ZOS对象存储Java_SDK使用手册

环境配置

安装SDK

前置环境要求

- jdk1.8及之后版本

下载SDK

- SDK压缩包快速下载地址: 见帮助中心

导入

Windows

- 以IntelliJ IDEA 为例，把jar包放到工程目录下，右键选择"Add as Library"即可。
- 如果不用IDE，则通过修改系统环境变量中过的CLASSPATH，添加本地jar包的路径即可。

Linux

- 配置环境变量CLASSPATH：

```
# <your-path>换成你的sdk保存路径即可
export CLASSPATH=.:<your-path>/*:$CLASSPATH
```

maven项目引入

- maven仓库引入（暂不支持）
- 本地引入

```
# 在pom.xml文件的dependencies中添加以下依赖

<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>zos-java-sdk-s3</artifactId>
        <version>1.11.998</version>
        <scope>system</scope>
        <systemPath>/path/to/zos-java-sdk-s3.jar</systemPath>
    </dependency>

    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>zos-java-sdk-sts</artifactId>
        <version>1.11.998</version>
        <scope>system</scope>
        <systemPath>/path/to/zos-java-sdk-sts.jar</systemPath>
    </dependency>

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

```
        <artifactId>aws-java-sdk-iam</artifactId>
        <version>1.11.998</version>
        <scope>system</scope>
        <systemPath>/path/to/aws-java-sdk-iam-1.11.998.jar</systemPath>
    </dependency>

    <dependency>
        <groupId>org.bouncycastle</groupId>
        <artifactId>bcprov-ext-jdk18on</artifactId>
        <version>1.77</version>
        <scope>system</scope>
        <systemPath>/path/to/bcprov-ext-jdk18on-177.jar</systemPath>
    </dependency>

    <dependency>
        <groupId>org.bouncycastle</groupId>
        <artifactId>bcprov-jdk18on</artifactId>
        <version>1.77</version>
        <scope>system</scope>
        <systemPath>/path/to/bcprov-jdk18on-177.jar</systemPath>
    </dependency>
</dependencies>
```

编译运行

Windows

- IDE中运行
- 如果不使用IDE，则在命令行输入以下命令：

```
javac xxx.java
java xxx
```

Linux

```
javac xxx.java
java xxx
```

创建客户端

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;

public class Main {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";

    public static void main(String[] args) {
```

```
        AmazonS3 s3Client = null;
        try {
            // 当使用HTTPS协议且采用自签名认证时，需关闭证书检查
            // System.setProperty("com.amazonaws.sdk.disableCertChecking",
"true");
            // 使用凭据和配置建立连接
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

参数说明

- ClientConfiguration

设定参数方法	是否必选	描述
void setSignerOverride(final String value)	可选	设置请求签名格式： "S3SignerType"，V2签名； "AWSS3V4SignerType"，V4签名。
void setProtocol(Protocol protocol)	可选	选择协议： Protocol.HTTP，HTTP协议； Protocol.HTTPS，HTTPS协议。
void setMaxConnections(int maxConnections)	可选	设置客户端最大连接数
void setRequestTimeout(int requestTimeout)	可选	设置请求超时时间

- AwsClientBuilder

设定参数方法	是否必选	描述
Subclass withCredentials(AWSCredentialsProvider credentialsProvider)	是	设置认证材料（AK、SK）
Subclass withClientConfiguration(ClientConfiguration config)	可选	设置客户端配置
Subclass withEndpointConfiguration(EndpointConfiguration endpointConfiguration)	是	设置终端节点
Subclass disableChunkedEncoding()	可选	禁止分块编码（建议开启）
Subclass enablePathStyleAccess()	可选	启用路径样式访问（建议开启）

桶操作

创建桶

功能说明

创建桶请求可以在指定账号下创建一个新的桶。**通过SDK创建的桶，如果要在控制台使用，则需手动添加跨域访问配置，允许`https://console.ctyun.cn`域名访问，具体参加设置跨域访问配置。**

方法原型

```
Bucket createBucket(CreateBucketRequest createBucketRequest)
```

参数说明

- CreateBucketRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名，长度范围为3到63个字符，支持小写字母、数字、中划线(-)，禁止以中划线(-)开头或结尾。
void setCannedAcl(CannedAccessControlList cannedAcl)	可选	设置桶CannedAccessControlList ，可选项： CannedAccessControlList.Private，私有（默认）； CannedAccessControlList.PublicRead，公共读，但只有桶所有者可以写入和更改ACL； CannedAccessControlList.BucketOwnerRead，对象所有者和存储桶所有者可以读取该对象，只有对象所有者可以写入和更改ACL； CannedAccessControlList.BucketOwnerFullControl，对象所有者和存储桶所有者可以读取和写入该对象以及更改ACL。
void setAccessControlList(AccessControlList accessControlList)	可选	设置桶AccessControlList。
void setObjectLockEnabledForBucket(boolean objectLockEnabled)	可选	设置是否开启对象锁，开启对象锁会自动开启版本控制且不可暂停。只有在桶创建时可以开启对象锁，后续不可。
void setAZPolicy(String policy)	可选	设置桶冗余策略，可选项： "SINGLE-AZ"，单可用区； "MULTI-AZ"，多可用区。
void setStorageClass(StorageClass storageClass)	可选	设置桶存储类型，可选项： StorageClass.Standard，标准存储； StorageClass.StandardInfrequentAccess，低频存储； StorageClass.Glacier，归档存储。

- AccessControlList

设定参数方法	是否必选	描述
void setOwner(Owner owner)	是	设置桶所有者
void grantPermission(Grantee grantee, Permission permission)	是	设置授权角色及权限： Permission.FullControl，所有权限； Permission.Read，读权限； Permission.ReadACP，读ACL权限； Permission.Write，写权限； Permission.WriteAcp，写ACL权限。

- Owner

设定参数方法	是否必选	描述
void setId(String id)	是	设置ID
void setDisplayName(String name)	否	设置名字

- Grantee，接口类，两种实现：CanonicalGrantee、EmailAddressGrantee

设定参数方法	是否必选	描述
CanonicalGrantee(String identifier)	可选	构造函数，设置被授权用户id
EmailAddressGrantee(String emailAddress)	可选	构造函数，设置被授权用户邮箱

返回结果说明

- Bucket

获取结果方法	描述
String getName()	获取桶名

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.Bucket;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.CreateBucketRequest;

public class createBucket {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
```

```
            //创建Bucket
            CreateBucketRequest var = new CreateBucketRequest(Bucket);
            Bucket bucket = s3Client.createBucket(var);
            System.out.println(bucket.getName());
            System.out.println("=====request success=====\n");
        }catch (Exception e) {
            System.out.println("=====request fail=====\n");
            System.out.println(e.getMessage());
        }
    }
}
```

删除桶

功能说明

删除桶请求可以在指定账号下删除指定桶，删除之前要求桶为空。

方法原型

`void deleteBucket(DeleteBucketRequest deleteBucketRequest )`

参数说明

- DeleteBucketRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.DeleteBucketRequest;

public class deleteBucket {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
```

```
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            //删除Bucket
            DeleteBucketRequest var = new DeleteBucketRequest("");
            var.setBucketName(Bucket);
            s3Client.deleteBucket(var);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

