天翼云媒体存储

JAVA SDK 使用指导书

2024-10-18

天翼云科技有限公司

目 录

1. SDK 安装 3
1.1. JDK 版本要求 3
1.2. 安装方式 3
2. 初始化 5
2.1. 获取访问密钥 5
2.2. 获取 Endpoint 5
2.3. 新建 Client 5
3. 桶相关接口 8
3.1. 创建桶 8
3.2. 获取桶列表 9
3.3. 判断桶是否存在 10
3.4. 删除桶 11
3.5. 设置桶访问权限 11
3.6. 获取桶访问权限 14
3.7. 设置桶策略 15
3.8. 获取桶策略 19
3.9. 删除桶策略 20
3.10. 设置桶生命周期配置 21
3.11. 获取桶生命周期配置 25
3.12. 删除桶生命周期配置 28
3.13. 设置桶跨域访问配置 28
3.14. 获取桶跨域访问配置 30
3.15. 删除桶跨域访问配置 32
3.16. 设置桶版本控制状态 33
3.17. 获取桶版本控制状态 34
4. 对象相关接口 36
4.1. 获取对象列表 36
4.2. 上传对象 39
4.3. 下载对象 43
4.4. 复制对象 47
4.5. 删除对象 49
4.6. 批量删除对象 50
4.7. 获取对象元数据 51
4.8. 设置对象访问权限 53
4.9. 获取对象访问权限 55
4.10. 获取对象标签 57
4.11. 删除对象标签 58
4.12. 设置对象标签 59
4.13. 生成预签名 URL 60
4.14. 上传对象-Post 上传 62
4.15. 上传对象-追加写 64

4.16. 获取多版本对象列表 66
5. 分片上传接口 71
5.1. 融合接口 71
5.2. 分片上传-初始化分片上传任务 75
5.3. 分片上传-上传分片 77
5.4. 分片上传-合并分片 79
5.5. 分片上传-列举分片上传任务 81
5.6. 分片上传-列举已上传的分片 86
5.7. 分片上传-复制分片 89
5.8. 分片上传-取消分片上传任务 92
6. 安全凭证服务(STS) 94
6.1. 初始化 STS 服务 94
6.2. 获取临时 token 94
6.3. Policy 设置例子 96
6.4. 使用临时 token 97

1. SDK 安装

1.1. JDK 版本要求

天翼云媒体存储 Java SDK 要求使用 J2SE Development Kit 6.0 或更高版本。可以从 http://www.oracle.com/technetwork/java/javase/downloads/ 下载最新版本 Java。

注意：Java 版本 1.6 (JS2E 6.0) 中没有 SHA256 签名的 SSL 证书的内置支持。Java 版本 1.7 或更高版本包含已更新证书，不受这一问题的影响。

1.2. 安装方式

方式一：官网下载 Java-SDK

在天翼云官网下载 xos-java-sdk-2.1.1.jar，下载地址： xos-java-sdk-2.1.1.jar

在 jar 包所在目录下执行以下 maven 命令，安装至 maven 本地仓库。

```
mvn org.apache.maven.plugins:maven-install-plugin:3.0.0-M1:install-file -Dfile=xos-java-sdk-2.1.1.jar
```

在您的 maven 工程的 pom.xml 文件中，在“dependencies”节点中加入以下配置：

```
<dependency>
    <groupId>cn.chinatelecom</groupId>
    <artifactId>xos-java-sdk</artifactId>
    <version>2.1.1</version>
</dependency>
```

方式二：添加 AWS s3 依赖

天翼云媒体存储兼容 AWS s3 接口，您可以通过 AWS s3 接口使用天翼云媒体存储。若您需要使用 AWS s3 接口，在您的 maven 工程的 pom.xml 文件中，在“dependencies”节点中加入以下配置：

```
<dependency>
    <groupId>com.amazonaws</groupId>
```

```
    <artifactId>aws-java-sdk-s3</artifactId>
    <version>1.11.336</version>
</dependency>
<!-- 使用 sts 服务需要添加以下依赖 -->
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-sts</artifactId>
    <version>1.11.336</version>
</dependency>
```

2. 初始化

代码中的 accessKey 与 secretKey 是在媒体存储中创建用户后，系统分配给用户用于访问媒体存储的凭证。而 endpoint 是天翼云提供的服务端地址。

2.1. 获取访问密钥

AccessKey（AK）和 SecretAccessKey（SK）是用户访问媒体存储服务的密钥，密钥的管理和获取方式请查阅 天翼云媒体存储 密钥管理 。

2.2. 获取 Endpoint

EndPoint 的获取方式请查阅天翼云媒体存储 基础信息查看。

2.3. 新建 Client

使用 SDK 功能前，需要新建 Client，代码如下：

```
String accessKey = "<your-access-key>";
String secretKey = "<your-secret-access-key>";
String endPoint = "<your-endpoint>";
BasicAWSCredentials credentials = new BasicAWSCredentials(accessKey,
secretKey);
ClientConfiguration clientConfiguration = new ClientConfiguration();
EndpointConfiguration endpointConfiguration = new EndpointConfigurati
on(
        endpoint, Regions.DEFAULT_REGION.getName());
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
        //客户端设置
        .withClientConfiguration(clientConfiguration)
        //凭证设置
        .withCredentials(new AWSStaticCredentialsProvider(credential
s))
```

```
        //endpoint 设置
        .withEndpointConfiguration(endpointConfiguration)
        .build();
```

如果需要修改客户端的一些默认配置，请在创建 Client 的时候使用 withClientConfiguration 方法传入 ClientConfiguration 实例，一部分示例设置代码如下：

```
ClientConfiguration clientConfiguration = new ClientConfiguration();
//设置 client 的最大 HTTP 连接数
clientConfiguration.setMaxConnections(50);
//设置 Socket 层超时时间
clientConfiguration.setSocketTimeout(50000);
//设置建立连接的超时时间
clientConfiguration.setConnectionTimeout(50000);
```

ClientConguration 是客户端的配置类，可配置代理、连接超时、最大连接等参数。主要参数如下：

参数	描述	方法
maxConnections	允许打开的最大 HTTP 连接数。默认为 50	ClientConguration.setMaxConnections
socketTimeout	Socket 层传输数据的超时时间（单位：毫秒）。默认为 50,000 毫秒	ClientConguration.setSocketTimeout
connectionTimeout	建立连接的超时时间（单位：毫秒）。默认为 50,000 毫秒	ClientConguration.setConnectionTimeout
requestTimeout	等待请求完成的超时时间（单位：毫秒）。默认不超时	ClientConguration.setRequestTimeout
clientExecutionTimeo	客户端请求执行超时时	ClientConguration.setClientExecutionTime

参数	描述	方法
ut	间 （单位：毫秒）。默认不超时	out
maxErrorRetry	请求失败后最大的重试次数。默认 3 次	ClientConguration.setMaxErrorRetry
connectionMaxIdleMillis	如果空闲时间超过此参数的设定值，则关闭连接（单位：毫秒）。默认为 60,000 毫秒	ClientConguration.setConnectionMaxIdleMillis
signerOverride	若需要使用 v2 签名，可以设置为"S3SignerType"；若需要使用 v4 签名，可以设置为" AWSS3V4SignerType"。默认为 v4 签名	ClientConguration.setSignerOverride

3. 桶相关接口

3.1. 创建桶

3.1.1. 功能说明

桶（Bucket）是用于存储对象（Object）的容器，所有的对象都必须隶属于某个桶。您可以通过 createBucket 接口创建桶。

3.1.2. 代码示例

```
public void createBucket() throws AmazonClientException {
    String bucket = "<your-bucket-name>";
    System.out.println("createBucket");
    Bucket createdBucket = s3Client.createBucket(bucket);
    System.out.println("Bucket Name:" + createdBucket.getName());
}
```

3.1.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名称	是

3.1.4. 返回结果

参数	类型	说明
Bucket	Bucket	桶信息

关于 Bucket 一些说明

参数	类型	说明

参数	类型	说明
name	String	桶名称

3.2. 获取桶列表

3.2.1. 功能说明

桶（Bucket）是用于存储对象（Object）的容器，所有的对象都必须隶属于某个桶。您可以通过 listBuckets 接口获取桶列表信息。

3.2.2. 代码示例

```
public void listBuckets() throws AmazonClientException {
    System.out.println("listBuckets");
    List<Bucket> buckets = s3Client.listBuckets();
    for (Bucket bucket : buckets) {
        System.out.println("Bucket Name:" + bucket.getName());
        System.out.println("Bucket Creation Date:" + bucket.getCreati
onDate());
        System.out.println("Bucket Owner:" + bucket.getOwner());
    }
}
```

3.2.3. 返回结果

参数	类型	说明
buckets	List<Bucket>	桶信息列表

关于 Bucket 一些说明

参数	类型	说明
name	String	桶名称

参数	类型	说明
creationDate	Date	桶创建日期
owner	Owner	桶的 owner

3.3. 判断桶是否存在

3.3.1. 功能说明

桶（Bucket）是用于存储对象（Object）的容器，所有的对象都必须隶属于某个桶。您可以使用 doesBucketExist 接口判断桶是否存在。

3.3.2. 代码示例

```
public void doesBucketExist() throws AmazonClientException {
    String bucket = "<your-bucket-name>";
    System.out.println("doesBucketExist");
    boolean exists = s3Client.doesBucketExistV2(bucket);
    if (exists) {
        System.out.printf("Bucket %s already exists.\n", bucket);
    } else {
        System.out.printf("Bucket %s not exists.\n", bucket);
    }
}
```

3.3.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名称	是

3.3.4. 返回结果

参数	类型	说明
exists	boolean	桶是否存在

3.4. 删除桶

3.4.1. 功能说明

桶（Bucket）是用于存储对象（Object）的容器，所有的对象都必须隶属于某个桶。您可以通过 deleteBucket 接口删除桶。删除一个桶前，需要先删除该桶中的全部对象（包括对象版本）。

3.4.2. 代码示例

```
public void deleteBucket() throws AmazonClientException {
    String bucket = "<your-bucket-name>";
    s3Client.deleteBucket(bucket);
}
```

3.4.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名称	是

3.5. 设置桶访问权限

3.5.1. 功能说明

桶（Bucket）是用于存储对象（Object）的容器，所有的对象都必须隶属于某个桶。您可以通过 setBucketAcl 接口设置桶的访问权限。桶访问权限包含了 CannedAccesssControlList 与 AccessControlList 两种格式。CannedAccesssControlList 提供了预定义的权限控制，如私有读写，公共读私有写，公共读写，AccessControlList 提供了更

细致的权限控制，可自定义更多权限控制。用户在设置 bucket 的 ACL 之前需要具备 WRITE_ACP 权限。

3.5.2. 代码示例

- CannedAccesssControlList

```
public void setBucketAcl1() throws AmazonClientException {
    String bucket = "<your-bucket-name>";
    System.out.println("setBucketAcl");
    s3Client.setBucketAcl(bucket, CannedAccessControlList.PublicRead);
    System.out.println("setBucketAcl success");
}
```

- AccessControlList

```
public void setBucketAcl2() throws AmazonClientException {
    String bucket = "<your-bucket-name>";
    System.out.println("setBucketAcl");
    // 增加用户 exampleuser 的 Write 权限
    AccessControlList controlList = s3Client.getBucketAcl(bucket);
    CanonicalGrantee canonicalGrantee = new CanonicalGrantee("example
user");//开启用户 exampleuser 的 Write 权限
    controlList.grantPermission(canonicalGrantee,Permission.Write);

    s3Client.setBucketAcl(bucket, controlList);
    System.out.println("setBucketAcl success");
}
```

3.5.3. 请求参数

- CannedAccesssControlList

参数	类型	说明	是否必要
bucket	String	桶名称	是
CannedAccessControlList	CannedAccessControlList	桶权限	是

关于 CannedAccessControlList 的说明：

参数	说明
CannedAccessControlList.Private	私有读写
CannedAccessControlList.PublicRead	公共读私有写
CannedAccessControlList.PublicReadWrite	公共读写

- AccessControlList

使用 AccessControlList 设置桶访问权限时，可以设置特定用户对桶的访问权限。桶的 AccessControlList 权限如下表：

参数	类型	说明	是否必要
bucket	String	桶名称	是
controlList	AccessControlList	桶权限	是

在 AccessControlList 中可通过 grantAllPermission 传入 Grant 设置权限，Grant 中关于 Permission 说明如下：

参数	说明
Permission.Read	允许列出桶中的对象
Permission.Write	允许创建、覆盖、删除桶中的对象
Permission.ReadAcp	允许获取桶的 ACL 信息
Permission.WriteAcp	允许修改桶的 ACL 信息
Permission.FullControl	获得 READ、WRITE、READ_ACP、WRITE_ACP

