# S3 java sdk 文档

# 1 前言

## 1.1 简介

- 对象存储 S3 接口 Java SDK 采用了开源的 awsjavasdks3。
- 本文档主要介绍 SDK 的安装、使用与注意事项。
- 假设您已经开通了对象存储服务， 并创建了 ak 与 sk。

# 2 安装

## 2.1 在 maven 项目中加入依赖项

向 Maven 工程中的 pom.xml 添加下述依赖项，即可使用原生的 S3 Java SDK，可以参考 aws-sdk-java。

```
<dependencies>
         <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-s3</artifactId>
            <version>1.10.69</version>
        </dependency>
           <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
           </dependency>
</dependencies>
```

## 2.2 初始化

要使用 SDK 的功能，需要先初始化，代码如下所示：

```
@Before
public void setUp() {
    config.setProtocol(Protocol.HTTP);
    config.setSignerOverride("S3SignerType");
    client = new AmazonS3Client(
           new BasicAWSCredentials(accessKeyID, secretAccessKeyID), config);
    client.setEndpoint(endpointName);
    S3ClientOptions options = new S3ClientOptions();
    options.setPathStyleAccess(true);
    client.setS3ClientOptions(options);
}
```

代码中，accessKeyID 与 secretAccessKeyID 是在对象存储中创建用户后，系统分配给用户用于访问对象　存储的凭证。而 endpointName 是云存储服务商提供的服务地址（可以是 IP 地址或者域名）。

## 2.3 关闭 client

```
client.shutdown();
```

# 3 接口说明

注意：对于接口的详细说明，可以参考 AWS JAVA SDK Javadoc。

# 4 管理 bucket

## 4.1 创建 bucket

您可以使用接口创建 Bucket。如下代码展示如何新建一个 Bucket：

```
client.createBucket(bucketName);
```

注：

- Bucket 的命名需要遵循对象存储中容器的命名规范。

- Bucket 的名字是全局唯一的，所以您需要保证 Bucket 名称不与别人重复。

## 4.2 列举 bucket

您可以使用接口列举指定用户下的所有 Bucket：

```
List<Bucket> bucketList = client.listBuckets();
```

## 4.3 删除 Bucket

您可以使用接口删除 Bucket：

```
client.deleteBucket(bucketName);
```

注：

- 如果容器不为空，则无法被删除，服务端会返回错误；
- 必选先删除容器中的所有对象后，容器才能成功删除。

## 4.4 判断 Bucket 是否存在

您可以使用 doesBucketExist 接口判断该 Bucket 是否已存在：

```
boolean exists = client.doesBucketExist(bucketName);
```

# 5 对象管理

## 5.1 上传对象

Java SDK 提供了两种接口来进行文件的上传，即简单上传与分片上传。

### 5.1.1 简单上传

简单上传采用 putObject 接口，数据可以是二进制流或者本地文件。使用该接口上传的 Object 其大小不能超过 5GB。

#### 5.1.1.1 流失上传

##### 5.1.1.1.1 上传字符串

```
String content = "Object Content";
client.putObject(bucketName, objectKey, new ByteArrayInputStream(co
ntent.getBytes()), null);
```

##### 5.1.1.1.2 上传 byte 数组

```
byte[] content = "Object Content".getBytes();
client.putObject(bucketName, objectKey, new ByteArrayInputStream(co
ntent.getBytes()), null);
InputStream inputStream = new URL("").openSt
ream();
client.putObject(bucketName, objectKey, inputStream, null);
```

##### 5.1.1.1.3 上传网络流

```
InputStream inputStream = new URL("").openSt
ream();
client.putObject(bucketName, objectKey, inputStream, null);
```

##### 5.1.1.1.4 上传文件流

```
InputStream inputStream = new FileInputStream("localFile");
client.putObject(bucketName, objectKey, inputStream, null);
```