判断桶是否存在

功能说明

判断某个Bucket是否存在或者是否有权限访问该Bucket(其他用户名下Bucket)。

方法原型

`HeadBucketResult headBucket(HeadBucketRequest headBucketRequest )`

参数说明

- HeadBucketRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名

返回结果说明

- HeadBucketResult

获取结果方法	描述
String getBucketRegion()	获取桶所在的region

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.Bucket;
import com.amazonaws.services.s3.model.HeadBucketRequest;
import com.amazonaws.services.s3.model.HeadBucketResult;

public class headBucket {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");


            HeadBucketRequest var = new HeadBucketRequest("");
            var.setBucketName(Bucket);
            HeadBucketResult result = s3Client.headBucket(var);
            System.out.print(Bucket + " is exist" + "\n");
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

列出所有桶

功能说明

可以列出该用户名下所有的桶列表。

方法原型

`List<Bucket> listBuckets()`

参数说明

无

返回结果说明

- Bucket

获取结果方法	描述
Owner getOwner()	获取桶owner
Date getCreationDate()	获取桶创建时间
String getName()	获取桶名

- Owner

获取结果方法	描述
String getDisplayName()	获取名字
String getId()	获取Id

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.Bucket;

import java.util.List;

public class listBuckets {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
```

```
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            //获取Bucket列表
            List<Bucket> buckets  = s3Client.listBuckets();
            for(int i = 0; i < buckets.size();i++){
                System.out.print(buckets.get(i).getOwner().toString() + "\n");
                System.out.print(buckets.get(i).getName()+ "\n");
                System.out.print(buckets.get(i).getCreationDate() + "\n");
            }
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

设置桶策略

功能说明

请求用于为某个Bucket设置桶策略。

方法原型

`void setBucketPolicy(SetBucketPolicyRequest setBucketPolicyRequest)`

参数说明

- SetBucketPolicyRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名
void setPolicyText(String policyText)	是	设置桶策略，通过String表示json格式的桶策略

- json格式的桶策略字段描述

字段	描述	是否必须
Version	保持与AmazonS3一致，当前支持"2012-10-17"	否
Id	桶策略ID，桶策略的唯一标识	否
Statement	桶策略描述，定义完整的权限控制。每条桶策略的Statement可由多条描述组成，每条描述是一个dict，每条描述可包含以下字段："Sid", "Effect", "Principal", "NotPrincipal", "Action", "NotAction", "Resource", "NotResource", "Condition"。	是
Sid	本条桶策略描述的ID	否
Effect	桶策略的效果，即指定本条桶策略描述的权限是接受请求还是拒绝请求。接受请求：配置为“Allow”，拒绝请求：配置为"Deny"	是
Principal	被授权人，即指定本条桶策略描述所作用的用户，支持通配符“*”，表示所有用户。当对某个user进行授权时，Principal格式为"AWS":"arn:aws:s3:::user/userId"	否，与NotPrincipal只能选一个
NotPrincipal	不被授权人，即指定本条桶策略描述所**排除**的用户，支持通配符“*”，表示所有用户。当对某个user进行授权时，Principal格式为"AWS":"arn:aws:s3:::user/userId"	否，与Principal只能选一个
Action	指定本条桶策略描述所作用的ZOS操作。以列表形式表示，可配置多条操作，以逗号间隔。支持通配符“*“，表示该资源能进行的所有操作。常用的Action有"s3:GetObject"，"s3:GetObjectAcl"，"s3:PutObject"，"s3:PutObjectAcl"等	否，与NotAction只能选一个
NotAction	指定本条桶策略描述所**排除**的ZOS操作。以列表形式表示，可配置多条操作，以逗号间隔。支持通配符“*“，表示该资源能进行的所有操作。常用的Action有"s3:GetObject"，"s3:GetObjectAcl"，"s3:PutObject"，"s3:PutObjectAcl"等	否，与Action只能选一个
Resource	桶策略作用的资源，"arn:aws:s3:::bucket_name"用于指定桶级的策略，适用于对整个桶的操作；"arn:aws:s3:::bucket_name/*"用于指定桶中所有对象的策略，适用于影响桶中所有对象的操作。	是，与NotResource只能选一个
NotResource	桶策略**排除**的资源，"arn:aws:s3:::bucket_name"用于指定桶级的策略，适用于对整个桶的操作；"arn:aws:s3:::bucket_name/*"用于指定桶中所有对象的策略，适用于影响桶中所有对象的操作。	是，与Resource只能选一个
Condition	条件语句，指定本条桶策略所限制的条件。可以通过Condition对ZOS资源设置防盗链，形如："Condition":{"StringEquals":{"aws:Referer":["`www.ctyun.cn`"]}，此时如果Effect为“Allow”，则允许来自"`www.ctyun.cn`"的请求；如果为“Deny”，则拒绝。	否

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

public class setBucketPolicy {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
```