参数	说明
	权限

3.6. 获取桶访问权限

3.6.1. 功能说明

桶（Bucket）是用于存储对象（Object）的容器，所有的对象都必须隶属于某个桶。您可以通过 getBucketAcl 接口获取桶的访问权限。

3.6.2. 代码示例

```
public void getBucketAcl() throws AmazonClientException {
    String bucket = "<your-bucket-name>";
    System.out.println("getBucketAcl");
    AccessControlList controlList = s3Client.getBucketAcl(bucket);
    List<Grant> grants = controlList.getGrantsAsList();
    System.out.println("getBucketAcl: owner=" + controlList.getOwner
());
    for (Grant grant: grants){
        System.out.println("getBucketAcl: grantee=" + grant.getGrante
e().getIdentifier()
                           + ", permission=" + grant.getPermission());
    }
}
```

3.6.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名称	是

3.6.4. 返回结果

返回的 AccessControlList 中包含的属性：

参数	类型	说明
owner	Owner	桶的 owner 信息
grants	List<Grant>	grants 授权信息，包含了每个用户与其权限 Permission。

关于 AccessControlList 中的访问权限说明可参考 设置桶访问权限 一节。

3.7. 设置桶策略

3.7.1. 功能说明

桶策略（Bucket Policy）可以灵活地配置用户各种操作和访问资源的权限。如果桶已经被设置了 policy，则新的 policy 会覆盖原有的 policy。关于 policy 的具体说明请参考 桶策略说明 。

您可以使用 setBucketPolicy 接口设定桶策略，您可以执行以下操作：

- 使用 JSON 格式的文本字符串直接指定策略。
- 使用 Policy 类构建策略，再转换为 JSON 格式的字符串。

您可以使用 Policy 类构建策略，免除设置文本字符串出现格式错误的麻烦，构建完成后使用 toJson 方法可以获取 JSON 策略文本。

policy 的示例如下：

```
{
    "Id":"PolicyId",
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ExampleStatementID1",
            "Principal":{
```

```
            "AWS":[
                "arn:aws:iam:::user/userId",
                "arn:aws:iam:::user/userName"
            ]
        },
        "Effect":"Allow",
        "Action":[
            "s3:ListBucket",
            "s3:CreateBucket"
        ],
        "Resource":[
            "arn:aws:iam:::exampleBucket"
        ],
        "Condition":"some conditions"
    },
    ......
]
}
```

Statement 的内容说明如下:

元素	描述	是否必要
Sid	statement Id，可选关键字，描述 statement 的字符串	否
Principal	可选关键字，被授权人，指定本条 statement 权限针对的 Domain 以及 User，支持通配符"*"，表示所有用户（匿名用户）。当对 Domain 下所有用户授权时，Principal 格式为 arn:aws:iam:::user/*。当对某个 User 进行授权时，Principal 格式为	可选，Principal 与 NotPrincipal 选其一

元素	描述	是否必要
	arn:aws:iam:::user/<your-user-name>	
NotPrincipal	可选关键字，不被授权人，statement 匹配除此之外的其他人。取值同 Principal	可选，NotPrincipal 与 Principal 选其一
Action	可选关键字，指定本条 statement 作用的操作，Action 字段为对象存储支持的所有操作集合，以字符串形式表示，不区分大小写。支持通配符“*”，表示该资源能进行的所有操作。例如：“Action”:[“s3:List*”, “s3:Get*”]	可选，Action 与 NotAction 选其一
NotAction	可选关键字，指定一组操作，statement 匹配除该组操作之外的其他操作。 取值同 Action	可选，NotAction 与 Action 选其一
Effect	必选关键字，指定本条 statement 的权限是允许还是拒绝，Effect 的值必须为 Allow 或者 Deny	必选
Resource	可选关键字，指定 statement 起作用的一组资源，支持通配符”*”，表示所有资源	可选，Resource 与 NotResource 选其一
NotResource	可选关键字，指定一组资源，statement 匹配除该组资源之外的其他资源。 取值同 Resource	可选，NotResource 与 Resource 选其一
Condition	可选关键字，本条 statement 生效的条件	可选

3.7.2. 代码示例

- 使用 JSON 格式的文本字符串直接指定策略。

```
public void setBucketPolicy1() throws AmazonClientException {
    // 以下示例策略允许 exampleuser1, exampleuser2 用户可读取桶内对象。
```

```
    String bucket = "<your-bucket-name>";
    System.out.println("setBucketPolicy");
    String bucketPolicy = "{\n" +
        "               \"Version\": \"2012-10-17\",\n" +
        "               \"Statement\": [{\n" +
        "                   \"Sid\": \"1\",\n" +
        "                   \"Effect\": \"Allow\",\n" +
        "                   \"Principal\": {\"AWS\": [\n" +
        "                                           \"arn:aws:iam:::u
ser/exampleuser1\",\n" +
        "                                           \"arn:aws:iam:::u
ser/exampleuser2\"\n" +
        "                                       ]},\n" +
        "                   \"Action\": [\"s3:GetObject\"],\n" +
        "                   \"Resource\": [\"arn:aws:s3:::" + bucket +
 "/*\"]\n" +
        "               }]\n" +
        "           }";
    s3Client.setBucketPolicy(bucket, bucketPolicy);
}
```

- 使用 Policy 类构建策略，再转换为 JSON 格式的字符串。

```
public void setBucketPolicy2() throws AmazonClientException {
    // 以下示例策略允许 exampleuser1, exampleuser2 用户可读取桶内对象。
    String bucket = "<your-bucket-name>";
    System.out.println("setBucketPolicy");
    Policy bucketPolicy = new Policy().withStatements(
        new Statement(Statement.Effect.Allow)//指定本条桶策略描述的权限
```

```
为允许请求
        .withPrincipals(Principal.AllUsers)
        .withPrincipals(new Principal("AWS", "arn:aws:iam:::user/exam
pleuser1"), //对 exampleuser1 用户生效
                        new Principal("AWS", "arn:aws:iam:::user/exampl
euser2"))//对 exampleuser2 用户生效
        .withActions(S3Actions.GetObject)//操作为 GetObject
        .withResources(new Resource("arn:aws:s3:::" + bucket + "/*
")));//指定资源为桶内所有对象
    s3Client.setBucketPolicy(bucket, bucketPolicy.toJson());
}
```

3.7.3. 参数说明

参数	类型	说明	是否必要
bucket	String	桶名称	是
bucketPolicy	String	桶策略的 JSON 格式字符串	是

3.8. 获取桶策略

3.8.1. 功能说明

桶策略（Bucket Policy）可以灵活地配置用户各种操作和访问资源的权限。您可以使用 getBucketPolicy 接口获取桶策略。

3.8.2. 代码示例

```
public void getBucketPolicy() throws AmazonClientException {
    String bucket = "<your-bucket-name>";
    System.out.println("getBucketPolicy");
```

```
    BucketPolicy policy = s3Client.getBucketPolicy(bucket);
    System.out.println("getBucketPolicy success, " + policy.getPolicy
Text());
}
```

3.8.3. 参数说明

参数	类型	说明	是否必要
bucket	String	桶名称	是

3.8.4. 返回结果

参数	类型	说明
policy	BucketPolicy	桶策略

关于返回的 BucketPolicy 的具体说明请参考 桶策略说明 。

3.9. 删除桶策略

3.9.1. 功能说明

桶策略（Bucket Policy）可以灵活地配置用户各种操作和访问资源的权限，您可以使用 deleteBucketPolicy 接口删除桶策略。

3.9.2. 代码示例

```
public void deleteBucketPolicy() throws AmazonClientException {
    String bucket = "<your-bucket-name>";
    System.out.println("deleteBucketPolicy");
    s3Client.deleteBucketPolicy(bucket);
}
```

3.9.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名称	是

3.10. 设置桶生命周期配置

3.10.1. 功能说明

生命周期管理可以通过设置规则实现自动清理过期的对象，优化存储空间。本文介绍如何设置桶（Bucket）生命周期规则。您可以通过 setBucketLifecycleConfiguration 操作可以设置桶的生命周期规则，规则可以通过匹配对象 key 前缀、标签的方法设置当前版本或者历史版本对象的过期时间，对象过期后会被自动删除。

桶的版本控制状态必须处于 Enabled 或者 Suspended，历史版本对象过期时间配置才能生效。每次执行 setBucketLifecycleConfiguration 操作会覆盖桶中已存在的生命周期规则。

3.10.2. 代码示例

```
public void setBucketLifecycleConfiguration() throws AmazonClientException {
    String bucket = "<your-bucket-name>";
    System.out.println("setBucketLifecycleConfiguration");
    BucketLifecycleConfiguration config = new BucketLifecycleConfiguration();
    List<BucketLifecycleConfiguration.Rule> rules = new ArrayList<BucketLifecycleConfiguration.Rule>();
    // mtime rule
    BucketLifecycleConfiguration.Transition transition = new BucketLifecycleConfiguration.Transition();
    transition.setDays(1);
    transition.setStorageClass(StorageClass.StandardInfrequentAccess);
```

```
    // atime rule
    BucketLifecycleConfiguration.AtimeTransition aTimeTransition = ne
w BucketLifecycleConfiguration.AtimeTransition();
    aTimeTransition.setDays(3);
    aTimeTransition.setStorageClass(StorageClass.StandardInfrequentAc
cess);
    aTimeTransition.setTransToStandard(true);
    aTimeTransition.setNoTransMaxSize(1024*1024);

    // incomplete upload rule
    AbortIncompleteMultipartUpload abort = new AbortIncompleteMultipa
rtUpload();
    abort.setDaysAfterInitiation(20);

    BucketLifecycleConfiguration.Rule rule = new BucketLifecycleConfi
guration.Rule();
    rule.setId("haha");
    rule.setExpirationInDays(10);
    rule.setAbortIncompleteMultipartUpload(abort);
    rule.setStatus(BucketLifecycleConfiguration.ENABLED);
    // ncv 过期时间
    rule.setNoncurrentVersionExpiration(new BucketLifecycleConfigurat
ion.NoncurrentVersionExpiration().withDays(365));
    // 设置应用范围，可以指定前缀或对象标签
    LifecycleFilterPredicate predicate = new LifecyclePrefixPredicate
("abc/");
    rule.setFilter(new LifecycleFilter(predicate));
```

```
    //LifecycleFilterPredicate predicate = new LifecycleTagPredicate
(new Tag("your-key", "your-value"));
    //rule.setFilter(new LifecycleFilter(predicate));
    rule.addTransition(transition);
    rule.addAtimeTransition(aTimeTransition);
    rules.add(rule);
    config.setRules(rules);
    SetBucketLifecycleConfigurationRequest req = new SetBucketLifecyc
leConfigurationRequest(bucket, config);
    s3Client.setBucketLifecycleConfiguration(req);
    System.out.println("setBucketLifecycleConfiguration success");
}
```

3.10.3. 请求参数

SetBucketLifecycleConfigurationRequest 的参数说明:

参数	类型	说明	是否必要
bucket	String	桶名称	是
BucketLifecycleConfiguration	BucketLifecycleConfiguration	生命周期设置	是

关于生命周期规则 Rule 的说明:

参数	类型	说明
id	String	规则 ID
status	String（Enabled|Disabled）	是否启用规则
expirationInDays	int	文件过期时间
noncurrentVersionExpiration	NoncurrentVersionExpiration	历史版本过期时间

参数	类型	说明
abortIncompleteMultipartUpload	AbortIncompleteMultipartUpload	未完成上传的分片过期时间
transition	Transition	文件转换到低频存储规则（距离修改时间）
atimeTransition	AtimeTransition	文件转换到低频存储规则（距离访问时间）
filter	LifecycleFilterPredicate	应用范围，可以指定前缀或对象标签

Transition：

参数	类型	说明
days	int	转换过期天数，默认-1，表示不转换
storageClass	StorageClass	要转换的存储类型

AtimeTransition：

参数	类型	说明
days	int	转换过期天数，默认-1，表示不转换
storageClass	StorageClass	要转换的存储类型
transToStandard	boolean	是否在下次访问时转换回标准存储，默认 false 表示不转回标准存储
noTransMaxSize	long	低于此大小的文件不进行转换，默认-1 表示所有文件都转换