注意：
使用流式上传时，如果采用上述的方式，会有 Out Of Memory 的风险（数据会缓存在内存中）。如果事先知道对象的大小，可以采用如下的方式，来规避风险。

```
String content = "Object Content";
ObjectMetadata meta = new ObjectMetadata();
// 设置对象大小
meta.setContentLength(content.length());
client.putObject(bucketName, objectKey, new ByteArrayInputStr
eam(content.getBytes()), meta);
```

#### 5.1.1.2 上传本地文件

```
client.putObject(bucketName, objectKey, new File("localFile"));
```

### 5.1.2 分片上传

#### 5.1.2.1 上传文件

在上传文件时，需要如下两步：

- 创建一个 TransferManager 类的实例。
- 执行 TransferManager.upload 来上传对象。您可以通过文件上传，也可以通过 stream 流上传。

```
String filePath = "Path of the file to upload";
TransferManager tm = new TransferManager(client);
// TransferManager  采用异步方式进行处理，因此该调用会立即返回。
Upload upload = tm.upload(bucketName, key, new File(filePath));
try {
    //  等待上传全部完成。
    upload.waitForCompletion();
    System.out.println("Upload complete.");
} catch (AmazonClientException amazonClientException) {
    System.out.println("Unable to upload file, upload was aborted.");
amazonClientException.printStackTrace();
}
// 在完成操作后，您必须显示关闭 TransferManager 。
tm.shutdownNow();
```

#### 5.1.2.2 上传文件流

```
TransferManager tm = new TransferManager(client);
// 设置对象大小，如果不设置，由于数据会全部缓存在内存中，可能会将内存耗尽。
ObjectMetadata meta = new ObjectMetadata();
meta.setContentLength(objectLength);
// TransferManager  采用异步方式进行处理，因此该调用会立即返回。
Upload upload = tm.upload(bucketName, key, inputStream, meta);
try {
    //  等待上传全部完成。
    upload.waitForCompletion();
```

```
    System.out.println("Upload complete.");
} catch (AmazonClientException amazonClientException) {
    System.out.println("Unable to upload file, upload was aborted.");
    amazonClientException.printStackTrace();
}
// 在完成操作后，您必须显示关闭 TransferManager 。
tm.shutdownNow();
```

### 5.1.2.3 上传文件夹

您可以使用 TransferManager.uploadDirectory 直接上传本地的一个文件夹到对象存储中。

```
String directory = "Path of the directory to upload";
String keyPrefix = "The key prefix of the virtual directory to uplo
ad to";
boolean includeSubdirectories = true;
TransferManager tm = new TransferManager(client);
// TransferManager  采用异步方式进行处理，因此该调用会立即返回。
MultipleFileUpload upload = tm.uploadDirectory(bucketName, keyPrefix,
new File(directory), includeSubdirectories);
try {
    //  等待上传全部完成。
    upload.waitForCompletion();
    System.out.println("Upload complete.");
} catch (AmazonClientException amazonClientException) {
    System.out.println("Unable to upload file, upload was aborted.");
    amazonClientException.printStackTrace();
}
// 在完成操作后，您必须显示关闭 TransferManager 。
tm.shutdownNow();
```

### 5.1.2.4 下载文件

```
String filePath = "Path to save the file";
TransferManager tm = new TransferManager(client);
// TransferManager  采用异步方式进行处理，因此该调用会立即返回。
Download download = tm.download(bucketName, key, new File(filePath));
try {
    //  等待下载全部完成。
```

```
    download.waitForCompletion();
    System.out.println("Download complete.");
} catch (AmazonClientException amazonClientException) {
    System.out.println("Unable to download file, download was aborted.");
    amazonClientException.printStackTrace();
}
// 在完成操作后，您必须显示关闭 TransferManager 。
tm.shutdownNow();
```