```
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            // 允许所有人对bucket进行getobject操作
            String policy =
                    "{" +
                    "   \"Version\": \"2012-10-17\"," +
                    "   \"Statement\": [" +
                    "       {" +
                    "           \"Sid\": \"policy1\"," +
                    "           \"Effect\": \"Allow\"," +
                    "           \"Principal\": \"*\"," +
                    "           \"Action\": [" +
                    "               \"s3:GetObject\"" +
                    "           ]," +
                    "           \"Resource\": [" +
                    "               \"arn:aws:s3:::"+Bucket+"/*\"" +
                    "           ]" +
                    "       }" +
                    "   ]" +
                    "}";
            //创建Bucket
            SetBucketPolicyRequest var = new SetBucketPolicyRequest();
            var.setBucketName(Bucket);
            var.setPolicyText(policy);
            s3Client.setBucketPolicy(var);
            System.out.println("=====request success=====\n");
        }catch (Exception e) {
            System.out.println("=====request fail=====\n");
            System.out.println(e.getMessage());
        }
    }
}
```

获取桶策略

功能说明

请求用于获取某个Bucket的桶策略。

方法原型

`BucketPolicy getBucketPolicy(String bucketName)`

参数说明

- bucketName，桶名

返回结果说明

- BucketPolicy

获取结果方法	描述
String getPolicyText()	获取policy

代码示例

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.model.BucketPolicy;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;

public class getBucketPolicy {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
```

```
            System.out.print("=====connect success=====\n");

            BucketPolicy bucket_policy = s3Client.getBucketPolicy(Bucket);
            String policy = bucket_policy.getPolicyText();
            System.out.println("Policy: " + policy);

        }catch (Exception e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

删除桶策略

功能说明

请求用于删除某个Bucket的桶策略。

方法原型

`void deleteBucketPolicy(String bucketName)`

参数说明

- bucketName，桶名

返回结果说明

无

代码示例

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;

public class deleteBucketPolicy {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);
```

```
            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            s3Client.deleteBucketPolicy(Bucket);
            System.out.print("=====request success=====\n");
        } catch (Exception e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

设置桶ACL

功能说明

设置Bucket的ACL，控制对Bucket的访问权限。该操作需要用户具有WRITE_ACP权限。

设置方法有三种：

1. CannedAccessControlList，对整个桶的权限设置；
2. AccessControlList ，根据给出的配置，具体的给某些用户授予某些权限；

方法原型

`void setBucketAcl(SetBucketAclRequest setBucketAclRequest )`

参数说明

- SetBucketAclRequest

设定参数方法	是否必选	描述
SetBucketAclRequest(String bucketName, CannedAccessControlList acl)	可选	构造函数，输入桶名和CannedAccessControlList：CannedAccessControlList.Private，私有（默认）；CannedAccessControlList.PublicRead，公共读，但只有桶所有者可以写入和更改ACL。
SetBucketAclRequest(String bucketName, AccessControlList acl)	可选	构造函数，输入桶名和AccessControlList

- AccessControlList

设定参数方法	是否必选	描述
void setOwner(Owner owner)	是	设置桶所有者
void grantPermission(Grantee grantee, Permission permission)	是	设置授权角色及权限： Permission.FullControl，所有权限； Permission.Read，读权限； Permission.ReadACP，读ACL权限； Permission.Write，写权限； Permission.WriteAcp，写ACL权限。

- Owner

设定参数方法	是否必选	描述
Owner(String id, String displayName)	可选	构造函数设置ID和名字
void setId(String id)	是	设置ID
void setDisplayName(String name)	否	设置名字

- Grantee，接口类，两种实现：CanonicalGrantee、EmailAddressGrantee

设定参数方法	是否必选	描述
CanonicalGrantee(String identifier)	可选	构造函数，设置被授权用户id
EmailAddressGrantee(String emailAddress)	可选	构造函数，设置被授权用户邮箱

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import java.util.List;

public class setBucketAcl {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
```

```
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");


            // 第一种方法
            SetBucketAclRequest req = new SetBucketAclRequest(Bucket,
CannedAccessControlList.Private);
            // 第二种方法
//            AccessControlList acl = new AccessControlList();
//            Owner owner = new Owner("test-1", "test-1");
//            acl.setOwner(owner);
//            CanonicalGrantee canonicalGrantee = new CanonicalGrantee("test-
2");
//            acl.grantPermission(canonicalGrantee, Permission.Write);
//            SetBucketAclRequest req = new SetBucketAclRequest(Bucket, acl);

            s3Client.setBucketAcl(req);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取桶ACL

功能说明

获取桶ACL接口用来获取 Bucket 的 ACL， 即存储桶（Bucket）的访问权限控制列表。该操作需要READ_ACP权限。

方法原型

`AccessControlList getBucketAcl(GetBucketAclRequest getBucketAclRequest)`

参数说明

- GetBucketAclRequest

设定参数方法	是否必选	描述
GetBucketAclRequest(String bucketName)	是	构造函数设置桶名

返回结果说明

- AccessControlList

获取结果方法	描述
Owner getOwner()	获取桶Owner
List<Grant> getGrantsAsList()	获取授权列表

- Owner

设定参数方法	描述
String getId()	获取ID
String getDisplayName()	获取名字

- Grant

获取结果方法	描述
Grantee getGrantee()	获取授权用户信息
Permission getPermission()	获取权限类型: Permission.FullControl，所有权限； Permission.Read，读权限； Permission.ReadACP，读ACL权限； Permission.Write，写权限； Permission.WriteAcp，写ACL权限。

- Grantee

获取结果方法	描述
String getTypeIdentifier()	获取用户类型： id，id用户； emailAddress，邮箱用户； uri，组用户
String getIdentifier();	获取用户标识

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import java.util.List;

public class getBucketAcl {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            GetBucketAclRequest req = new GetBucketAclRequest(Bucket);
            AccessControlList accessControlList = s3Client.getBucketAcl(req);

            for (Grant grant:accessControlList.getGrantsAsList()) {
                System.out.println(grant.toString());
            }
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

设置桶生命周期配置

功能说明

设置桶的生命周期规则。

方法原型

void setBucketLifecycleConfiguration(SetBucketLifecycleConfigurationRequest setBucketLifecycleConfigurationRequest)

参数说明

- SetBucketLifecycleConfigurationRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名
void setLifecycleConfiguration(BucketLifecycleConfiguration lifecycleConfiguration)	是	设置生命周期配置

- BucketLifecycleConfiguration

设定参数方法	是否必选	描述
void setRules(List<Rule> rules)	可选	设置规则列表

- Rule

设定参数方法	是否必选	描述
void setStatus(String status)	是	设置规则是否启用："Enabled"，启用；"Disabled"，不启用。
void setId(String id)	可选	设置规则ID
void setFilter(LifecycleFilter filter)	是	设置对象过滤条件
void setTransitions(List<Transition> transitions)	可选	设置对象到期转存规则
Rule addTransition(Transition transition)	可选	添加一条对象到期转存规则
void setExpirationInDays(int expirationInDays)	可选	设置对象到期删除天数
void setExpirationDate(Date expirationDate)	可选	设置对象到期删除时间
void setNoncurrentVersionTransitions(List<NoncurrentVersionTransition> noncurrentVersionTransitions)	可选	设置对象历史版本到期转存规则
Rule addNoncurrentVersionTransition(NoncurrentVersionTransition noncurrentVersionTransition)	可选	添加一条对象历史版本到期转存规则
void setNoncurrentVersionExpirationInDays(int value)	可选	设置对象历史版本到期删除天数
void setExpiredObjectDeleteMarker(boolean expiredObjectDeleteMarker)	可选	设置是否删除“删除标记”

- LifecycleFilter

设定参数方法	是否必选	描述
void setPredicate(LifecycleFilterPredicate predicate)	是	设置对象过滤条件， LifecycleFilterPredicate 子类： 1. LifecyclePrefixPredicate； 2. LifecycleTagPredicate。

- LifecyclePrefixPredicate

设定参数方法	是否必选	描述
LifecyclePrefixPredicate(String prefix)	是	构造函数设置对象前缀过滤条件

- LifecycleTagPredicate

设定参数方法	是否必选	描述
LifecycleTagPredicate(Tag tag)	是	构造函数设置对象标签过滤条件

- Tag

设定参数方法	是否必选	描述
Tag(String key, String value)	是	构造函数设置key和value

- Transition

设定参数方法	是否必选	描述
void setStorageClass(StorageClass storageClass)	是	设置转存的存储类型，可选项： StorageClass.Standard，标准存储； StorageClass.StandardInfrequentAccess，低频存储； StorageClass.Glacier，归档存储。
void setDate(Date expirationDate)	可选	设置固定时间点转存