NoncurrentVersionExpiration：

参数	类型	说明
days	int	过期天数

注意：atime 规则只在官网的 sdk 提供，AWS S3 没有 atime 规则。只有部分资源池支持 atime 规则。

3.11. 获取桶生命周期配置

3.11.1. 功能说明

生命周期管理可以通过设置规则实现自动清理过期的对象，优化存储空间。本文介绍如何查看桶（Bucket）的生命周期规则。

3.11.2. 代码示例

```
public void getBucketLifecycleConfiguration() throws AmazonClientExce
ption {
    String bucket = "<your-bucket-name>";
    System.out.println("getBucketLifecycleConfiguration");
    BucketLifecycleConfiguration config = s3Client.getBucketLifecycle
Configuration(bucket);
    List<BucketLifecycleConfiguration.Rule> rules = config.getRules();
    for (BucketLifecycleConfiguration.Rule rule: rules){
        System.out.println("getBucketLifecycleConfiguration: " + rule.
getId());
        System.out.println("getBucketLifecycleConfiguration expire: "
 +
                rule.getExpirationInDays());

        if(rule.getAbortIncompleteMultipartUpload() != null){
```

```
            System.out.println("getBucketLifecycleConfiguration abort:
 " +
                    rule.getAbortIncompleteMultipartUpload().getDaysAf
terInitiation());
        }

        if(rule.getNoncurrentVersionExpiration() != null) {
            System.out.println("getBucketLifecycleConfiguration ncv e
xpire days: " +
                    rule.getNoncurrentVersionExpiration().getDays());
        }

        LifecycleFilterPredicate predicate = rule.getFilter().getPred
icate();
        if (predicate instanceof LifecyclePrefixPredicate) {
            LifecyclePrefixPredicate prefixPredicate = (LifecyclePrefi
xPredicate)predicate;
            System.out.println("getBucketLifecycleConfiguration prefi
x filter: " +
                    prefixPredicate.getPrefix());
        }

        if (rule.getTransitions() != null) {
            for (BucketLifecycleConfiguration.Transition tran : rule.g
etTransitions()) {
                System.out.println("getBucketLifecycleConfiguration tr
an: " + tran.getStorageClassAsString() + "," +
                        tran.getDays());
```

```
            }
        }
        if (rule.getAtimeTransitions() != null) {
            for (BucketLifecycleConfiguration.AtimeTransition tran: ru
le.getAtimeTransitions()){
                System.out.println("getBucketLifecycleConfiguration at
ime tran: " + tran.getStorageClassAsString() + "," +
                        tran.getDays() + "," + tran.getTransToStandard()
 + "," + tran.getNoTransMaxSize());
            }
        }
    }
}
```

3.11.3. 请求参数

GetBucketLifecycleConfigurationRequest 的参数说明:

参数	类型	说明	是否必要
bucket	String	桶名称	是

3.11.4. 返回参数

参数	类型	说明
BucketLifecycleConfiguration	BucketLifecycleConfiguration	桶生命周期设置

关于 BucketLifecycleConfiguration 的生命周期说明可参考 设置桶生命周期 一节。

3.12. 删除桶生命周期配置

3.12.1. 功能说明

生命周期管理可以通过设置规则实现自动清理过期的对象，优化存储空间。本文介绍如何删除桶（Bucket）生命周期规则。

3.12.2. 代码示例

```
public void deleteBucketLifecycleConfiguration() throws AmazonClientE
xception {
    String bucket = "<your-bucket-name>";
    System.out.println("deleteBucketLifecycleConfiguration");
    s3Client.deleteBucketLifecycleConfiguration(bucket);
    System.out.println("deleteBucketLifecycleConfiguration success");
}
```

3.12.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名称	是

3.13. 设置桶跨域访问配置

3.13.1. 功能说明

跨域资源共享（CORS）定义了在一个域中加载的客户端 Web 应用程序与另一个域中的资源交互的方式。利用 CORS 支持，您可以构建丰富的客户端 Web 应用程序，同时可以选择性地允许跨源访问您的资源。您可以通过 setBucketCrossOriginConfiguration 接口设置桶的跨域访问配置。

3.13.2. 示例代码

```
public void setBucketCrossOriginConfiguration() throws AmazonClientEx
ception {
    String bucket = "<your-bucket-name>";
    System.out.println("setBucketCrossOriginConfiguration");
    List<CORSRule.AllowedMethods> rule2AM = new ArrayList<>();
    rule2AM.add(CORSRule.AllowedMethods.PUT);
    rule2AM.add(CORSRule.AllowedMethods.GET);
    rule2AM.add(CORSRule.AllowedMethods.HEAD);
    rule2AM.add(CORSRule.AllowedMethods.POST);
    rule2AM.add(CORSRule.AllowedMethods.DELETE);
    CORSRule rule = new CORSRule().withId("CORSRule").withAllowedMeth
ods(rule2AM)
        .withAllowedOrigins(Arrays.asList("*"))
        .withExposedHeaders(Arrays.asList("x-amz-server-side-encrypti
on", "ETag", "x-amz-request-id", "Date",
                                          "Content-Length", "Accept-Rang
es", "Access-Control-Allow-Origin"))
        .withAllowedHeaders(Arrays.asList("*"))
        .withMaxAgeSeconds(600);
    List<CORSRule> rules = new ArrayList<>();
    rules.add(rule);
    BucketCrossOriginConfiguration config = new BucketCrossOriginConf
iguration();
    config.setRules(rules);

    s3Client.setBucketCrossOriginConfiguration(bucket, config);
```

```
    System.out.println("setBucketCrossOriginConfiguration success");
}
```

3.13.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名称	是
BucketCrossOriginConfiguration	BucketCrossOriginConfiguration	跨域访问规则	是

关于 BucketCrossOriginConfiguration 一些说明

参数	说明
AllowedMethods	允许的请求方法
AllowedOrigins	允许的请求源
AllowedHeaders	允许的请求头
ExposedHeaders	允许返回的 Response Header
MaxAgeSeconds	跨域请求结果的缓存时间

3.14. 获取桶跨域访问配置

3.14.1. 功能说明

跨域资源共享（CORS）定义了在一个域中加载的客户端 Web 应用程序与另一个域中的资源交互的方式。利用 CORS 支持，您可以构建丰富的客户端 Web 应用程序，同时可以选择性地允许跨源访问您的资源。您可以通过 getBucketCrossOriginConfiguration 接口获取桶跨域访问配置。

3.14.2. 代码示例

```
public void getBucketCrossOriginConfiguration() throws AmazonClientEx
ception {
    String bucket = "<your-bucket-name>";
    System.out.println("getBucketCrossOriginConfiguration");
    BucketCrossOriginConfiguration config = s3Client.getBucketCrossOr
iginConfiguration(bucket);
    List<CORSRule> rules = config.getRules();
    for (CORSRule rule: rules){
        System.out.println("getBucketCrossOriginConfiguration: " + ru
le.getAllowedHeaders());
        System.out.println("getBucketCrossOriginConfiguration: " + ru
le.getAllowedOrigins());
        System.out.println("getBucketCrossOriginConfiguration: " + ru
le.getAllowedMethods());
        System.out.println("getBucketCrossOriginConfiguration: " + ru
le.getExposedHeaders());
        System.out.println("getBucketCrossOriginConfiguration: " + ru
le.getMaxAgeSeconds());
    }
}
```

3.14.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名称	是

3.14.4. 返回结果

参数	类型	说明
BucketCrossOriginConfiguration	BucketCrossOriginConfiguration	跨域访问规则

关于 BucketCrossOriginConfiguration 一些说明

参数	说明
AllowedMethods	允许的请求方法
AllowedOrigins	允许的请求源
AllowedHeaders	允许的请求头
ExposedHeaders	允许返回的 Response Header
MaxAgeSeconds	跨域请求结果的缓存时间

3.15. 删除桶跨域访问配置

3.15.1. 功能说明

跨域资源共享（CORS）定义了在一个域中加载的客户端 Web 应用程序与另一个域中的资源交互的方式。利用 CORS 支持，您可以构建丰富的客户端 Web 应用程序，同时可以选择性地允许跨源访问您的资源。您可以通过 deleteBucketCrossOriginConfiguration 接口删除桶跨域访问配置。

3.15.2. 代码示例

```
public void deleteBucketCrossOriginConfiguration() throws AmazonClien
tException{
    String bucket = "<your-bucket-name>";
    System.out.println("deleteBucketCrossOriginConfiguration");
    s3Client.deleteBucketCrossOriginConfiguration(bucket);
    System.out.println("deleteBucketCrossOriginConfiguration success
```

```
");
}
```

3.15.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名称	是

3.16. 设置桶版本控制状态

3.16.1. 功能说明

通过媒体存储提供的版本控制，您可以在一个桶中保留多个对象版本。例如，image.jpg(版本 1)和 image.jpg(版本 2)。如果您希望防止自己意外覆盖和删除版本，或存档对象，以便您可以检索早期版本的对象，您可以开启版本控制功能。

- 开启版本控制

  对桶中的所有对象启用版本控制，之后每个添加到桶中的对象都会被设置一个唯一的 version id。

- 暂停版本控制

  对桶中的所有对象暂停版本控制，之后每个添加到桶中的对象的 version ID 会被设置为 null。桶开启版本控制功能之后，无法再关闭该功能，只能暂停。

您可以使用 setBucketVersioningConfiguration 接口开启或暂停版本控制。

3.16.2. 示例代码

```
public void setBucketVersioningConfiguration() throws AmazonClientExc
eption {
    String bucket = "<your-bucket-name>";
    System.out.println("setBucketVersioningConfiguration");
    BucketVersioningConfiguration config = new BucketVersioningConfig
uration();
```

```
    config.setStatus(BucketVersioningConfiguration.ENABLED);
    SetBucketVersioningConfigurationRequest request = new SetBucketVe
rsioningConfigurationRequest(bucket, config);
    s3Client.setBucketVersioningConfiguration(request);
    System.out.println("setBucketVersioningConfiguration success");
}
```

3.16.3. 请求参数

SetBucketVersioningConfigurationRequest 的参数说明:

参数	类型	说明	是否必要
bucket	String	桶名称	是
BucketVersioningConfiguration	BucketVersioningConfiguration	版本控制设置	是

关于 BucketVersioningConfiguration 一些说明

参数	说明
BucketVersioningConfiguration.ENABLED	开启版本控制
BucketVersioningConfiguration.SUSPENDED	暂停版本控制

3.17. 获取桶版本控制状态

3.17.1. 功能说明

通过媒体存储提供的版本控制，您可以在一个桶中保留多个对象版本。您可以使用 getBucketVersioningConfiguration 接口获取版本控制配置。

3.17.2. 代码示例

```
public void getBucketVersioningConfiguration() throws AmazonClientExc
eption {
```

```
    String bucket = "<your-bucket-name>";
    System.out.println("getBucketVersioningConfiguration");
    BucketVersioningConfiguration config = s3Client.getBucketVersioni
ngConfiguration(bucket);
    System.out.println("getBucketVersioningConfiguration success, " +
 config.getStatus());
}
```

3.17.3. 参数说明

参数	类型	说明	是否必要
bucket	String	桶名称	是

3.17.4. 返回结果

参数	类型	说明
BucketVersioningConfiguration	BucketVersioningConfiguration	桶版本控制

关于 BucketVersioningConfiguration 一些说明。

参数	说明
BucketVersioningConfiguration.ENABLED	开启版本控制
BucketVersioningConfiguration.SUSPENDED	暂停版本控制
BucketVersioningConfiguration.OFF	关闭版本控制

4. 对象相关接口

4.1. 获取对象列表

4.1.1. 功能说明

您可以使用 listObjects 接口列举对象，每次最多返回 1000 个对象。

4.1.2. 代码示例

以下代码展示如何简单获取对象列表:

```
public void listObjects1() throws AmazonClientException {
    System.out.println("listObjects");
    String bucket = "<your-bucket-name>";
    ListObjectsRequest request = new ListObjectsRequest();
    request.setBucketName(bucket);
    // 过滤前缀
    //request.setPrefix("abc/");
    ObjectListing objects = s3Client.listObjects(request);
    for (S3ObjectSummary object : objects.getObjectSummaries()) {
        System.out.println(" - " + object.getKey());
    }
}
```

如果 list 大于 1000, 则返回的结果中 isTruncated 为 true, 通过返回的 nextMarker 标记可以作为下次读取的起点。列举所有对象示例代码如下:

```
public void listObjects2() throws AmazonClientException {
    System.out.println("listObjects");
    String bucket = "<your-bucket-name>";
    String nextMarker = null;
```

```
    ObjectListing objectListing;
    do {
        ListObjectsRequest listObjectsRequest = new ListObjectsReques
t();
        listObjectsRequest.withBucketName(bucket).withMarker(nextMark
er);
        objectListing = s3Client.listObjects(listObjectsRequest);
        for (S3ObjectSummary s3ObjectSummary: objectListing.getObject
Summaries()) {
            System.out.println(s3ObjectSummary.getKey());
        }
        nextMarker = objectListing.getNextMarker();
    } while (objectListing.isTruncated());
}
```

4.1.3. 请求参数

ListObjectsRequest 中可设置的参数如下:

参数	类型	说明	是否必须
bucketName	String	设置桶名称	是
encodingType	String	用于设置返回对象的字符编码类型	否
marker	String	指定列出对象读取对象的起点	否
maxKeys	int	设置 response 中返回对象的数量，默认值和最大值均为 1000	否
prefix	String	限定列举 objectKey 匹配前缀 prefix 的对	否

参数	类型	说明	是否必须
		象	
delimiter	String	与 prefix 参数一起用于对对象 key 进行分组的字符。所有 key 包含指定的 prefix 且第一次出现 Delimiter 字符的对象作为一组。如果没有指定 prefix 参数，按 delimiter 对所有对象 key 进行分割，多个对象分割后从对象 key 开始到第一个 delimiter 之间相同的部分形成一组	否

4.1.4. 返回结果

返回的 ObjectListing 属性如下：

属性名	类型	说明
commonPrefixes	List<String>	当请求中设置了 delimiter 和 prefix 属性时，所有包含指定的 Prefix 且第一次出现 delimiter 字符的对象 key 作为一组
objectSummaries	List<S3ObjectSummary>	对象数据，每个对象包含了 Entity Tag 、 Key 、 LastModifiedTime、Owner 和 Size 等信息。
delimiter	String	与请求中设置的 delimiter 一致
encodingType	String	返回对象 key 的字符编码类型

属性名	类型	说明
isTruncated	boolean	当为 false 时表示返回结果中包含了全部符合本次请求查询条件的对象信息，否则只返回了数量为 MaxKeys 个的对象信息
marker	String	与请求中设置的 Marker 一致
maxKeys	int	本次返回结果中包含的对象数量的最大值
bucketName	String	执行本操作的桶名称
nextMarker	String	当返回结果中的 IsTruncated 为 true 时，可以使用 NextMarker 作为下次查询的 Marker，继续查询出下一部分的对象信息
prefix	String	与请求中设置的 prefix 一致

4.2. 上传对象

4.2.1. 功能说明

您可以使用 putObject 接口上传对象。如果对同一个对象同时发起多个上传请求，最后一次完成的请求将覆盖之前所有请求的上传的对象。可以通过设置请求头部中的 Content-MD5 字段来保证数据被完整上传，如果上传的数据不能通过 MD5 校验，该操作将返回一个错误提示。用户可以通过比较上传对象后获得的 ETag 与原文件的 MD5 值是否相等来确认上传操作是否成功。

上传对象操作在上传对象时可以在请求里携带 HTTP 协议规定的 6 个请求头：Cache-Control 、 Expires 、 Content-Encoding 、 Content-Disposition 、 Content-Type 、 Content-Language。如果上传对象的请求设置了这些请求头，服务端会直接将这些头域的

值保存下来。这 6 个值也可以通过修改对象元数据操作进行修改。在该对象被下载或者执行 HeadObject 操作的时候，这些保存的值将会被设置到对应的 HTTP 头域中返回客户端。

PutObject 操作可以上传最大不超过 5GB 的文件，超过 5GB 的文件可以通过分片上传操作上传到对象存储（融合版）服务，对象 key 的命名使用 UTF-8 编码，长度必须在 1~1023 字节之间，不能以[/][\]字符开头。

4.2.2. 代码示例

- 文件上传

```
public void putObject1() throws AmazonClientException {
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";
    String localPath = "<your-local-path>";
    PutObjectResult ret = s3Client.putObject(bucket, key, new File(lo
calPath));
    System.out.println("putObject: " + ret.getETag());
}
```

- 流式上传

```
public void putObject2() throws AmazonClientException {
    System.out.println("putObject");
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";
    String content = "1234";
    byte[] contentBytes = content.getBytes();
    InputStream is = new ByteArrayInputStream(contentBytes);
    ObjectMetadata meta = new ObjectMetadata();
    //meta.setContentMD5(md5Base64(content));
    //meta.setContentLength(4);
```

```
    //meta.setContentType("text/plain");
    PutObjectRequest req = new PutObjectRequest(bucket, key, is, met
a);
    // 设置 acl
    req.setCannedAcl(CannedAccessControlList.PublicRead);
    // 设置存储类型
    //req.setStorageClass(StorageClass.StandardInfrequentAccess);
    // rate 上传限速，单位 KB
    //req.putCustomRequestHeader("x-amz-limit", String.format("rate=%
d", 10));
    PutObjectResult ret = s3Client.putObject(req);
    System.out.println("putObject: " + ret.getETag());
}

public String md5Base64(String data) {
    try {
        MessageDigest md = MessageDigest.getInstance("md5");
        byte[] md5 = md.digest(data.getBytes());
        BASE64Encoder be = new BASE64Encoder();
        return be.encode(md5);
    } catch (NoSuchAlgorithmException e) {}
    return "";
}
```

4.2.3. 请求参数

PutObjectRequest 可设置的参数如下：

参数	类型	说明	是否必要

参数	类型	说明	是否必要
bucket	String	执行本操作的桶名称	是
key	String	上传文件到对象存储（融合版）服务后对应的 key。PutObject 操作支持将文件上传至文件夹，如需要将对象上传至"/folder"文件下，只需要设置 Key="/folder/{exampleKey}" 即可	是
file	File	上传的本地文件	文件上传必须
inputStream	InputStream	流式上传的 Stream	流式上传必须
cannedAcl	CannedAccessControlList	配置上传对象的预定义的标准 ACL 信息，详细说明见设置对象访问权限 一节	否
storageClass	StorageClass	配置上传对象的存储类型，包括标准类型 STANDARD、低频类型 STANDARD_IA 以及归档类型 GLACIER	否
accessControlList	AccessControlList	配置上传对象的详细 ACL 信息，详细说明见设置 对象访问权限 一节	是
objectMetadata	ObjectMetadata	对象的元数据信息	否
tagging	ObjectTagging	对象的标签信息	否

您可以在上传对象时通过 ObjectMetadata 设置对象元数据。对象可设置的元数据如下:

方法	作用
ObjectMetadata.setContentType	设置 HTTP/HTTPS 请求头部中的

方法	作用
	Content-Type
ObjectMetadata.setContentLanguage	设置 HTTP/HTTPS 请求头部中的 Content-Language
ObjectMetadata.setCacheControl	设置 HTTP/HTTPS 请求头部中的 Cache-Control
ObjectMetadata.setContentDisposition	设置 HTTP/HTTPS 请求头部中的 Content-Disposition
ObjectMetadata.setContentEncoding	设置 HTTP/HTTPS 请求头部中的 Content-Encoding
ObjectMetadata.setContentLength	设置 HTTP/HTTPS 请求头部中的 Content-Length，设置请求 body 的长度（单位：字节）
ObjectMetadata.setContentMD5	对象数据的 MD5 值（经过 Base64 编码），提供给服务端校验数据完整性。

4.2.4. 返回结果

PutObjectResult 返回的属性如下：

参数	类型	说明
ETag	String	上传对象后对应的 Entity Tag
VersionId	String	上传对象后相应的版本 Id

4.3. 下载对象

4.3.1. 功能说明

您可以使用 getObject 接口下载对象。

4.3.2. 代码示例

```
public void getObject() throws AmazonClientException {
    System.out.println("getObject");
    try {
        String bucket = "<your-bucket-name>";
        String key = "<your-object-key>";
        long start = 0, end = 10;
        GetObjectRequest req = new GetObjectRequest(bucket, key);
        req.setRange(start, end); // 设置对象内容的范围，单位字节
        // 覆盖返回 header
        // ResponseHeaderOverrides header = new ResponseHeaderOverrid
es();
        // header.setContentDisposition("attachment; filename=testing.
txt");
        // req.setResponseHeaders(header);

        S3Object object = s3Client.getObject(req);

        S3ObjectInputStream s3is = object.getObjectContent();
        ObjectMetadata meta = object.getObjectMetadata();
        System.out.println("getobject: meta mymd5: " + meta.getUserMe
taDataOf("mymd5"));
        String content = IOUtils.toString(s3is);
        System.out.println("getobject: " + content);
        System.out.println("getobject header: " + object.getObjectMet
adata().getContentDisposition());
    } catch (IOException e) {
        e.printStackTrace();
```

```
    }
}
```

4.3.3. 请求参数

GetObjectRequest 参数

参数	类型	说明	是否必要
bucket	String	执行本操作的桶名称。	是
key	String	对象的 key。	是
range	long, long	下载对象指定范围内的数据（单位：字节），setRange(start, end) 表示前 2 字节的数据，详情请参见 RFC2616	否
versionId	String	当 bucket 开启版本控制的时候，用于指定获取指定版本的对象数据，当不指定该参数的时候，默认获取最新版本的对象数据	否
responseHeaders	ResponseHeaderOverrides	重写 HTTP/HTTPS 响应头信息	否
modifiedSinceConstraint	Date	如果指定的时间早于实际修改时间，则正常传送。否则返回错误	否
unmodifiedSinceConstraint	Date	如果传入参数中的时间等于或者晚于文件实际修改时	否

参数	类型	说明	是否必要
		间，则正常传输文件；否则返回错误	
matchingETagConstraints	List<String>	如果传入的 ETag 和 Object 的 ETag 匹配，则正常传输；否则返回错误	否
nonmatchingEtagConstraints	List<String>	如果传入的 ETag 值和 Object 的 ETag 不匹配，则正常传输；否则返回错误	否

通过 ResponseHeaderOverrides 可重写部分 HTTP/HTTPS 响应头信息。可重写的响应头信息见下表：

参数	类型	作用
contentType	String	重写 HTTP/HTTPS 响应中的 Content-Type
contentLanguage	String	重写 HTTP/HTTPS 响应中的 Content-Language
expires	String	重写 HTTP/HTTPS 响应中的 Expires
cacheControl	String	重写 HTTP/HTTPS 响应中的 Cache-Control
contentDisposition	String	重写 HTTP/HTTPS 响应中的 Content-Disposition
contentEncoding	String	重写 HTTP/HTTPS 响应中的 Content-Encoding

4.3.4. 返回结果

返回的 S3Object 属性如下：

参数	类型	说明
objectContent	S3ObjectInputStream	对象的数据流
metadata	ObjectMetadata	对象的元数据
taggingCount	int	对象标签的数量
bucket	String	对象所属的桶
key	String	对象 key

4.4. 复制对象

4.4.1. 功能说明

您可以使用 copyObject 接口复制对象，您需要设置复制的对象名，所在的桶以及目标桶和对象名。

4.4.2. 代码示例

复制一个对象

```
public void copyObject() throws AmazonClientException {
    String destBucketName = "<your-bucket-name>";
    String destObjectKey = "<your-object-key>";
    String sourceBucketName = "<source-bucket-name>";
    String sourceObjectKey = "<source-object-key>";
    ObjectMetadata metadataCopy = new ObjectMetadata();
    metadataCopy.setContentType("text/json");
    CopyObjectRequest request = new CopyObjectRequest(sourceBucketNam
e, sourceObjectKey, destBucketName, destObjectKey);
    // request.setStorageClass(StorageClass.Standard); // 设置对象的存
储类型
    // 如果源对象是归档存储类型，需要先解冻对象
```

```
    // ObjectUnfreezeSDK unfreezeSDK = new ObjectUnfreezeSDK(accessKe
y, secretKey, endPoint);
    // System.out.println(unfreezeSDK.unfreeze(oriBucket, oriKey));
    // Thread.sleep(60000); // 等待解冻完成
    CopyObjectResult result = s3Client.copyObject(request);
    System.out.println("CopyObject success" + result.getETag());
}
```

对象存储（融合版）没有直接修改对象元数据的方法，上传文件之后就不能修改了，可以使用 copyObject 接口修改对象元数据，注意：源 bucket 和目的 bucket 一致，源 key 和目的 key 一致。

```
public void changeMetadataViaCopyObject() throws AmazonClientExceptio
n {
    String bucket = "<your-bucket-name>";
    String sourceObjectKey = "<your-object-key>";
    String destObjectKey = "<your-object-key>";        // 复制到原来的
key
    ObjectMetadata metadataCopy = new ObjectMetadata();
    metadataCopy.setContentType("text/json");
    CopyObjectRequest request = new CopyObjectRequest(bucket, sourceO
bjectKey, bucket, destObjectKey)
        .withNewObjectMetadata(metadataCopy);
    CopyObjectResult result = s3Client.copyObject(request);
    System.out.println("changeMetadataViaCopyObject success, Etag:" +
 result.getETag());
}
```

文件比较大（超过 1GB）的情况下，直接使用 copyObject 可能会出现超时，需要使用分片复制的方式进行文件复制，TransferManager 封装了分片复制的接口，可以用于复制文件，具体示例请参考 分片上传融合接口 中的使用 TransferManager 进行分片复制部分。

4.4.3. 请求参数

参数	类型	说明	是否必要
bucket	String	源桶	是
oriKey	String	源对象 key	是
destBucket	String	目的桶	是
destKey	String	目的对象 key	是
storageClass	StorageClass	配置目的对象的存储类型，包括标准类型 STANDARD、低频类型 STANDARD_IA 以及归档类型 GLACIER	否