### 5.1.2.5 下载文件夹

您可以使用 TransferManager.downloadDirectory 来下载整个文件夹，包括子文件夹中的内容。

```
String destinationDirectory = "Destination Directory";
String keyPrefix = "The key prefix for the virtual directory";
TransferManager tm = new TransferManager(client);
// TransferManager 采用异步方式进行处理，因此该调用会立即返回。
MultipleFileDownload download = tm.downloadDirectory(bucketName, keyPrefix,
new File(destinationDirectory));
try {
    //  等待下载全部完成。
    download.waitForCompletion();
    System.out.println("Download complete.");
} catch (AmazonClientException amazonClientException) {
    System.out.println("Unable to download file, download was aborted.");
    amazonClientException.printStackTrace();
}
// 在完成操作后，您必须显示关闭 TransferManager 。
tm.shutdownNow();
```

### 5.1.2.6 取消分片上传

您可以使用 TransferManager.abortMultipartUploads 来取消分片上传。

```
int sevenDays = 1000 * 60 * 60 * 24 * 7;
Date oneWeekAgo = new Date(System.currentTimeMillis() - sevenDays);
try {
    // 取消那些在一个星期前初始化并还未完成的分片上传
    tm.abortMultipartUploads(bucketName, oneWeekAgo);
} catch (AmazonClientException amazonClientException) {
```

```
    System.out.println("Unable to upload file, upload was aborted.");
    amazonClientException.printStackTrace();
}
tm.shutdownNow();
```

### 5.1.2.7 分块上传进度追踪

在使用 TransferManager 进行分块上传时，Java SDK 还提供了一个监听接口 ProgressListener 来追踪上传进度。

```
String filePath = "Path of the file to upload";
TransferManager tm = new TransferManager(client);
PutObjectRequest request = new PutObjectRequest(
bucketName, key, new File(filePath));
//Set the listener
request.setGeneralProgressListener(new ProgressListener() {
    public void progressChanged(ProgressEvent progressEvent) {
    System.out.println("Transferred bytes: " +
        progressEvent.getBytesTransferred());
        }
    });
Upload upload = tm.upload(request);
try {
    //  等待上传全部完成。
    upload.waitForCompletion();
    System.out.println("Upload complete.");
} catch (AmazonClientException amazonClientException) {
    System.out.println("Unable to upload file, upload was aborted.");
amazonClientException.printStackTrace();
}
// 在完成操作后，您必须显示关闭 TransferManager 。
tm.shutdownNow();
```

### 5.1.2.8 分块上传配置

在使用 TransferManager 进行上传时，您可以通过 TransferManagerConfiguration 进行配置。

```
TransferManager tm = new TransferManager(client);
TransferManagerConfiguration configuration = new TransferManagerConfiguration();
// 设置最小分片大小，默认是 5MB 。如果设置过小，会导致切片过多，影响上传速度。
configuration.setMinimumUploadPartSize(10*1024*1024L);
```

```
// 设置采用分片上传的阈值。只有当文件大于该值时，才会采用分片上传，否则采用普通上传。默认值 // 是 16MB 。
configuration.setMultipartUploadThreshold(100*1024*1024L);
tm.setConfiguration(configuration);
```

您也可以设置线程池大小，来设置并发数。

```
// 创建一个线程池，大小为 5
ExecutorService pool = Executors.newFixedThreadPool(5);
// 实例化 TransferManager ，并将上面创建的线程池作为参数。默认情况下， TransferManager 会创建一个大小为 10 的线程池。
TransferManager tm = new TransferManager(client, pool);
```

## 5.2 下载对象

SDK 提供了丰富的对象下载接口，包括了：流式下载、下载到本地文件、范围下载。此外，利用范围下载还可以实现断点下载功能以及并发下载功能。

### 5.2.1 流式下载

在进行大文件下载时，往往不希望一次性处理全部内容，而是希望流式地处理，一次处理一部分内容。

```
// 返回的结果中包含了元数据与一个输入流
S3Object s3Object= client.getObject(bucketName, objectKey);
BufferedReader reader = new BufferedReader(new
InputStreamReader(s3Object.getObjectContent()));
while (true) {
    String line = reader.readLine();
    if (line == null) break;
    System.out.println("\n" + line);
}
// 一定要显示 close ，否则会造成资源泄露
reader.close();
```