void setDays(int expirationInDays)	可选	设置固定天数后转存
void setIsAccessTime(Boolean IsAccessTime)	可选	设置是否采用基于最后一次访问时间的生命周期规则

- NoncurrentVersionTransition

设定参数方法	是否必选	描述
void setStorageClass(StorageClass storageClass)	是	设置转存的存储类型，可选项： StorageClass.Standard，标准存储； StorageClass.StandardInfrequentAccess，低频存储； StorageClass.Glacier，归档存储。
void setDays(int expirationInDays)	可选	设置固定天数后转存
void setIsAccessTime(Boolean IsAccessTime)	可选	设置是否采用基于最后一次访问时间的生命周期规则

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
```

```
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.SetBucketLifecycleConfigurationRequest;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration.Rule;
import com.amazonaws.services.s3.model.lifecycle.LifecycleAndOperator;
import com.amazonaws.services.s3.model.lifecycle.LifecycleFilter;
import com.amazonaws.services.s3.model.lifecycle.LifecycleFilterPredicate;
import com.amazonaws.services.s3.model.lifecycle.LifecyclePrefixPredicate;
import com.amazonaws.services.s3.model.lifecycle.LifecycleTagPredicate;
import com.amazonaws.services.s3.model.Tag;
import java.util.List;
import java.util.ArrayList;

public class setBucketLifecycleConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            //创建BucketLifecycleConfiguration
            BucketLifecycleConfiguration configuration = new
BucketLifecycleConfiguration(new ArrayList<Rule>());
            //创建rule
            Rule currentRule = new Rule();
            currentRule.setId("for java test");
            currentRule.setStatus("Enabled");
            currentRule.setExpirationInDays(365);
            //创建filter
            LifecycleFilter currentFilter = new LifecycleFilter();
            List<LifecycleFilterPredicate> andOperandsList = new
ArrayList<LifecycleFilterPredicate>();
            andOperandsList.add(new LifecyclePrefixPredicate(""));
            andOperandsList.add(new LifecycleTagPredicate(new Tag("key",
"val")));
```

```
            currentFilter.setPredicate(new
LifecycleAndOperator(andOperandsList));
            currentRule.setFilter(currentFilter);
            //rule添加到configuration
            configuration.getRules().add(currentRule);

            //创建SetBucketLifecycleConfigurationRequest
            SetBucketLifecycleConfigurationRequest var = new
SetBucketLifecycleConfigurationRequest(Bucket, configuration);
            s3Client.setBucketLifecycleConfiguration(var);

            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取桶生命周期配置

功能说明

获取 Bucket 的生命周期规则。

方法原型

`BucketLifecycleConfiguration getBucketLifecycleConfiguration(GetBucketLifecycleConfigurationRequest request)`

参数说明

- GetBucketLifecycleConfigurationRequest

设定参数方法	是否必选	描述
GetBucketLifecycleConfigurationRequest(String bucketName)	是	构造函数设置桶名

返回结果说明

- BucketLifecycleConfiguration

获取结果方法	描述
List<Rule> getRules()	获取生命周期规则列表

- Rule

获取结果方法	描述
String getId()	获取ID
LifecycleFilter getFilter()	获取过滤器
String getStatus()	获取规则开启状态： "Enabled"，启用； "Disabled"，不启用
List<Transition> getTransitions()	获取对象过期转存规则
int getExpirationInDays()	获取对象过期删除天数
Date getExpirationDate()	获取对象过期删除日期
List<NoncurrentVersionTransition> getNoncurrentVersionTransitions()	获取对象历史版本过期转存规则
int getNoncurrentVersionExpirationInDays()	获取对象历史版本过期删除天数
boolean isExpiredObjectDeleteMarker()	获取是否删除“删除标记”

- LifecycleFilter

获取结果方法	描述
LifecycleFilterPredicate getPredicate()	获取过滤器

- LifecycleFilterPredicate

获取结果方法	描述
String getPrefix()	子类LifecyclePrefixPredicate：获取前缀
Tag getTag()	子类LifecycleTagPredicate：获取标签

- Tag

获取结果方法	描述
String getKey()	获取标签key
String getValue()	获取标签value

- Transition

获取结果方法	描述
String getStorageClassAsString()	获取转存的存储类型，可选项： "STANDARD"，标准存储； "STANDARD_IA"，低频存储； "GLACIER"，归档存储。
Date getDate()	获取固定时间点转存
int getDays()	获取固定天数后转存
Boolean getIsAccessTime()	获取是否采用基于最后一次访问时间的生命周期规则

- NoncurrentVersionTransition

获取结果方法	描述
String getStorageClassAsString()	获取转存的存储类型，可选项： "STANDARD"，标准存储； "STANDARD_IA"，低频存储； "GLACIER"，归档存储。
int getDays()	获取固定天数后转存
Boolean getIsAccessTime()	获取是否采用基于最后一次访问时间的生命周期规则

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.internal.ServiceUtils;
import com.amazonaws.services.s3.model.GetBucketLifecycleConfigurationRequest;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration.Rule;
import com.amazonaws.services.s3.model.lifecycle.LifecycleAndOperator;
import com.amazonaws.services.s3.model.lifecycle.LifecycleFilter;
import com.amazonaws.services.s3.model.lifecycle.LifecycleFilterPredicate;
import com.amazonaws.services.s3.model.lifecycle.LifecyclePrefixPredicate;
import com.amazonaws.services.s3.model.lifecycle.LifecycleTagPredicate;
import com.amazonaws.services.s3.model.Tag;
import java.util.List;
import java.util.ArrayList;

public class getBucketLifecycleConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
```

```
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            //创建getBucketLifecycleConfigurationRequest
            GetBucketLifecycleConfigurationRequest var = new
GetBucketLifecycleConfigurationRequest(Bucket);
            BucketLifecycleConfiguration bl =
s3Client.getBucketLifecycleConfiguration(var);
            LifecyclePrefixPredicate prefixPredicate;
            List<LifecycleTagPredicate> tagPredicates = new
ArrayList<LifecycleTagPredicate>();
            for (Rule rule : bl.getRules()) {
                tagPredicates.clear();
                System.out.println("Rule begin!");
                System.out.println("Status: " + rule.getStatus());
                System.out.println("Id: " + rule.getId());
                System.out.println("ExpirationInDays: " +
rule.getExpirationInDays());
                LifecycleFilter filter = rule.getFilter();
                if (filter != null) {
                    if
(filter.getPredicate().getClass().getName().contains("LifecyclePrefixPredicate")
)  {
                        prefixPredicate =
(LifecyclePrefixPredicate)filter.getPredicate();
                        System.out.println("Prefix: " +
prefixPredicate.getPrefix());
                    } else if
(filter.getPredicate().getClass().getName().contains("LifecycleTagPredicate")) {

 tagPredicates.add((LifecycleTagPredicate)filter.getPredicate());
                    }
                    for (LifecycleTagPredicate tagPredi: tagPredicates)
                    {
                        System.out.println("Key:" + tagPredi.getTag().getKey());
                        System.out.println("Value:" +
tagPredi.getTag().getValue());
                    }
                }
                System.out.println("Rule end!\n");
            }
            System.out.print("=====request success=====\n");
```

```
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

删除桶生命周期配置

功能说明

删除 Bucket 的生命周期规则。

方法原型

`void deleteBucketLifecycleConfiguration(DeleteBucketLifecycleConfigurationRequest request)`

参数说明

- DeleteBucketLifecycleConfigurationRequest

设定参数方法	是否必选	描述
DeleteBucketLifecycleConfigurationRequest(String bucketName)	是	构造函数设置桶名

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.internal.ServiceUtils;
import
com.amazonaws.services.s3.model.DeleteBucketLifecycleConfigurationRequest;
import java.util.List;
import java.util.ArrayList;

public class deleteBucketLifecycleConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
```

```
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            DeleteBucketLifecycleConfigurationRequest var = new
DeleteBucketLifecycleConfigurationRequest(Bucket);
            s3Client.deleteBucketLifecycleConfiguration(var);

            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

设置桶合规保留配置

功能说明

在指定的存储桶上增加对象锁定配置，默认规则将会应用到每一个新放入桶中的对象。**必须在创建桶时设置开启对象锁，不然后续无法进行配置。**

方法原型

`SetObjectLockConfigurationResult setObjectLockConfiguration(SetObjectLockConfigurationRequest setObjectLockConfigurationRequest)`

参数说明

- SetObjectLockConfigurationRequest

设定参数方法	是否必选	描述