4.4.4. 返回结果

返回的 CopyObjectRequest 属性

参数	类型	说明
ETag	string	对象的唯一标签

4.5. 删除对象

4.5.1. 功能说明

您可以使用 deleteObject 接口删除单个对象。

4.5.2. 代码示例

```
public void deleteObject() throws AmazonClientException {
    System.out.println("deleteObject");
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";
    s3Client.deleteObject(bucket, key);
    System.out.println("deleteObject success");
}
```

4.5.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名	是
key	String	对象名	是

4.6. 批量删除对象

4.6.1. 功能说明

您可以使用 deleteObjects 接口批量删除多个对象，提供两种返回模式：详细(verbose)模式和简单(quiet)模式，详细模式包括成功与失败的结果，简单模式只返回失败的结果，默认为详细模式。

4.6.2. 代码示例

```
public void deleteObjects() throws AmazonClientException {
    System.out.println("deleteObjects");
    String bucket = "<your-bucket-name>";
    String [] keys = {"<your-object-key1>", "<your-object-key2>"};
    DeleteObjectsRequest request = new DeleteObjectsRequest(bucket);
```

```
    request.withKeys(keys);
    DeleteObjectsResult ret = s3Client.deleteObjects(request);
    List<DeleteObjectsResult.DeletedObject> list = ret.getDeletedObje
cts();
    for (DeleteObjectsResult.DeletedObject del: list){
        System.out.println("deleteObjects: " + del.getKey());
    }
}
```

4.6.3. 请求参数

DeleteObjectsRequest 可设置的参数如下:

参数	类型	说明	是否必要
bucket	String	执行本操作的桶名称	是
keys	String[]	要删除的对象 key	是

4.6.4. 返回结果

返回的 List<DeleteObjectsResult.DeletedObject> 可获取被删除的对象 key

4.7. 获取对象元数据

4.7.1. 功能说明

您可以使用 getObjectMetadata 接口获取对象元数据。getObjectMatadata 操作的请求参数与 getObject 一样，但是 getObjectMetadata 返回的 http 响应中没有对象数据。

4.7.2. 代码示例

```
public void getObjectMetadata() throws AmazonClientException {
    System.out.println("getObjectMetadata");
```

```
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";
    ObjectMetadata ret = s3Client.getObjectMetadata(bucket, key);
    System.out.println("getObjectMetadata content-length: " + ret.get
ContentLength());
}
```

4.7.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名	是
key	String	对象名	是

4.7.4. 返回结果

返回的 ObjectMetadata 属性如下：

参数	类型	说明
contentLength	long	本次请求返回对象数据的大小（单位：字节）
contentType	String	对象文件格式的标准 MIME 类型
eTag	String	对象的 Entity Tag
lastModified	Date	最近一次修改对象的时间。
versionId	String	对象最新的版本 ID。

4.8. 设置对象访问权限

4.8.1. 功能说明

与桶访问权限类似，对象访问权限同样具有 CannedAccessControlList 与 AccessControlList 两种。需要注意的是，对象的访问优先级要高于桶访问权限。比如桶访问权限是 private，但是对象访问权限是 public read，则所有用户都可以访问该对象。默认情况下，只有对象的拥有者才能访问该对象，即对象的访问权限默认是 private。设置对象 ACL 操作需要具有对象的 WRITE_ACP 权限。

4.8.2. 代码示例

- CannedAccesssControlList

CannedAccesssControlList 格式的对象访问权限包含了：Private(私有读写)，PublicRead（公共读私有写），PublicReadWrite（公共读写）。使用 CannedAccesssControlList 设置桶的访问权限示例代码如下:

```
public void setObjectAcl1() throws AmazonClientException {
    System.out.println("setObjectAcl");
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";

    s3Client.setObjectAcl(bucket, key, CannedAccessControlList.PublicRead);
    System.out.println("setObjectAcl success");
}
```

- AccessControlList

使用 AccessControlList 设置对象访问权限时，可以设置特定用户对象的访问权限。使用 AccesssControlList 设置对象的权限示例代码如下:

```
public void setObjectAcl2() throws AmazonClientException {
    System.out.println("setObjectAcl");
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";
    // 增加用户 exampleuser 的 Write 权限
    AccessControlList controlList = s3Client.getObjectAcl(bucket, ke
y);
    CanonicalGrantee canonicalGrantee = new CanonicalGrantee("example
user");//开启用户 exampleuser 的 Write 权限
    controlList.grantPermission(canonicalGrantee,Permission.Write);

    s3Client.setObjectAcl(bucket, key, controlList);
    System.out.println("setObjectAcl success");
}
```

4.8.3. 请求参数

- CannedAccesssControlList

参数	类型	说明	是否必要
bucket	String	桶名称	是
key	String	对象 key	是
CannedAccessControlList	CannedAccessControlList	对象权限	是

关于 CannedAccessControlList 的说明:

参数	说明
CannedAccessControlList.Private	私有读写
CannedAccessControlList.PublicRead	公共读私有写

参数	说明
CannedAccessControlList.PublicReadWrite	公共读写

- AccessControlList

使用 AccessControlList 设置桶访问权限时，可以设置特定用户对桶的访问权限。桶的 AccessControlList 权限如下表：

参数	类型	说明	是否必要
bucket	String	桶名称	是
key	String	对象 key	是
controlList	AccessControlList	对象权限	是

在 AccessControlList 中可通过 grantAllPermission 传入 Grant 设置权限，Grant 中关于 Permission 说明如下：

参数	说明
Permission.Read	允许读取对象数据和元数据
Permission.Write	不可作用于对象
Permission.ReadAcp	允许获取对象的 ACL 信息
Permission.WriteAcp	允许修改对象的 ACL 信息
Permission.FullControl	获得 READ、READ_ACP、WRITE_ACP 权限

4.9. 获取对象访问权限

4.9.1. 功能说明

您可以使用 getObjectAcl 接口获取对象访问的权限。

4.9.2. 代码示例

```
public void getObjectAcl() throws AmazonClientException {
    System.out.println("getObjectAcl");
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";
    AccessControlList ret = s3Client.getObjectAcl(bucket, key);
    List<Grant> grants = ret.getGrantsAsList();
    System.out.println("getObjectAcl: owner=" + ret.getOwner());
    for (Grant grant: grants){
        System.out.println("getObjectAcl: grantee=" + grant.getGrante
e().getIdentifier()
                           + ", permission=" + grant.getPermission());
    }
}
```

4.9.3. 请求参数

参数	类型	说明	是否必要
bucket	String	对象所属桶的名称	是
key	String	对象的 key	是
versionId	String	设置标签信息的对象的版本 Id	否

4.9.4. 返回结果

返回的 AccessControlList 中包含的属性:

参数	类型	说明
owner	Owner	对象的 owner 信息

参数	类型	说明
grants	List<Grant>	grants 授权信息，包含了每个用户与其权限 Permission

关于 AccessControlList 中的访问权限说明可参考 设置对象访问权限 一节。

4.10. 获取对象标签

4.10.1. 功能说明

您可以使用 getObjectTagging 接口获取对象标签。

4.10.2. 代码示例

```
public void getObjectTagging() throws AmazonClientException{
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";
    GetObjectTaggingRequest request = new GetObjectTaggingRequest(bucket, key);
    GetObjectTaggingResult config = s3Client.getObjectTagging(request);
    for (Tag tag: config.getTagSet()){
        System.out.println("getObjectTagging success, tags: " + tag.getKey() + ":" + tag.getValue());
    }
}
```

4.10.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名称	是

参数	类型	说明	是否必要
key	String	对象 key	是
versionId	String	设置标签信息的对象的版本 Id	否

4.10.4. 返回结果

参数	类型	说明
tagging	ObjectTagging	设置的标签信息，包含了一个 Tag 结构体的数组，每个 Tag 以 Key-Value 的形式说明了标签的内容

4.11. 删除对象标签

4.11.1. 功能说明

您可以使用 deleteObjectTagging 接口删除对象标签。

4.11.2. 代码示例

```
public void deleteObjectTagging() throws AmazonClientException{
    System.out.println("deleteObjectTagging");
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";
    DeleteObjectTaggingRequest request = new DeleteObjectTaggingReque
st(bucket, key);
    s3Client.deleteObjectTagging(request);
    System.out.println("deleteObjectTagging success");
}
```

4.11.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名称	是
key	String	对象 key	是
versionId	String	设置标签信息的对象的版本 Id	否

4.12. 设置对象标签

4.12.1. 功能说明

您可以使用 setObjectTagging 接口为对象设置标签。标签是一个键值对，每个对象最多可以有 10 个标签。bucket 的拥有者默认拥有给 bucket 中的对象设置标签的权限，并且可以将权限授予其他用户。每次执行 PutObjectTagging 操作会覆盖对象已有的标签信息。每个对象最多可以设置 10 个标签，标签 Key 和 Value 区分大小写，并且 Key 不可重复。每个标签的 Key 长度不超过 128 字节，Value 长度不超过 256 字节。SDK 通过 HTTP header 的方式设置标签且标签中包含任意字符时，需要对标签的 Key 和 Value 做 URL 编码。设置对象标签信息不会更新对象的最新更改时间。

4.12.2. 代码示例

```
public void setObjectTagging() throws AmazonClientException{
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";
    List<Tag> tagList = new ArrayList<>();
    tagList.add(new Tag("<your-tag-key1>", "<your-tag-value1>"));
    tagList.add(new Tag("<your-tag-key2>", "<your-tag-value2>"));

    ObjectTagging config = new ObjectTagging(tagList);
```

```
    SetObjectTaggingRequest request = new SetObjectTaggingRequest(buc
ket, key, config);
    s3Client.setObjectTagging(request);
    System.out.println("setObjectTagging success");
}
```

4.12.3. 请求参数

参数	类型	说明	是否必要
bucket	String	桶名称	是
key	String	对象 key	是
tagging	ObjectTagging	设置的标签信息，包含了一个 Tag 结构体的数组，每个 Tag 以 Key-Value 的形式说明了标签的内容	是
versionId	String	设置标签信息的对象的版本 Id	否

4.13. 生成预签名 URL

4.13.1. 功能说明

您可以利用 GeneratePresignedUrl 接口为一个对象生成一个预签名的 URL 链接。

4.13.2. 代码示例

- 生成 getObject 的预签名 URL

```
public void generateGetPresignedUrl() throws AmazonClientException {
    System.out.println("generateGetPresignedUrl");
    String bucket = "<your-bucket-name>";
```

```
    String key = "<your-object-key>";


    Date now = new Date();
    Calendar newTime = Calendar.getInstance();
    newTime.setTime(now);
    newTime.add(Calendar.SECOND, 900);
    Date expire = newTime.getTime();
    GeneratePresignedUrlRequest request = new GeneratePresignedUrlReq
uest(bucket, key);
    // 设置下载的文件名和 content-type
    //ResponseHeaderOverrides responseHeader = new ResponseHeaderOver
rides();
    //responseHeader.setContentDisposition("inline;filename=\"1.txt\"
");
    //responseHeader.setContentType("text/plain");
    //request.setResponseHeaders(responseHeader);
    request.withExpiration(expire);
    // concurrency 并发限制， rate 下载限速，单位 KB
    request.addRequestParameter("x-amz-limit", String.format("rate=%d
", 100));
    //request.addRequestParameter("x-amz-limit", String.format("concu
rrency=%d,rate=%d", 1, 100));
    URL ret = s3Client.generatePresignedUrl(request);
    System.out.println("generateGetPresignedUrl: " + ret);
    // 预签名 url 支持使用 range 头进行分片下载
}
```

4.13.3. 请求参数

参数	类型	描述
bucketName	String	桶名
key	String	对象名
expiration	Date	过期时间，默认 900 秒

4.14. 上传对象-Post 上传

4.14.1. 功能说明

ObjectPostSDK 接口为一个指定对象生成一个支持 post 方式上传文件的参数集合，可以在前端使用 post form-data 的方式上传文件。ObjectPostSDK 实现请查看 Java SDK Demo 。

4.14.2. 代码示例

```
public void postObject() throws Exception {
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";
    String localFilePath = "<your-local-file-path>"";

    ObjectPostSDK.PostObjectPolicy policy = new ObjectPostSDK.PostObj
ectPolicy(900);
    policy.addEqualCondition("acl", "public-read");
    ObjectPostSDK.PostObjectData data = sdk.generatePostObjectData(bu
cket, key, policy);
    System.out.println("Policy:" + policy.getPolicy());
    Map<String, String> formFields = data.getFormFields();
    formFields.put("acl", "public-read");
```

```
    for (Entry<String, String> entry: formFields.entrySet()){
        System.out.println(entry.getKey() + ":" + entry.getValue());
    }
    // String storageClass = StorageClass.Standard.toString(); // 设置对象存储类型
    // String ret = formUpload(data.getUrl(), data.getFormFields(), localFilePath, storageClass);
    String ret = formUpload(data.getUrl(), data.getFormFields(), localFilePath);
    System.out.println("Post Object [" + objectKey + "] to bucket [" + bucket + "]");
    System.out.println("post response:" + ret);
}
```

