xml**.)

```
<!-- 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>

<!-- access log -->
<Logger name="com.obs.log.AccessLogger" level="ERROR" additivity="false">
        <AppenderRef ref="AccessLogAppender" />
</Logger>
```

18 FAQs

18.1 How Can I Create a Folder?

To create a folder in an OBS bucket is to create an object whose size is 0 and whose name ends with a slash (/). For details, see **Creating a Folder**.

18.2 How Can I List All Objects in a Bucket?

For details, see **Listing Objects** and **Listing Versioning Objects**.

18.3 How Can I Use a URL for Authorized Access?

See **10.1 Using a URL for Authorized Access**.

18.4 How Can I Upload an Object in Browser-Based Mode?

For details, see **Performing a Browser-Based Upload**.

18.5 How Can I Download a Large Object in Multipart Mode?

For details, see **Performing a Partial Download**.

18.6 What Can I Do to Implement Server-Side Root Certificate Verification?

For details, see **Configuring Server-Side Certificate Verification**.

18.7 How Can I Set an Object to Be Accessible to Anonymous Users?

To enable anonymous users to access an object, perform the following steps:

Step 1 Set the object access permission to **public-read** by referring to **9.2 Managing Object ACLs**.

Step 2 Obtain the URL of the object by referring to **18.11 How Do I Obtain the Object URL?** and provide it to anonymous users.

Step 3 An anonymous user can access the object by entering the URL on a browser.

----End

18.8 How Can I Identify the Endpoint and Region of OBS?

For details, see **Obtaining Endpoints**.

18.9 What Is the Retry Mechanism of SDK?

SDK uses the **maxErrorRetry** parameter configured in **4.3 Configuring an Instance of ObsClient** to retry. The default value for retry times is **3**. A value ranges from **0** to **5** is recommended. If the network connection is abnormal or the server returns the 5*XX* error when an ObsClient API is called, the SDK performs an exponential backoff retry.

NOTICE

- For **ObsClient.putObject**, when the data source is an InputStream other than FileInputStream, the SDK does not retry when an I/O exception occurs because the data stream cannot be read back. The upper-layer application needs to retry.
- When **ObsClient.getObject** is successfully called and **ObsObject** is returned, the SDK does not retry when an I/O exception occurs during data reading from **ObsObject.getObjectContent** because this situation is beyond the scope of the processing logic of the SDK. The upper-layer application needs to retry.

18.10 How Do I Obtain the Static Website Access Address of a Bucket?

After a bucket is configured to work in static website hosting mode, you can use the following method to combine the static website access address of the bucket.

https://*bucket name.static website hosting domain name*

NOTE

- You can click **here** to view the static website hosting domain names in each region.

18.11 How Do I Obtain the Object URL?

If the uploaded object is set to be read by anonymous users, anonymous users can download the object through the object URL directly. Methods to obtain the object URL are as follows:

Method 1: Query by calling the API. After an object is uploaded using the ObsClient API, **PutObjectResult** is returned. You can call **getObjectUrl** to obtain the URL of the uploaded object. The sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Call putObject to upload the object and obtain the return result.
PutObjectResult result = obsClient.putObject("bucketname", "objectname", new File("localfile"));
// Read the URL of the uploaded object.
System.out.println("\t" + result.getObjectUrl());
```

Method 2: Compose the URL in the format of **https://***Bucket name.Domain name/Directory level/Object name*.

NOTE

- If the object resides in the root directory of a bucket, its URL does not contain a directory level.
- You can click **here** to view the domain names of each region.

18.12 How to Improve the Speed of Uploading Large Files over the Public Network?

If the size of a file exceeds 100 MB, you are advised to upload the file using multipart upload over the public network. Multipart upload allows uploading a single object as parts separately. Each part is a part of consecutive object data. You can upload parts in any sequence. A part can be reloaded after an upload failure, without affecting other parts. Uploading multiple parts of an object using multiple threads concurrently can greatly improve the transmission efficiency.

For details about the code example, see **7.7 Performing a Multipart Upload**.

18.13 How Do I Stop an Ongoing Upload Task?

The SDK does not support this feature and secondary development is required. You can stop an ongoing upload task by stopping the data flow and capturing exceptions.

18.14 How Can I Perform a Multipart Upload?

In a multipart upload, you can specify a part of the file to be uploaded by performing the following steps:

Step 1 You need to initialize an instance of ObsClient by using AK, SK, and endpoint.

Step 2 Specify the bucket name and object name to initialize **InitiateMultipartUploadRequest**. Call **InitiateMultipartUploadRequest.setMetadata** to set the metadata of the object to be uploaded. Then, call **ObsClient.initiateMultipartUpload** to initialize a multipart upload task. A globally unique identifier (upload ID) is returned to identify this task.

Step 3 Specify the bucket name and object name to initialize **UploadPartRequest**. Call **UploadPartRequest.setUploadId** to set the upload ID to which the part to be uploaded belongs; call **setPartNumber** to set the part number of the part; call **setFile** to set the large file to which the part belongs; call **setPartSize** to set the part size; and then call **ObsClient.uploadPart** to upload the part. The ETag value of the uploaded part is returned.

Step 4 After all parts are uploaded, specify the bucket name, object name, **uploadId**, and **partEtags** to initialize a **CompleteMultipartUploadRequest** request. Then, call **ObsClient.completeMultipartUpload** to merge parts.

----End

For details, see **7.7 Performing a Multipart Upload**.

18.15 How Can I Perform a Download in Multipart Mode?

In a multipart download, you can specify the range of data to be downloaded. The procedure is as follows:

Step 1 You need to initialize an instance of ObsClient by using AK, SK, and endpoint.

Step 2 Specify the bucket name and object name to initialize **GetObjectRequest**. Call **GetObjectRequest.setRangeStart** and **GetObjectRequest.setRangeEnd** to set the start and end points of the object data to be downloaded.

Step 3 Call **ObsClient.getObject** to send the **GetObjectRequest** request in step 2 to download the data in multipart mode.

----End

For details, see **8.3 Performing a Partial Download**.

18.16 How Can I Obtain the AK and SK?

Step 1 Log in to OBS Console. In the upper right corner of the page, hover the cursor over the username and click **My Credentials**.

Step 2 On the **My Credentials** page, select **Access Keys** in the navigation pane on the left.

Step 3 On the **Access Keys** page, click **Create Access Key**.

Step 4 In the **Create Access Key** dialog box that is displayed, enter the password and verification code.

Step 5 Click **OK**.

Step 6 In the **Download Access Key** dialog box that is displayed, click **OK** to save the access keys to your browser's default download path.

Step 7 Open the downloaded **credentials.csv** file to obtain the access keys (AK and SK).

----End

For information, see **3.2 Creating Access Keys**.

18.17 How Do I Confirm That the Uploaded Object Has Overwritten the Existing Object in the Bucket with the Same Name?

After the upload is complete, you can call **ObsClient.getObjectMetadata** to obtain the size and last modification time of the target object and compare them with those in the data source. If the sizes are the same and the last modification time of the target object is later than that of the data source, the upload is successful. Otherwise, the upload fails. For details about **ObsClient.getObjectMetadata**, see **9.1 Obtaining Object Properties**.

18.18 Does the SDK Support Uploading, Downloading, or Copying Objects in a Batch?

No.

Currently, the SDK does not provide such APIs. You need to encapsulate the service codes for uploading, downloading, or copying objects in a batch by yourself. The procedure is as follows:

Step 1 List all objects to be uploaded, downloaded, or copied. For details about how to list objects to be downloaded, see **9.3 Listing Objects**.

Step 2 Call the API for uploading, downloading, or copying a single object for the listed objects.

----End

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
final String bucketName = "bucketname";
// Define the prefix of objects in a bucket.
```

```
final String objectPre = "object/";
// Folder to be uploaded
final String localDirPath = "localDirPath";
final List<File> list = new ArrayList<>();
// Scan all objects in the folder.
static void listFiles(File file){
    File[] fs = file.listFiles();
    assert fs != null;
    if (fs.length < 1){
        // If an empty folder needs to be uploaded, add it to the list.
        list.add(file);
    }else{
        for (File f:fs){
            if (f.isDirectory()){
                listFiles(f);
            }
            if (f.isFile()){
                // Add objects to be uploaded to the list.
                list.add(f);
            }
        }
    }
}
// Traverse the folder to be uploaded and obtain all objects to be uploaded.
File file = new File(localDirPath);
listFiles(file);

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Initialize the thread pool.
ExecutorService executorService = Executors.newFixedThreadPool(20);

// Concurrently upload parts.
for (File f:list){
    executorService.execute(() -> {
        if (f.isDirectory()){
            // For empty folders, create empty folder objects in the bucket.
            String remoteObjectKey = objectPre + f.getPath().substring(localDirPath.length() + 1) + "/";
            obsClient.putObject(bucketName, remoteObjectKey, new ByteArrayInputStream(new byte[0]));
        }else{
            String remoteObjectKey = objectPre + f.getPath().substring(localDirPath.length() + 1);
            obsClient.putObject(bucketName, remoteObjectKey, new File(f.getPath()));
        }
    });
}

// Wait until the upload is complete.
executorService.shutdown();
while (!executorService.isTerminated())
{
    try
    {
        executorService.awaitTermination(5, TimeUnit.SECONDS);
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
}
// Close obsClient.
try {
    obsClient.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

NOTE

You can use multiple threads to concurrently upload, download, and copy data to improve efficiency.

A API Reference

For details about all parameters and definitions of APIs in the OBS Java SDK, see the **Object Storage Service Java SDK API Reference**.

B Change History

Release Date	**What's New**
2020-02-26	This is the first official release.