void setBucketName(String bucket)	是	设置桶名
void setObjectLockConfiguration(ObjectLockConfiguration objectLockConfiguration)	是	设置合规保留规则

- ObjectLockConfiguration

设定参数方法	是否必选	描述
void setObjectLockEnabled(String objectLockEnabled)	是	设置是否启用，需设置为：“Enabled”
void setRule(ObjectLockRule rule)	是	设置规则

- ObjectLockRule

设定参数方法	是否必选	描述
void setDefaultRetention(DefaultRetention defaultRetention)	是	设置默认保留期限

- DefaultRetention

设定参数方法	是否必选	描述
void setMode(String mode)	是	设置模式： “COMPLIANCE”，设置后不允许修改； “GOVERNANCE”，设置后允许修改。
void setDays(Integer days)	可选	设置保留天数
void setYears(Integer years)	可选	设置保留年数

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import java.util.*;
import com.amazonaws.services.s3.model.*;

public class setObjectLockConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
```

```
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            DefaultRetention default_retention = new DefaultRetention();
            default_retention.setMode("GOVERNANCE");
            default_retention.setDays(1);

            ObjectLockRule rule = new ObjectLockRule();
            rule.setDefaultRetention(default_retention);

            ObjectLockConfiguration configure = new ObjectLockConfiguration();
            configure.setObjectLockEnabled("Enabled");
            configure.setRule(rule);

            SetObjectLockConfigurationRequest Req = new
SetObjectLockConfigurationRequest();
            Req.setBucketName(Bucket);
            Req.setObjectLockConfiguration(configure);

            SetObjectLockConfigurationResult resp =
s3Client.setObjectLockConfiguration(Req);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取桶合规保留配置

功能说明

获取存储桶的对象锁定配置。默认的对象锁定功能将会应用到每一个新放入到存储桶中的对象。

方法原型

`GetObjectLockConfigurationResult getObjectLockConfiguration(GetObjectLockConfigurationRequest getObjectLockConfigurationRequest )`

参数说明

- GetObjectLockConfigurationRequest

设定参数方法	是否必选	描述
void setBucketName(String bucket)	是	设置桶名

返回结果说明

- GetObjectLockConfigurationResult

获取结果方法	描述
ObjectLockConfiguration getObjectLockConfiguration()	获取合规保留配置

- ObjectLockConfiguration

获取结果方法	描述
String getObjectLockEnabled()	查看是否启用
ObjectLockRule getRule()	获取规则

- ObjectLockRule

获取结果方法	描述
DefaultRetention getDefaultRetention()	获取默认保留期限

- DefaultRetention

获取结果方法	描述
String getMode()	获取模式
Integer getDays()	获取保留天数
Integer getYears()	获取保留年数

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import java.util.*;
import com.amazonaws.services.s3.model.*;

public class getObjectLockConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
```

```
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            GetObjectLockConfigurationRequest Req = new
GetObjectLockConfigurationRequest();
            Req.setBucketName(Bucket);

            GetObjectLockConfigurationResult resp =
s3Client.getObjectLockConfiguration(Req);
            System.out.print("=====request success=====\n");

            ObjectLockConfiguration configure =
resp.getObjectLockConfiguration();
            String object_lock_enabled = configure.getObjectLockEnabled();
            System.out.println(object_lock_enabled);
            ObjectLockRule rule = configure.getRule();
            if(rule != null) {
                DefaultRetention retention = rule.getDefaultRetention();
                String mode = retention.getMode();
                System.out.println("mode: " + mode);
                Integer days = retention.getDays();
                System.out.println("days: " + days);
                Integer years = retention.getYears();
                System.out.println("years: " + years);
            }
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

设置桶跨域访问配置

功能说明

请求设置 Bucket 的跨域资源访问配置。

方法原型

`void setBucketCrossOriginConfiguration(SetBucketCrossOriginConfigurationRequest setBucketCrossOriginConfigurationRequest)`

参数说明

- SetBucketCrossOriginConfigurationRequest

设定参数方法	是否必选	描述
SetBucketCrossOriginConfigurationRequest(String bucketName, BucketCrossOriginConfiguration crossOriginConfiguration)	是	构造函数设置桶名、跨域访问规则

- CORSConfiguration

设定参数方法	是否必选	描述
void setRules(List<CORSRule> rules)	是	设置规则列表

- CORSRule

设定参数方法	是否必选	描述
void setAllowedHeaders(List<String> allowedHeaders)	可选	设置客户端允许携带的headers
void setAllowedMethods(List<CORSRule.AllowedMethods> allowedMethods)	可选	设置允许的http方法
void setAllowedOrigins(List<String> allowedOrigins)	可选	设置允许的域名来源
void setExposedHeaders(List<String> exposedHeaders)	可选	设置响应中允许暴露的headers
void setId(String id)	可选	设置规则ID
void setMaxAgeSeconds(int maxAgeSeconds)	可选	设置有效时间（秒）

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import java.util.ArrayList;
import java.util.List;

public class setBucketCrossOriginConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            CORSRule rule1 = new CORSRule();
            rule1.setId("test1");

 rule1.setAllowedMethods(CORSRule.AllowedMethods.GET,CORSRule.AllowedMethods.PUT
);
            List orgs = new ArrayList();
            orgs.add("http://www.ctyun.cn");
            rule1.setAllowedOrigins(orgs);
            CORSRule rule2 = new CORSRule();
            rule2.setId("test2");
```

```
 rule2.setAllowedMethods(CORSRule.AllowedMethods.GET,CORSRule.AllowedMethods.PUT
,
                    CORSRule.AllowedMethods.HEAD);
            rule2.setAllowedOrigins(orgs);
            BucketCrossOriginConfiguration config = new
BucketCrossOriginConfiguration();
            List<CORSRule> lr = new ArrayList<CORSRule>();
            lr.add(rule1);
            lr.add(rule2);
            config.setRules(lr);
            SetBucketCrossOriginConfigurationRequest req =
                    new SetBucketCrossOriginConfigurationRequest(Bucket,
config);
            s3Client.setBucketCrossOriginConfiguration(req);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取桶跨域访问配置

功能说明

请求获取 Bucket 的跨域资源访问配置。

方法原型

`BucketCrossOriginConfiguration getBucketCrossOriginConfiguration(GetBucketCrossOriginConfigurationRequest getBucketCrossOriginConfigurationRequest )`

参数说明

- GetBucketCorsRequest

设定参数方法	是否必选	描述
GetBucketCrossOriginConfigurationRequest (String bucketName)	是	构造函数设置桶名

返回结果说明

- BucketCrossOriginConfiguration

获取结果方法	描述
List<CORSRule> getRules()	获取跨域访问规则

- CORSRule

获取结果方法	描述
List<String> getAllowedHeaders()	获取客户端允许携带的headers
List<CORSRule.AllowedMethods> getAllowedMethods()	获取允许的http方法
List<String> getAllowedOrigins()	获取允许的域名来源
List<String> getExposedHeaders()	获取响应中允许暴露的headers
String getId()	获取规则ID
int getMaxAgeSeconds()	获取有效时间（秒）

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import java.util.ArrayList;
import java.util.List;

public class getBucketCrossOriginConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            GetBucketCrossOriginConfigurationRequest req =
                    new GetBucketCrossOriginConfigurationRequest(Bucket);
```

```
            BucketCrossOriginConfiguration config =
s3Client.getBucketCrossOriginConfiguration(req);
            System.out.print("=====request success=====\n");
            List<CORSRule> rules = config.getRules();
            for(CORSRule rule : rules){
                System.out.println("Rule: " + rule.getId());
                System.out.println("Allowed Methods: " +
rule.getAllowedMethods());
                System.out.println("Allowed Origins: " +
rule.getAllowedOrigins());
                System.out.println("Allowed Headers: " +
rule.getAllowedHeaders());
            }

        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

删除桶跨域访问配置

功能说明

删除 Bucket 的跨域资源访问配置。

方法原型

`void deleteBucketCrossOriginConfiguration(DeleteBucketCrossOriginConfigurationRequest deleteBucketCrossOriginConfigurationRequest)`

参数说明

- DeleteBucketCrossOriginConfigurationRequest

设定参数方法	是否必选	描述
DeleteBucketCrossOriginConfigurationRequest (String bucketName)	是	构造函数设置桶名

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