4.14.3. 请求参数

参数	类型	说明	是否必要
bucket	字符串	bucket 的名称	是
key	字符串	对象的 key	是
expires	整型数	超时时间（秒）	否，默认 900 秒
storageClass	字符串	配置上传对象的存储类型，包括标准类型 STANDARD、低频类型 STANDARD_IA 以及归档类型 GLACIER	否，默认为 STANDARD

前端使用方式如下：

```
<form action="<data.url>" method="POST" enctype="multipart/form-data">
    <input type="hidden" name="Policy" value="<data.fields['Policy']>
```

```
" />
    <input type="hidden" name="X-Amz-Algorithm" value="<data.fields['
X-Amz-Algorithm']>" />
    <input type="hidden" name="X-Amz-Credential" value="<data.fields
['X-Amz-Credential']>" />
    <input type="hidden" name="X-Amz-Date" value="<data.fields['X-Amz
-Date']>" />
    <input type="hidden" name="X-Amz-Signature" value="<data.fields['
X-Amz-Signature']>" />
    <input type="hidden" name="bucket" value="<data.fields['bucket']>
" />
    <input type="hidden" name="key" value="<data.fields['key']>" />

    <input type="file" name="file" value="" />
    <input type="submit" value="Submit" />
</form>
```

4.15. 上传对象-追加写

4.15.1. 功能说明

PutObject 可以对桶中的一个对象进行追加写操作，如果该对象已经存在，执行该操作则向文件末尾追加内容，否则将创建对象。

通过 Append 操作创建的 Object 类型为 Appendable，而通过 PutObject 操作上传的 Object 的类型是 Normal。对 Appendable 类型的对象进行普通上传操作之后会覆盖原有对象的内容并且将其类型设置为 Normal。

Append 操作仅可以在未开启版本控制的桶中执行，如果桶的版本控制状态为启用（Enabled）或者暂停（Suspended）状态将不支持 Append 操作。

4.15.2. 代码示例

```
public void putObjectAppend() throws AmazonServiceException {
    System.out.println("putObjectAppend");
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";

    long contentLength = 0;
    try {
        // 获取原始文件长度,也可以使用业务自己的接口获取
        ObjectMetadata meta = s3Client.getObjectMetadata(bucket, key);
        contentLength = meta.getContentLength();
    } catch (AmazonServiceException e){

    }

    String content = "123";
    byte[] contentBytes = content.getBytes();
    InputStream is = new ByteArrayInputStream(contentBytes);
    ObjectMetadata metadata = new ObjectMetadata();
    metadata.setContentType("text/plain");

    AppendObjectRequest req = new AppendObjectRequest(bucket, key, is,
 metadata);
    // req.setStorageClass(StorageClass.Standard); // 设置对象的存储类
型
    req.setPosition(contentLength);
    AppendObjectResult ret = s3Client.appendObject(req);
```

```
    System.out.println("putObjectAppend success" + ret.getETag());
}
```

4.15.3. 请求参数

AppendObjectRequest 中可设置的参数如下:

参数	说明	是否必须
bucket	桶名称	是
key	对象名称	是
position	追加前对象大小	是
storageClass	对象的存储类型	否

4.15.4. 返回结果

AppendObjectResult 返回的属性如下:

参数	类型	说明
ETag	String	上传对象后对应的 Entity Tag

4.16. 获取多版本对象列表

4.16.1. 功能说明

如果桶开启了版本控制，您可以使用 listVersions 接口列举对象的版本，每次 list 操作最多返回 1000 个对象版本。

4.16.2. 代码示例

简单列举对象版本代码如下:

```
public void listVersions() throws AmazonClientException {
    System.out.println("listVersions");
    String bucket = "<your-bucket-name>";
```

```
    ListVersionsRequest request = new ListVersionsRequest();
    request.setBucketName(bucket);
    VersionListing list = s3Client.listVersions(request);
    for (S3VersionSummary obj: list.getVersionSummaries()){
        System.out.println("key: " + s3VersionSummary.getKey());
        System.out.println("versionId: " + s3VersionSummary.getVersio
nId());
    }
}
```

如果 list 大于 1000, 则返回的结果中 isTruncated 为 true, 通过返回的 NextKeyMarker 和 NextUploadIdMarker 标记可以作为下次读取的起点。列举所有对象版本示例代码如下:

```
public void listVersions2() throws AmazonClientException {
    System.out.println("listVersions");
    String bucket = "<your-bucket-name>";
    String nextMarker = null;
    String nextVersionIdMarker = null;
    VersionListing versionListing;
    do {
        ListVersionsRequest listVersionsRequest = new ListVersionsReq
uest();
        listVersionsRequest.withBucketName(bucket).withKeyMarker(next
Marker).withVersionIdMarker(nextVersionIdMarker);
        versionListing = s3Client.listVersions(listVersionsRequest);
        for (S3VersionSummary s3VersionSummary: versionListing.getVer
sionSummaries()) {
            System.out.println("key: " + s3VersionSummary.getKey());
            System.out.println("versionId: " + s3VersionSummary.getVer
```

```
sionId());
            }
            nextMarker = versionListing.getNextKeyMarker();
            nextVersionIdMarker = versionListing.getNextVersionIdMarker();
        } while (versionListing.isTruncated());

    }
```

4.16.3. 请求参数

ListVersionsRequest 中可设置的参数如下：

参数	类型	说明	是否必须
bucketName	String	设置桶名称	是
encodingType	String	用于设置返回对象的字符编码类型	否
keyMarker	String	指定列出对象版本读取对象的起点	否
versionIdMarker	String	指定列出对象版本读取对象的版本的起点	否
maxKeys	int	设置 response 中返回对象的数量，默认值和最大值均为 1000	否
prefix	String	限定列举 objectKey 匹配前缀 prefix 的对象	否
delimiter	String	与 prefix 参数一起用于对对象 key 进行分组的字符。所有 key 包含指定的 prefix 且第一次出现 Delimiter	否

参数	类型	说明	是否必须
		字符的对象作为一组。如果没有指定 prefix 参数，按 delimiter 对所有对象 key 进行分割，多个对象分割后从对象 key 开始到第一个 delimiter 之间相同的部分形成一组	

4.16.4. 返回结果

返回的 VersionListing 属性如下：

属性名	类型	说明
commonPrefixes	List<String>	当请求中设置了 delimiter 和 prefix 属性时，所有包含指定的 Prefix 且第一次出现 delimiter 字符的对象 key 作为一组
versionSummaries	List<S3VersionSummary>	对象版本数据，每个对象包含了 Entity Tag、Key、VersionId、LastModifiedTime、Owner 和 Size 等信息
delimiter	String	与请求中设置的 delimiter 一致
encodingType	String	返回对象版本的字符编码类型
isTruncated	boolean	当为 false 时表示返回结果中包含了全部符合本次请求查询条件的对象版本信息，否则只返回了数量为 MaxKeys 个的对象版本信息

属性名	类型	说明
marker	String	与请求中设置的Marker一致
maxKeys	int	本次返回结果中包含的对象版本数量的最大值
bucketName	String	执行本操作的桶名称
nextMarker	String	当返回结果中的 IsTruncated 为 true 时，可以使用 NextMarker 作为下次查询的 Marker，继续查询出下一部分的对象信息
prefix	String	与请求中设置的 prefix 一致

5. 分片上传接口

5.1. 融合接口

5.1.1. 功能说明

分片上传步骤较多，包括初始化、文件切片、各个分片上传、完成上传。为了简化分片上传，Java SDK 提供了分片上传的封装接口。您可以调用 TransferManager 接口，快速实现文件的分片上传与分片的管理。文件的上传时，TransferManager 会采用多线程的方式，同时进行多个文件的上传。

5.1.2. 代码示例

- 使用 TransferManager 分片上传

```
public void upload() {
    try {
        String bucket = "<your-bucket-name>";
        String key = "<your-object-key>";
        String localPath = "<your-local-path>";

        // TransferManager 只需要初始化一次，可以用于多个上传任务
        TransferManager transMgr = TransferManagerBuilder.standard()
            .withS3Client(s3Client)
            // 设置最小分片大小，默认是 5MB。
            .withMinimumUploadPartSize(10*1024*1024L)
            // 设置采用分片上传的阈值为 100MB。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。
            .withMultipartUploadThreshold(100*1024*1024L)
            .build();
```

```
        // TransferManager 采用异步方式进行处理，因此该调用会立即返回。
        PutObjectRequest request = new PutObjectRequest(bucket, key,
new File(localPath));
        request.withCannedAcl(CannedAccessControlList.Private);     /
/ 设置对象 ACL，可以设置公共读 CannedAccessControlList.PublicRead
        // request.setStorageClass(StorageClass.Standard);
 // 设置对象的存储类别
        ObjectMetadata meta = new ObjectMetadata();
        meta.setContentType("application/octet-stream");            //
 设置 Content-Type，默认 application/octet-stream
        request.setMetadata(meta);                                  //
还可以设置其他自定义元数据

        Upload upload = transMgr.upload(request);
        // 等待上传全部完成。
        UploadResult result = upload.waitForUploadResult();
        System.out.println("upload success, etag=" + result.getETag
());
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

- 使用 TransferManager 分片复制

```
public void copy() {
    try {
        String destBucketName = "<your-bucket-name>";
```

```
        String destObjectKey = "<your-object-key>";
        String sourceBucketName = "<source-bucket-name>";
        String sourceObjectKey = "<source-object-key>";

        // TransferManager 只需要初始化一次，可以用于多个上传任务
        TransferManager transMgr = TransferManagerBuilder.standard()
            .withS3Client(s3Client)
            .withMinimumUploadPartSize(10*1024*1024L)
            .withMultipartUploadThreshold(100*1024*1024L)
            .build();

        // TransferManager 采用异步方式进行处理，因此该调用会立即返回。
        CopyObjectRequest request = new CopyObjectRequest(sourceBucke
tName, sourceObjectKey, destBucketName, destObjectKey);
        Copy copy = transMgr.copy(request);
        CopyResult result = copy.waitForCopyResult();
        System.out.println("copy success, etag=" + result.getETag());
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

- 取消分片上传

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

```
public void abortMultipartUploads() {
    String bucket = "<your-bucket-name>";
    int sevenDays = 1000 * 60 * 60 * 24 * 7;
    Date oneWeekAgo = new Date(System.currentTimeMillis() - sevenDay
```

```
s);
    TransferManager transMgr = TransferManagerBuilder.standard()
        .withS3Client(s3Client)
        .build();
    //取消在一个星期前初始化并还未完成的分片上传
    transMgr.abortMultipartUploads(bucket, oneWeekAgo);
}
```

- 关于 Content-Type 的说明

Content-Type 用于标识文件的资源类型，比如 *image/png*，*image/jpg* 是图片类型，*video/mpeg*，*video/mp4* 是视频类型，*text/plain*，*text/html* 是文本类型，浏览器针对不同的 Content-Type 会有不同的操作，比如图片类型可以预览，视频类型可以播放，文本类型可以直接打开。*application/octet-stream* 类型会直接打开下载窗口。

在 java sdk 中，如果用户没有设置 Content-Type，会根据 PutObjectRequest 中 file 参数的后缀扩展名自动生成 Content-Type。

5.1.3. 请求参数

参数	类型	说明
localPath	String	要上传的本地文件路径
bucket	String	桶名
key	String	对象 key
contentType	String	http contentType 头
storageClass	StorageClass	配置上传对象的存储类型，包括标准类型 STANDARD、低频类型 STANDARD_IA 以及归档类型 GLACIER
metadata	ObjectMetadata	对象自定义元数据

注意：acl 在 PutObjectRequest 中设置

5.2. 分片上传-初始化分片上传任务

5.2.1. 功能说明

分片上传操作可以将超过 5GB 的大文件分割后上传，分片上传对象首先需要发起分片上传请求获取一个 upload id。

5.2.2. 代码示例

```
System.out.println("multiPartUpload");
String bucket = "<your-bucket-name>";
String key = "<your-object-key>";
InitiateMultipartUploadRequest initReq = new InitiateMultipartUploadR
equest(bucket, key);
initReq.withCannedACL(CannedAccessControlList.PublicRead);
//  initReq.setStorageClass(StorageClass.Standard); // 设置对象的存储
类型
InitiateMultipartUploadResult initRes = s3Client.initiateMultipartUp
load(initReq);
System.out.println("multiPartUpload: init success, uploadId=" + initR
es.getUploadId());
```