### 5.2.2 下载到本地文件

可以把 object 的内容下载到指定的本地文件中，如果指定的本地文件不存在，则会新建文件。

```
GetObjectRequest request = new GetObjectRequest(bucketName, objectKey);
// 将对象存在文件中，并返回对象的元数据
```

```
ObjectMetadata meta = client.getObject(request, new File("<localfile>"));
```

## 5.3 管理 object

### 5.3.1 判断对象是否存在

```
boolean exists = client.doesObjectExist(bucketName, objectKey);
```

### 5.3.2 列出 bucket 中对象

利用 listObjects 接口可以列取指定容器中的对象。 listObject 有三种形式：

- ObjectListing listObjects(String bucketName)
- ObjectListing listObjects(String bucketName, String prefix)
- ObjectListing listObjects(ListObjectsRequest listObjectsRequest)

前两种形式是简单模式，每次最多可以返回 1000 个对象。参数 prefix 是指返回 Object 的前缀。最后一种形式可以提供过滤，指定对象数，分页等功能。返回结果保存在 ObjectListing 中，主要包括了：

| 参数 | 含义 |
| --- | --- |
| ObjectSummaries | 返回的对象列表，包含了对象的元数据信息 |
| Prefix | 本次查询结果的开始前缀 |
| Delimiter | 是一个用于对 Object 名字进行分组的字符 |
| Marker | 标明这次 List Object 的起点 |
| MaxKeys | 响应请求内返回结果的最大数目 |
| NextMarker | 下一次 List Object 的起点 |
| IsTruncated | 指明是否所有的结果都已经返回 |
| CommonPrefixes | 如果请求中指定了 delimiter 参数，则返回结果中会包含<br>CommonPrefixes 元素。该元素标明以 delimiter 结尾，并有共同前缀<br>的 object 的集合。 |

#### 5.3.2.1 简单列举

```
ObjectListing objects = client.listObjects(bucketName);
for (S3ObjectSummary s3ObjectSummary: objects.getObjectSummaries())
{
```

```
    System.out.println(s3ObjectSummary.getKey());
}
```

#### 5.3.2.2 按指定前缀列举

```
ObjectListing objects = client.listObjects(bucketName, "prefix");
for (S3ObjectSummary s3ObjectSummary: objects.getObjectSummaries())
{
    System.out.println(s3ObjectSummary.getKey());
}
```

### 5.3.3 删除对象

#### 5.3.3.1 删除单个对象

可以使用 deleteObject 接口删除单个对象。

```
client.deleteObject(bucketName, objectKey);
```

#### 5.3.3.2 批量删除对象

您可以使用 deleteObjects 接口批量删除对象。每次最多删除 1000 个 Object，并提供两种返回模式：详细(verbose)模式和简单(quiet)模式:
详细模式：包括了成功的与失败的结果，默认模式；
简单模式：只返回删除失败的结果。

```
DeleteObjectsRequest request = new DeleteObjectsRequest(bucketNam
e);
List<KeyVersion> list = new ArrayList<KeyVersion>();
// 在开启多版本功能后，可以在 KeyVersion 中携带 verion id ，删除指定版本的对象
KeyVersion key1 = new KeyVersion("key1");
KeyVersion key2 = new KeyVersion("key2");
list.add(key1);
list.add(key2);
request.setKeys(list);
//true 是简单模式， false 是详细模式
request.setQuiet(true);
DeleteObjectsResult result = client.deleteObjects(request);
```

## 5.4 生成共享外链

利用 generatePresignedUrl 接口，可以为一个对象生成一个预签名的 URL 链接。使用浏览器访问该链接即可下载该对象。您可以将该链接共享给其他人，从而达到共享该文件的目的。

```
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequ
est(bucketName, objectKey);
// 设置过期时间，当到达该时间点时， URL 就会过期，其他人不再能访问该对象。
request.setExpiration(date);
URL url = client.generatePresignedUrl(request);
```

注意：

- 带有预签名的 URL 链接，都带有过期时间。
- 如果您希望链接不过期，可以将过期时间设置的很大，或将对象设置成公共可访问（访问时不需要签名）。