```
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import java.util.ArrayList;
import java.util.List;

public class deleteBucketCrossOriginConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            DeleteBucketCrossOriginConfigurationRequest req =
                    new DeleteBucketCrossOriginConfigurationRequest(Bucket);
            s3Client.deleteBucketCrossOriginConfiguration(req);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

设置桶版本控制状态

功能说明

启用或者暂停Bucket的版本控制功能。开启后只能暂停不能关闭。

方法原型

`void setBucketVersioningConfiguration(SetBucketVersioningConfigurationRequest setBucketVersioningConfigurationRequest )`

参数说明

- SetBucketVersioningConfigurationRequest

设定参数方法	是否必选	描述
SetBucketVersioningConfigurationRequest(String bucketName, BucketVersioningConfiguration configuration)	是	构造函数设置桶名、版本控制配置
void setBucketName(String bucketName)	是	设置桶名
setVersioningConfiguration(BucketVersioningConfiguration versioningConfiguration)	是	设置配置

BucketVersioningConfiguration

设定参数方法	是否必选	描述
BucketVersioningConfiguration(String status)	是	构造函数设置是否开启版本控制： "Enabled"，开启； "Suspended"，暂停。
void setStatus(String status)	是	设置是否开启版本控制： "Enabled"，开启； "Suspended"，暂停。

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.SetBucketVersioningConfigurationRequest;

public class setBucketVersioningConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
```

```
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            //设置Bucket Versioning
            BucketVersioningConfiguration innervar = new
BucketVersioningConfiguration("Enabled");
            SetBucketVersioningConfigurationRequest var =
                    new SetBucketVersioningConfigurationRequest
(Bucket,innervar);
            s3Client.setBucketVersioningConfiguration(var);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取桶版本控制状态

功能说明

获得Bucket的版本控制配置。

方法原型

`BucketVersioningConfiguration getBucketVersioningConfiguration(String bucket)`

参数说明

- bucket，桶名

返回结果说明

- BucketVersioningConfiguration

获取结果方法	描述
String getStatus()	获取版本控制开启状态

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;

public class getBucketVersioningConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            //获取Bucket Versioning
            BucketVersioningConfiguration var =
s3Client.getBucketVersioningConfiguration(Bucket);
            System.out.print(var.getStatus() + "\n");
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

设置桶标签

功能说明

为指定的Bucket设置标签。一个Bucket最多设置50个标签。

方法原型

`void setBucketTaggingConfiguration(SetBucketTaggingConfigurationRequest setBucketTaggingConfigurationRequest)`

参数说明

- SetBucketTaggingConfigurationRequest

设定参数方法	是否必选	描述
SetBucketTaggingConfigurationRequest(String bucketName, BucketTaggingConfiguration taggingConfiguration)	是	构造函数设置桶名和标签

- BucketTaggingConfiguration

设定参数方法	是否必选	描述
BucketTaggingConfiguration(Collection<TagSet> tagSets)	是	构造函数设置标签集

- TagSet

设定参数方法	是否必选	描述
TagSet(Map<String, String> tags)	是	构造函数设置标签集合
void setTag(String key, String value)	可选	添加标签

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.*;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class setBucketTaggingConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
```

```
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            Map<String, String> tags = new HashMap<String, String>();
            tags.put("key1", "val1");
            tags.put("key2", "val2");
            TagSet tagSet = new TagSet(tags);
            tagSet.setTag("key3", "val3");
            List<TagSet> tagSetList = new ArrayList<>();
            tagSetList.add(tagSet);
            BucketTaggingConfiguration bucketTaggingConfiguration = new
BucketTaggingConfiguration(tagSetList);
            SetBucketTaggingConfigurationRequest req = new
SetBucketTaggingConfigurationRequest(Bucket, bucketTaggingConfiguration);
            s3Client.setBucketTaggingConfiguration(req);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取桶标签

功能说明

返回桶的标签集。

方法原型

`BucketTaggingConfiguration getBucketTaggingConfiguration(GetBucketTaggingConfigurationRequest getBucketTaggingConfigurationRequest)`

参数说明

- GetBucketTaggingConfigurationRequest

设定参数方法	是否必选	描述
GetBucketTaggingConfigurationRequest(String bucketName)	是	构造函数设置桶名

返回结果说明

- BucketTaggingConfiguration

获取结果方法	描述
List<TagSet> getAllTagSets()	获取标签集

- TagSet

设定参数方法	描述
Map<String, String> getAllTags()	获取标签

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.*;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class getBucketTaggingConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
```

```
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            GetBucketTaggingConfigurationRequest req = new
GetBucketTaggingConfigurationRequest(Bucket);
            BucketTaggingConfiguration bucketTaggingConfiguration =
s3Client.getBucketTaggingConfiguration(req);
            List<TagSet> tagSetList =
bucketTaggingConfiguration.getAllTagSets();
            for (TagSet tagSet: tagSetList) {
                Map<String, String> tags = tagSet.getAllTags();
                System.out.println(tags);
            }
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

删除桶标签

功能说明

从指定的桶中删除整个标记集。

方法原型

`void deleteBucketTaggingConfiguration(DeleteBucketTaggingConfigurationRequest deleteBucketTaggingConfigurationRequest)`

参数说明

- DeleteBucketTaggingConfigurationRequest

设定参数方法	是否必选	描述
DeleteBucketTaggingConfigurationRequest(String bucketName)	是	构造函数设置桶名

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.*;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class deleteBucketTaggingConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            DeleteBucketTaggingConfigurationRequest req = new
DeleteBucketTaggingConfigurationRequest(Bucket);
            s3Client.deleteBucketTaggingConfiguration(req);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

设置桶日志转存配置

功能说明

设置日志转存参数。所有的日志将会保留到和源存储桶属于同一拥有者的目标存储桶中。

方法原型

`void setBucketLoggingConfiguration(SetBucketLoggingConfigurationRequest setBucketLoggingConfigurationRequest)`

参数说明

- SetBucketLoggingConfigurationRequest

设定参数方法	是否必选	描述
SetBucketLoggingConfigurationRequest(String bucketName, BucketLoggingConfiguration loggingConfiguration)	是	构造函数设置桶名、日志转存配置

- BucketLoggingConfiguration，**当两个参数都设置时才是开启状态，想要关闭只要不设置参数就行。**

设定参数方法	是否必选	描述
void setDestinationBucketName(String destinationBucketName)	可选	设置存储日志的桶
void setLogFilePrefix(String logFilePrefix)	可选	设置日志路径前缀

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import java.util.*;
import com.amazonaws.services.s3.model.*;

public class setBucketLoggingConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
```

```
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            BucketLoggingConfiguration configure = new
BucketLoggingConfiguration();
            configure.setDestinationBucketName(Bucket);
            configure.setLogFilePrefix("log/");

            SetBucketLoggingConfigurationRequest req = new
SetBucketLoggingConfigurationRequest(Bucket, configure);
            s3Client.setBucketLoggingConfiguration(req);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取桶日志转存配置