5.2.3. 请求参数

InitiateMultipartUploadRequest 可设置的参数如下：

参数	类型	说明	是否必要
bucket	String	桶名称	是
key	String	对象的 key	是

参数	类型	说明	是否必要
cannedAcl	CannedAccessControlList	配置上传对象的预定义的标准 ACL 信息，详细说明见 设置对象访问权限 一节	否
storageClass	StorageClass	配置上传对象的存储类型，包括标准类型 STANDARD、低频类型 STANDARD_IA 以及归档类型 GLACIER	否
accessControlList	AccessControlList	配置上传对象的详细 ACL 信息，详细说明见 设置对象访问权限 一节	是
objectMetadata	ObjectMetadata	对象的元数据信息	否
tagging	ObjectTagging	对象的标签信息	否

5.2.4. 返回结果

InitiateMultipartUploadResult 返回的属性如下:

参数	类型	说明
bucket	String	执行分片上传的桶的名称
key	String	本次分片上传对象的名称
uploadId	String	本次生成分片上传任务的 id

5.3. 分片上传-上传分片

5.3.1. 功能说明

初始化分片上传任务后，指定分片上传任务的 id 可以上传分片数据，可以将大文件分割成分片后上传，除了最后一个分片，每个分片的数据大小为 5MB~5GB，每个分片上传任务最多上传 10000 个分片。

5.3.2. 代码示例

```
String bucket = "<your-bucket-name>";
String key = "<your-object-key>";
String uploadId = "<your-upload-id>";

InputStream stream1 = new ByteArrayInputStream(new byte[5*1024*1024]);
UploadPartRequest partReq1 = new UploadPartRequest();
partReq1.setBucketName(bucket);
partReq1.setKey(key);
partReq1.setUploadId(uploadId); // 在 initiateMultipartUpload 获取
partReq1.setInputStream(stream1);
partReq1.setPartNumber(1); // 设置分片号代表这次复制是整个分片上传任务中的第几个分片，从 1 开始
partReq1.setObjectMetadata(new ObjectMetadata());
partReq1.setPartSize(stream1.available());
// rate 上传限速，单位 KB
partReq1.putCustomRequestHeader("x-amz-limit", String.format("rate=%d", 10));
UploadPartResult partRes1 = s3Client.uploadPart(partReq1);
System.out.println("multiPartUpload: uploadPart success, etag=" + partRes1.getETag());
```

```
InputStream stream2 = new ByteArrayInputStream(new byte[1*1024*1024]);
UploadPartRequest partReq2 = new UploadPartRequest();
partReq2.setBucketName(bucket);
partReq2.setKey(key);
partReq2.setUploadId(uploadId);
partReq2.setInputStream(stream2);
partReq2.setPartNumber(2);
partReq2.setPartSize(stream2.available());
// rate 上传限速，单位 KB
partReq2.putCustomRequestHeader("x-amz-limit", String.format("rate=%
d", 10));
UploadPartResult partRes2 = s3Client.uploadPart(partReq2);
System.out.println("multiPartUpload: uploadPart success, etag=" + par
tRes2.getETag());
```

5.3.3. 请求参数

UploadPartRequest 可设置的参数如下：

参数	类型	说明	是否必要
bucket	String	执行分片上传的桶的名称	是
key	String	对象的 key	是
inputStream	InputStream	对象的输入数据流	是
partNumber	int	说明当前数据在文件中所属的分片，大于等于 1，小于等于 10000	是
uploadId	String	通过 initiateMultipartUpload 操	是

参数	类型	说明	是否必要
		作获取的 UploadId，与一个分片上传的对象对应	

5.3.4. 返回结果

UploadPartRequest 返回的属性如下:

参数	类型	说明
etag	String	本次上传分片对应的 Entity Tag

5.4. 分片上传-合并分片

5.4.1. 功能说明

合并指定分片上传任务 id 对应任务中已上传的对象分片，使之成为一个完整的文件。

5.4.2. 代码示例

```
String bucket = "<your-bucket-name>";
String key = "<your-object-key>";
String uploadId = "<your-upload-id>";
PartETag tag1 = new PartETag(partReq1.getPartNumber(), partRes1.getETag()); //partNumber 与 eTag 在上传分片时获取
PartETag tag2 = new PartETag(partReq2.getPartNumber(), partRes2.getETag());
List<PartETag> partETags = new ArrayList<>();
partETags.add(tag1);
partETags.add(tag2);

CompleteMultipartUploadRequest completeReq = new CompleteMultipartUploadRequest(bucket, key, uploadId, partETags);
```

```
CompleteMultipartUploadResult completeRes = s3Client.completeMultipa
rtUpload(completeReq);
System.out.println("multiPartUpload: complete success, etag=" + compl
eteRes.getETag());
System.out.println("bucket=" + completeRes.getBucketName());
System.out.println("key=" + completeRes.getKey());
System.out.println("location=" + completeRes.getLocation());
System.out.println("versionId=" + completeRes.getVersionId());
```

5.4.3. 请求参数

CompleteMultipartUploadRequest 可设置的参数如下：

参数	类型	说明	是否必要
bucket	String	执行分片上传的桶的名称	是
key	String	对象的 key	是
partETags	List<PartETag>	包含了每个已上传的分片的 ETag 和 PartNumber 等信息	是
uploadId	String	通过 CreateMultipartUpload 操作获取的 UploadId，与一个对象的分片上传对应	是

5.4.4. 返回结果

CompleteMultipartUploadResult 返回的属性如下：

参数	类型	说明
bucketName	String	执行分片上传的桶的名称
key	String	对象的 key

参数	类型	说明
etag	String	本次上传对象后对应的 Entity Tag
location	String	合并生成对象的 URI 信息
versionId	String	上传对象后相应的版本 ID

5.5. 分片上传-列举分片上传任务

5.5.1. 功能说明

列举分片上传操作可以列出一个桶中正在进行的分片上传，这些分片上传的请求已经发起，但是还没完成或者被中止。listMultipartUploads 操作可以通过指定 maxUploads 参数来设置返回分片上传信息的数量，maxUploads 参数的最大值和默认值均为 1000。如果返回结果中的 isTruncated 字段为 true，表示还有符合条件的分片上传信息没有列出，可以通过设置请求中的 keyMarker 和 uploadIdMarker 参数，来列出符合筛选条件的正在上传的分片信息。

5.5.2. 代码示例

```
public void listMultipartUploads(){
    String bucket = "<your-bucket-name>";
    ListMultipartUploadsRequest listMultipartUploadsRequest = new ListMultipartUploadsRequest(bucket);
    MultipartUploadListing multipartUploadListing = s3Client.listMultipartUploads(listMultipartUploadsRequest);
    for (MultipartUpload multipartUpload : multipartUploadListing.getMultipartUploads()) {
        System.out.println("uploadId=" + multipartUpload.getUploadId());
        System.out.println("initiator=" + multipartUpload.getInitiato
```

```
r());
        System.out.println("initiated=" + multipartUpload.getInitiate
d());
        System.out.println("key=" + multipartUpload.getKey());
    }
}
```

如果 list 大于 1000，则返回的结果中 isTruncated 为 true，并返回 NextKeyMarker NextUploadIdMarker 作为下次读取的起点。如果没有一次性获取所有的分片上传事件，可以采用分页列举的方式。列举所有分片上传事件示例代码如下：

```
public void listMultipartUploads2(){
    String bucket = "<your-bucket-name>";
    MultipartUploadListing multipartUploadListing;
    ListMultipartUploadsRequest listMultipartUploadsRequest = new
        ListMultipartUploadsRequest(bucket);
    do {
        multipartUploadListing = s3Client.listMultipartUploads(listMu
ltipartUploadsRequest);
        for (MultipartUpload multipartUpload : multipartUploadListing.
getMultipartUploads()) {
            System.out.println("uploadId=" + multipartUpload.getUploa
dId());
            System.out.println("initiator=" + multipartUpload.getInit
iator());
            System.out.println("initiated=" + multipartUpload.getInit
iated());
            System.out.println("key=" + multipartUpload.getKey());
        }
```

```
        listMultipartUploadsRequest.setKeyMarker(multipartUploadListi
ng.getNextKeyMarker());
        listMultipartUploadsRequest.setUploadIdMarker(multipartUpload
Listing.getNextUploadIdMarker());
    } while (multipartUploadListing.isTruncated());
}
```

5.5.3. 请求参数

ListMultipartUploadsRequest 可设置的参数如下：

参数	类型	说明	是否必要
bucket	String	执行本操作的桶名称	是
delimiter	String	与 Prefix 参数一起用于对对象 key 进行分组的字符。所有 key 包含指定的 Prefix 且第一次出现 Delimiter 字符之间的对象作为一组。如果没有指定 Prefix 参数，按 Delimiter 对所有对象 key 进行分割，多个对象分割后从对象 key 开始到第一个 Delimiter 之间相同的部分形成一组	否
encodingType	String	用于设置 response 中 object key 的字符编码类型	否
keyMarker	String	和 uploadIdMarker 参数一起用于指定返回哪部分分片上传的信息。如果没有设置 uploadIdMarker 参数，则只返回对象 key 按照字典顺序排序后位于	否

参数	类型	说明	是否必要
		keyMarker 标识符之后的分片信息。如果设置了 uploadIdMarker 参数，则会返回对象 key 等于 keyMarker 且 uploadId 大于 uploadIdMarker 的分片信息	
maxUploads	int	用于指定相应消息体中正在进行的分片上传信息的最大数量，最小值为 1，默认值和最大值都是 1000	否
prefix	String	与 delimiter 参数一起用于对对象 key 进行分组的字符。所有 key 包含指定的 Prefix 且第一次出现 delimiter 字符之间的对象作为一组	否
uploadIdMarker	String	和 keyMarker 参数一起用于指定返回哪部分分片上传的信息，仅当设置了 keyMarker 参数的时候有效。设置后返回对象 key 等于 keyMarker 且 uploadId 大于 uploadIdMarker 的分片信息	否

5.5.4. 返回结果

MultipartUploadListing 返回的属性如下：

参数	类型	说明
bucketName	String	执行本操作的桶名称

参数	类型	说明
commonPrefixes	List<String>	当请求中设置了 delimiter 和 prefix 属性时，所有包含指定的 prefix 且第一次出现 delimiter 字符的对象 key 作为一组
delimiter	String	与请求中设置的 delimiter 一致
isTruncated	boolean	当为 false 时表示返回结果中包含了全部符合本次请求查询条件的上传分片信息，否则只返回了数量为 maxUploads 个的分片信息
keyMarker	String	返回上传分片列表中的起始对象的 key
maxUploads	int	本次返回结果中包含的上传分片数量的最大值
nextKeyMarker	String	当 isTruncated 为 true 时，nextKeyMarker 可以作为后续查询已初始化的上传分片请求中的 keyMarker 的值
nextUploadIdMarker	String	当 isTruncated 为 true 时，nextKeyMarker 可以作为后续查询已初始化的上传分片请求中的 uploadIdMarker 的值
prefix	String	限定返回分片中对应对象的 key 必须以 prefix 作为前缀
uploadIdMarker	String	返回上传分片列表中的起始 uploadId

参数	类型	说明
uploads	List<MultipartUpload>	包含了零个或多个已初始化的上传分片信息的数组。数组中的每一项包含了分片初始化时间、分片上传操作发起者、对象 key、对象拥有者、存储类型和 uploadId 等息

5.6. 分片上传-列举已上传的分片

5.6.1. 功能说明

列举已上传分片操作可以列出一个分片上传操作中已经上传完毕但是还未合并的分片信息。请求中需要提供 object key 和 upload id，返回的结果最多包含 1000 个已上传的分片信息，默认返回 1000 个，可以通过设置 maxParts 参数的值指定返回结果中分片信息的数量。如果已上传的分片信息的数量多于 1000 个，则返回结果中的 isTruncated 字段为 true，可用通过设置 partNumberMarker 参数获取 partNumber 大于该参数的分片信息。

5.6.2. 代码示例

```
public void listParts() {
    System.out.println("ListParts");
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";
    String uploadId = "<your-upload-id>";
    ListPartsRequest listParts = new ListPartsRequest(bucket, key, up
loadId);
    PartListing partListing = this.s3Client.listParts(listParts);
    System.out.println("bukcet=" + partListing.getBucketName() + ", k
ey=" + partListing.getKey() + ", uploadId="+partListing.getUploadId
());
```

```
    for (PartSummary part : partListing.getParts()) {
        System.out.println("part number="+part.getPartNumber()+", siz
e="+part.getSize());
    }
}
```

如果分片数大于 1000，返回的 PartListing 中 isTruncated 为 true ，并可以根据 NextPartNumberMarker 为下一次请求的 list 的起始位置。

```
public void listParts2() {
    System.out.println("ListParts");
    String bucket = "<your-bucket-name>";
    String key = "<your-object-key>";
    String uploadId = "<your-upload-id>";
    ListPartsRequest listParts = new ListPartsRequest(bucket, key, up
loadId);
    PartListing partListing;
    ListPartsRequest listPartsRequest = new ListPartsRequest(bucket,
key, uploadId);
    do {
            partListing = s3Client.listParts(listPartsRequest);
            System.out.println("bukcet=" + partListing.getBucketName()
 + ", key=" + partListing.getKey() + ", uploadId="+partListing.getUpl
oadId());
            for (PartSummary part : partListing.getParts()) {
                System.out.println("part number="+part.getPartNumber()
+", size="+part.getSize());
            }
            listPartsRequest.setPartNumberMarker(partListing.getNextP
```

```
artNumberMarker());
        } while (partListing.isTruncated());
}
```

5.6.3. 请求参数

ListPartsRequest 可设置的参数如下：

参数	类型	说明	是否必要
bucket	String	执行本操作的桶名称	是
key	String	对象的 key	是
maxParts	int	指定返回分片信息的数量，默认值和最大值均为 1000	否
partNumberMarker	int	用于指定返回 part number 大于 partNumberMarker 的分片信息	否
uploadId	String	指定返回该 id 所属的分片上传的分片信息	是

5.6.4. 返回结果

PartListing 返回的属性如下：

参数	类型	说明
bucketName	String	执行本操作的桶名称
key	String	本次分片上传对象的名称
isTruncated	boolean	当为 false 时表示返回结果中包含了全部符合本次请求查询条件的上传分片信息，否则只返回了数量为 MaxParts 个

参数	类型	说明
		的分片信息
maxParts	int	本次返回结果中包含的上传分片数量的最大值
nextPartNumberMarker	int	当 IsTruncated 为 true 时，NextPartNumberMarker 可以作为后续查询已上传分片请求中的 PartNumberMarker 的值
parts	List<PartSummary>	包含了已上传分片信息的数组，数组中的每一项包含了该分片的 Entity tag、最后修改时间、PartNumber 和大小等信息
uploadId	String	本次分片上传操作 Id

5.7. 分片上传-复制分片

5.7.1. 功能说明

复制分片操作可以从一个已存在的对象中复制指定分片的数据。您可以使用 copyPart 复制分片。在复制分片前，需要使用 initiateMultipartUpload 接口获取一个 upload id，在完成复制和上传分片操作之后，需要使用 completeMultipartUpload 操作组装分片成为一个对象。当复制的对象大小超过 5GB，必须使用复制分片操作完成对象的复制。除了最后一个分片外，每个复制分片的大小范围是[5MB，5GB]。

5.7.2. 代码示例

```
String destBucketName = "<your-bucket-name>";
String destObjectKey = "<your-object-key>";
String sourceBucketName = "<source-bucket-name>";
String sourceObjectKey = "<source-object-key>";
```

```
List<PartEtag> partEtags = new ArrayList<PartEtag>();
// 创建分片复制请求
CopyPartRequest copyRequest = new CopyPartRequest()
    //设置源桶和对象，目标桶和对象
    .withSourceBucketName(sourceBucketName)
    .withSourceKey(sourceObjectKey)
    .withDestinationBucketName(destBucketName)
    .withDestinationKey(destObjectKey)
    //uploadId 为 initiateMultipartUpload 中返回值
    .withUploadId(initResult.getUploadId())
    //设置分片复制范围
    .withFirstByte(firstByte)
    .withLastByte(lastByte)
    //设置分片号，代表这次复制是整个分片上传任务中的第几个分片
    .withPartNumber(partNum);

CopyPartResult copyPartResult = s3.copyPart(copyRequest);
partETags.add(copyPartResult.getPartETag()); //把 partETag 放入 PartE
tag 列表中，合并分片是需要此参数
System.out.println("multiPartCopy: copyPart success, part " + partNum
 + ", etag=" + copyPartResult.getETag());
System.out.println("lastModifiedDate=" + copyPartResult.getLastModif
iedDate());
System.out.println("partNumber=" + copyPartResult.getPartNumber());
```

5.7.3. 请求参数

copyRequest 可设置的参数如下：

参数	类型	说明	是否必要
sourceBucketName	String	源桶名称	是
sourceKey	String	源对象 key	是
destinationBucketName	String	目标桶名称	是
destinationKey	String	目标对象 key	是
matchingETagConstraints	List<String>	用于指定只有在源对象的 eTag 和该参数值匹配的情况下才进行复制操作。	否
modifiedSinceConstraint	Date	用于只有当源对象在指定时间后被修改的情况下才进行复制操作	否
nonmatchingEtagConstraints	List<String>	用于指定只有在源对象的 eTag 和该参数值不匹配的情况下才进行复制操作。	否
unmodifiedSinceConstraint	Date	用于仅当源自指定时间以来未被修改的情况下才进行复制操作	否
firstByte	long	指定本次分片复制的数据范围，源对象的起始字节	是
lastByte	long	指定本次分片复制的数据范围，源对象的结束字节	是
partNumber	int	说明本次分片复	是

参数	类型	说明	是否必要
		制的数据在原对象中所属的部分	
uploadId	String	与本次复制操作相应的分片上传 Id	是

5.7.4. 返回结果

CopyPartResult 返回的属性如下:

参数	类型	说明
etag	String	包含复制分片的 Entity Tag
lastModifiedDate	Date	复制分片的最新修改时间
partNumber	String	分片序号

5.8. 分片上传-取消分片上传任务

5.8.1. 功能说明

取消分片上传任务操作用于终止一个分片上传。当一个分片上传被中止后，不会再有数据通过与之相应的 upload id 上传，同时已经被上传的分片所占用的空间会被释放。执行取消分片上传任务操作后，正在上传的分片可能会上传成功也可能会被中止，所以必要的情况下需要执行多次取消分片上传任务操作去释放全部上传成功的分片所占用的空间。可以通过执行列举已上传分片操作来确认所有中止分片上传后所有已上传分片的空间是否被被释放。

5.8.2. 代码示例

```
String bucket = "<your-bucket-name>";
String key = "<your-object-key>";
String uploadId = "<your-upload-id>";
System.out.println("multiPartUpload: error=" + e.getMessage());
```

```
AbortMultipartUploadRequest abortReq = new AbortMultipartUploadReques
t(bucket, key, uploadId);
s3Client.abortMultipartUpload(abortReq);
```

5.8.3. 请求参数

AbortMultipartUploadRequest 可设置的参数如下:

参数	类型	说明	是否必要
bucket	String	执行本操作的桶名称	是
key	String	分片上传的对象的key	是
uploadId	String	指定需要终止的分片上传的 id	是

6. 安全凭证服务(STS)

STS 即 Secure Token Service 是一种安全凭证服务，可以使用 STS 来完成对于临时用户的访问授权。对于跨用户短期访问对象存储资源时，可以使用 STS 服务。这样就不需要透露主账号 AK/SK，只需要生成一个短期访问凭证给需要的用户使用即可，避免主账号 AK/SK 泄露带来的安全风险。

6.1. 初始化 STS 服务

```
String accessKey = "<your-access-key>";
String secretKey = "<your-secret-access-key>";
String endPoint = "<your-endpoint>";
BasicAWSCredentials credentials = new BasicAWSCredentials(accessKey,
secretKey);
        AwsClientBuilder.EndpointConfiguration endpointConfiguration
=
                new AwsClientBuilder.EndpointConfiguration(endPoint, R
egions.DEFAULT_REGION.getName());
        return AWSSecurityTokenServiceClientBuilder.standard()
                .withCredentials(new AWSStaticCredentialsProvider(cred
entials))
                .withEndpointConfiguration(endpointConfiguration)
                .build();
```

6.2. 获取临时 token

```
private static final String DEFAULT_BUCKET = "<your-bucket-name>";
private static final String ROLE_SESSION_NAME = "<your-session-name>";
private static final String ARN = "arn:aws:iam:::role/xxxxxx";
private static final String POLICY = "{\"Version\":\"2012-10-17\"," +
```

```
 "\"Statement\":" + "{\"Effect\":\"Allow\","
        + "\"Action\":[\"s3:*\"]," // 允许进行 S3 的所有操作。如果仅需要上传，这里可以设置为 PutObject
        + "\"Resource\":[\"arn:aws:s3:::" + DEFAULT_BUCKET + "\",\"arn:aws:s3:::" + DEFAULT_BUCKET + "/*\"]"// 允许操作默认桶中的所有文件，可以修改此处来保证操作的文件
            + "}}";

public static void assumeRole() {
    try {
        AWSSecurityTokenService stsClient = buildSTSClient();
        AssumeRoleRequest assumeRoleRequest = new AssumeRoleRequest();
        assumeRoleRequest.setRoleArn(ARN);
        assumeRoleRequest.setPolicy(POLICY);
        assumeRoleRequest.setRoleSessionName(ROLE_SESSION_NAME);
        assumeRoleRequest.setDurationSeconds(60*60*2);      // 单位秒，有效时间，默认 1 小时，最长 12 小时

        System.out.println("policy=" + POLICY);
        AssumeRoleResult assumeRoleRes = stsClient.assumeRole(assumeRoleRequest);
        Credentials stsCredentials = assumeRoleRes.getCredentials();
        System.out.println("ak=" + stsCredentials.getAccessKeyId());
        System.out.println("sk=" + stsCredentials.getSecretAccessKey());
        System.out.println("token=" + stsCredentials.getSessionToken());
    } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
```

6.3. Policy 设置例子

允许所有的操作

```
{"Version":"2012-10-17","Statement":{"Effect":"Allow","Action":["s3:*"],"Resource":["arn:aws:s3:::<your-bucket-name>","arn:aws:s3:::<your-bucket-name>/*"]}}
```

限制只能上传和下载

```
{"Version":"2012-10-17","Statement":{"Effect":"Allow","Action":["s3:PutObject","s3:GetObject"],"Resource":["arn:aws:s3:::<your-bucket-name>","arn:aws:s3:::<your-bucket-name>/*"]}}
```

使用分片上传

```
{"Version":"2012-10-17","Statement":{"Effect":"Allow","Action":["s3:PutObject","s3:AbortMultipartUpload","s3:ListBucketMultipartUploads","s3:ListMultipartUploadParts"],"Resource":["arn:aws:s3:::<your-bucket-name>","arn:aws:s3:::<your-bucket-name>/*"]}}
```

其他操作权限

上传权限：s3:PutObject

下载权限：s3:GetObject

删除权限：s3:DeleteObject

获取列表权限：s3:ListBucket

注意：

```
1.ListObjects 操作是由 ListBucket 权限控制的
2."Version:2012-10-17"是系统的 policy 格式的版本号，不能改成其他日期

更多操作权限可以参考:
https://www.ctyun.cn/document/10306929/10136179
```

参数	类型	描述	是否必要
RoleArn	String	角色的 ARN，在控制台创建角色后可以查看	是
Policy	String	角色的 policy，需要是 json 格式，限制长度 1~2048	是
RoleSessionName	String	角色会话名称，此字段为用户自定义，限制长度 2~64	是
DurationSeconds	Integer	会话有效期时间，默认为 3600s，范围 15 分钟至 12 小时	否

6.4. 使用临时 token

实现一个 CredentialsProvider，支持更新 ak/sk 和 token。

```
public class MyCredentialsProvider implements AWSCredentialsProvider
{
    private AWSCredentials credentials;

    public MyCredentialsProvider(String ak, String sk, String token)
{
        this.credentials = new BasicSessionCredentials(ak, sk, token);
```

```
    }

    public synchronized AWSCredentials getCredentials() {
        return credentials;
    }

    public synchronized void refresh() {
    }

    // 更新 ak,sk,token
    public synchronized void updateCred(String ak, String sk, String
token) {
        this.credentials = new BasicSessionCredentials(ak, sk, token);
    }
}
```

使用临时 token

```
String secretKey = "<your-secret-access-key>";
String endPoint = "<your-endpoint>";
String sessionToken = "<your-session-token>";
MyCredentialsProvider credProvider = new MyCredentialsProvider(access
Key, secretKey, sessionToken);
ClientConfiguration clientConfiguration = new ClientConfiguration();
AwsClientBuilder.EndpointConfiguration endpointConfiguration = new Aw
sClientBuilder.EndpointConfiguration(
        endPoint, Regions.DEFAULT_REGION.getName());
return AmazonS3ClientBuilder.standard()
        .withCredentials(credProvider)
```

```
        .withClientConfiguration(clientConfiguration)
        .withEndpointConfiguration(endpointConfiguration)
        .build();
```