功能说明

获取存储桶的日志转存配置。

方法原型

`BucketLoggingConfiguration getBucketLoggingConfiguration(GetBucketLoggingConfigurationRequest getBucketLoggingConfigurationRequest)`

参数说明

- GetBucketLoggingConfigurationRequest

设定参数方法	是否必选	描述
GetBucketLoggingConfigurationRequest(String bucketName)	是	设置桶名

返回结果说明

- BucketLoggingConfiguration

获取结果方法	描述
String getDestinationBucketName()	获取存储日志的桶名
String getLogFilePrefix()	获取日志前缀

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import java.util.*;
import com.amazonaws.services.s3.model.*;

public class getBucketLoggingConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
```

```
            GetBucketLoggingConfigurationRequest req = new
GetBucketLoggingConfigurationRequest(Bucket);
            BucketLoggingConfiguration configure =
s3Client.getBucketLoggingConfiguration(req);
            System.out.print("=====request success=====\n");
            System.out.println(configure);
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

设置服务端加密配置

功能说明

启用存储桶默认加密功能

方法原型

`SetBucketEncryptionResult setBucketEncryption(SetBucketEncryptionRequest setBucketEncryptionRequest)`

参数说明

- SetBucketEncryptionRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名
void setServerSideEncryptionConfiguration(ServerSideEncryptionConfiguration serverSideEncryptionConfiguration)	是	设置加密配置

- ServerSideEncryptionConfiguration

设定参数方法	是否必选	描述
void setRules(Collection<ServerSideEncryptionRule> rules)	是	设置加密规则

- ServerSideEncryptionRule

设定参数方法	是否必选	描述
void setApplyServerSideEncryptionByDefault(ServerSideEncryptionByDefault applyServerSideEncryptionByDefault)	是	设置加密方式

- ServerSideEncryptionByDefault

设定参数方法	是否必选	描述
void setSSEAlgorithm(String sseAlgorithm)	是	设置加密算法： "AES256"; "aws:kms"
void setKMSMasterKeyID(String kmsMasterKeyID)	可选（加密算法为kms时必选）	设置加密密钥，当选择"AES256"时，长度需为32位；当选择"aws:kms"时，需以cmkuuid:keyspec:userid的格式。

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import java.util.*;
import com.amazonaws.services.s3.model.*;

public class setBucketEncryption {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
```

```
            ServerSideEncryptionByDefault SSE_default = new
ServerSideEncryptionByDefault();
            SSE_default.setSSEAlgorithm("AES256");

            ServerSideEncryptionRule rule = new ServerSideEncryptionRule();
            rule.setApplyServerSideEncryptionByDefault(SSE_default);

            List<ServerSideEncryptionRule> rules = new ArrayList<>();
            rules.add(rule);

            ServerSideEncryptionConfiguration SSEConfigure = new
ServerSideEncryptionConfiguration();
            SSEConfigure.setRules(rules);

            SetBucketEncryptionRequest putReq = new
SetBucketEncryptionRequest();
            putReq.setBucketName(Bucket);
            putReq.setServerSideEncryptionConfiguration(SSEConfigure);

            s3Client.setBucketEncryption(putReq);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取服务端加密配置

功能说明

返回存储桶默认加密配置。若是存储桶不存在默认加密配置，则返回NoSuchEncryptionSetError错误。

方法原型

`GetBucketEncryptionResult getBucketEncryption(GetBucketEncryptionRequest getBucketEncryptionRequest)`

参数说明

- GetBucketEncryptionRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名

返回结果说明

- GetBucketEncryptionResult

获取结果方法	描述
ServerSideEncryptionConfiguration getServerSideEncryptionConfiguration()	获取服务端加密配置

- ServerSideEncryptionConfiguration

获取结果方法	描述
List<ServerSideEncryptionRule> getRules()	获取服务端加密规则

- ServerSideEncryptionRule

获取结果方法	描述
ServerSideEncryptionByDefault getApplyServerSideEncryptionByDefault()	获取服务端加密规则

- ServerSideEncryptionByDefault

获取结果方法	描述
String getSSEAlgorithm()	获取服务端加密算法
String getKMSMasterKeyID()	获取密钥

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import java.util.*;
import com.amazonaws.services.s3.model.*;

public class getBucketEncryption {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
```

```
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            GetBucketEncryptionRequest Req = new GetBucketEncryptionRequest();
            Req.setBucketName(Bucket);

            GetBucketEncryptionResult resp = s3Client.getBucketEncryption(Req);
            System.out.print("=====request success=====\n");
            ServerSideEncryptionConfiguration SSE_Configure =
resp.getServerSideEncryptionConfiguration();
            List<ServerSideEncryptionRule> rules = SSE_Configure.getRules();
            for (ServerSideEncryptionRule rule : rules) {
                System.out.println(rule);
            }
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

删除服务端加密配置

功能说明

删除存储桶默认加密配置

方法原型

`DeleteBucketEncryptionResult deleteBucketEncryption(DeleteBucketEncryptionRequest deleteBucketEncryptionRequest)`

参数说明

- DeleteBucketEncryptionRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import java.util.*;
```

```
import com.amazonaws.services.s3.model.*;

public class deleteBucketEncryption {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            DeleteBucketEncryptionRequest Req = new
DeleteBucketEncryptionRequest();
            Req.setBucketName(Bucket);

            s3Client.deleteBucketEncryption(Req);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

设置桶静态网站配置

功能说明

将桶设置成静态网站托管模式并设置跳转规则。

方法原型

void setBucketWebsiteConfiguration(SetBucketWebsiteConfigurationRequest setBucketWebsiteConfigurationRequest)

参数说明

- SetBucketWebsiteConfigurationRequest

设定参数方法	是否必选	描述
SetBucketWebsiteConfigurationRequest(String bucketName, BucketWebsiteConfiguration configuration)	是	构造函数设置桶名、静态网站配置

- BucketWebsiteConfiguration，**ErrorDocument、IndexDocumentSuffix、RoutingRules三项参数结合使用（不能全为空）并与RedirectAllRequestsTo互斥，当设置了这三个字段时，不能设置RedirectAllRequestsTo；反之，当设置了RedirectAllRequestsTo时，不能设置这三项。**

设定参数方法	是否必选	描述
void setErrorDocument(String errorDocument)	可选	当4XX错误出现时使用的对象的名称。这个元素指定当错误出现时返回的页面。
void setIndexDocumentSuffix(String indexDocumentSuffix)	可选	设置索引文档(该字段被追加在对文件夹的请求的末尾（例如：参数设置为“index.html”，请求的是“samplebucket/images/”，返回的数据将是“samplebucket”桶内名为“images/index.html”的对象的内容）。
void setRoutingRules(List<RoutingRule> routingRules)	可选	设置重定向规则
void setRedirectAllRequestsTo(RedirectRule redirectAllRequestsTo)	可选	设置所有请求重定向规则

- RoutingRule

设定参数方法	是否必选	描述
void setCondition(RoutingRuleCondition condition)	可选	设置重定向规则的匹配条件
void setRedirect(RedirectRule redirect)	必选	重定向请求时的具体信息。

- RoutingRuleCondition

设定参数方法	是否必选	描述
void setKeyPrefixEquals(String keyPrefixEquals)	可选	如果对象名前缀等于这个值，那么重定向生效。
void setHttpErrorCodeReturnedEquals(String httpErrorCodeReturnedEquals)	可选	当发生错误时，如果错误码等于这个值，那么重定向生效。（当跳转规则类型为镜像回源时，此项必须设置为404）

- RedirectRule

设定参数方法	是否必选	描述
void setType(String type)	可选	设置跳转的类型（不设置时为静态网站相关跳转，设置时则启用数据回源功能）： 空，为静态网站相关跳转 "MIRROR"，镜像回源； "REDIRECTION"，外部跳转，返回3XX请求，指定跳转到另一个地址；
void setProtocol(String protocol)	可选	设置重定向使用的协议，"http"或"https"，默认使用源请求的协议
void setHostName(String hostName)	必选	设置重定向主机名
void setReplaceKeyPrefixWith(String replaceKeyPrefixWith)	可选	指定重定向规则的具体重定向目标的对象键，替换方式为替换原始请求中所匹配到的前缀部分
void setReplaceKeyWith(String replaceKeyWith)	可选	指定重定向规则的具体重定向目标的对象键，替换方式为替换整个原始请求的对象键
void setHttpRedirectCode(String httpRedirectCode)	可选	指定重定向规则的错误码匹配条件，只支持配置4XX返回码，例如403或404
void setMirrorFollowRedirect(boolean mirrorFollowRedirect)	可选	默认false，只有Type设置为"MIRROR"时才生效，当镜像回源获取结果为3XX时，是否继续跳转到指定Location获取数据。例如发起镜像回源请求时，源站返回302，并且指定了location。如果此项设置为true，则ZOS会继续请求location对应的地址；如果为false，则ZOS会返回302，并透传location。

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

public class setBucketWebsiteConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            BucketWebsiteConfiguration website_config = new
BucketWebsiteConfiguration();
            website_config.setErrorDocument("err.html");
            website_config.setIndexDocumentSuffix("index.html");
            SetBucketWebsiteConfigurationRequest req =
                    new SetBucketWebsiteConfigurationRequest(Bucket,
website_config);
            s3Client.setBucketWebsiteConfiguration(req);
            System.out.println("=====request success=====\n");
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取桶静态网站配置

功能说明

查询与存储桶关联的静态网站配置信息。

方法原型

```
BucketWebsiteConfiguration
getBucketWebsiteConfiguration(GetBucketWebsiteConfigurationRequest request)
```

参数说明

- GetBucketWebsiteConfigurationRequest

设定参数方法	是否必选	描述
GetBucketWebsiteConfigurationRequest(String bucketName)	是	构造函数设置桶名

返回结果说明

- BucketWebsiteConfiguration

获取结果方法	描述
String getErrorDocument()	获取错误文档配置
String getIndexDocumentSuffix()	获取索引文档
List<RoutingRule> getRoutingRules()	获取重定向规则配置
RedirectRule getRedirectAllRequestsTo()	获取对所有请求重定向的配置

- RoutingRule

获取结果方法	描述
RoutingRuleCondition getCondition()	获取重定向规则的匹配条件
RedirectRule getRedirect()	获取重定向规则配置信息

- RoutingRuleCondition

获取结果方法	描述
String getKeyPrefixEquals()	获取前缀匹配条件
String getHttpErrorCodeReturnedEquals()	获取错误码匹配条件

- RedirectRule

获取结果方法	描述
String getType()	获取跳转类型
String getprotocol()	获取重定向协议配置
String getHostName()	获取重定向主机名
String getReplaceKeyPrefixWith()	获取重定向规则的具体重定向目标的对象键前缀
String getReplaceKeyWith()	获取重定向规则的具体重定向目标的对象键
String getHttpRedirectCode()	获取重定向规则的错误码匹配条件
boolean getMirrorFollowRedirect()	获取是否继续跳转到指定Location获取数据

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.BucketWebsiteConfiguration;
import com.amazonaws.services.s3.model.GetBucketWebsiteConfigurationRequest;
import com.amazonaws.services.s3.model.RedirectRule;

public class getBucketWebsiteConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            GetBucketWebsiteConfigurationRequest req = new
GetBucketWebsiteConfigurationRequest(Bucket);
```

```
            BucketWebsiteConfiguration website_config =
s3Client.getBucketWebsiteConfiguration(req);
            System.out.println("Index: " +
website_config.getIndexDocumentSuffix());
            System.out.println("Err: " + website_config.getErrorDocument());
            System.out.println("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

删除桶静态网站配置

功能说明

删除桶的静态网站配置。

方法原型

`void deleteBucketWebsiteConfiguration(DeleteBucketWebsiteConfigurationRequest request)`

参数说明

- DeleteBucketWebsiteConfigurationRequest

设定参数方法	是否必选	描述
DeleteBucketWebsiteConfigurationRequest(String bucketName)	是	构造函数设置桶名

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.BucketWebsiteConfiguration;
import com.amazonaws.services.s3.model.DeleteBucketWebsiteConfigurationRequest;
import com.amazonaws.services.s3.model.RedirectRule;

public class deleteBucketWebsiteConfiguration {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
```

```
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            DeleteBucketWebsiteConfigurationRequest req = new
DeleteBucketWebsiteConfigurationRequest(Bucket);
            s3Client.deleteBucketWebsiteConfiguration(req);
            System.out.println("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

对象操作

下载对象

功能说明

将一个文件（Object）下载至本地。

方法原型

`S3Object getObject(GetObjectRequest getObjectRequest )`

参数说明

- GetObjectRequest

设定参数方法	是否必选	描述
GetObjectRequest(String bucketName, String key)	可选	构造函数设置桶名、对象名
GetObjectRequest(String bucketName, String key, String versionId)	可选	构造函数设置桶名、对象名、版本号
void setMatchingETagConstraints(List<String> eTagList)	可选	当对象的ETag和值相同时才下载
void setNonmatchingETagConstraints(List<String> eTagList)	可选	当对象的ETag和值不同时才下载
void setModifiedSinceConstraint(Date date)	可选	只有指定时间之后有修改记录的对象才会返回
void setUnmodifiedSinceConstraint(Date date)	可选	只有指定时间之后没有修改记录的对象才会返回
void setRange(long start, long end)	可选	指定下载对象的字节范围

返回结果说明

- S3Object

获取结果方法	描述
S3ObjectInputStream getObjectContent()	获取对象内容
String getBucketName()	获取对象所在桶
String getKey()	获取对象key
ObjectMetadata getObjectMetadata()	获取对象元数据

- ObjectMetadata

获取结果方法	描述
Map<String, String> getUserMetadata()	获取用户元数据
Date getLastModified()	获取对象创建时间
String getETag()	获取对象ETag
String getVersionId()	获取对象版本号
Long[] getContentRange()	若请求时指定了Range或partnumber，则返回结果中包含content range
String getObjectLockMode()	获取对象的合规保留模式
Date getObjectLockRetainUntilDate()	获取Retention 过期日期
String getStorageClass()	获取对象存储类型

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.S3Object;

import java.io.*;

public class getObject {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String savePath = "<your-path>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
```

```
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            GetObjectRequest request = new GetObjectRequest(Bucket, Key);
            S3Object result = s3Client.getObject(request);
            System.out.print("=====request success=====\n");
            // 保存到本地
            try {
                InputStream in = result.getObjectContent();
                File outputFile = new File(savePath);
                FileOutputStream outputStream = new
FileOutputStream(outputFile);

                byte[] read_buf = new byte[1024 * 1024];
                int read_len = 0;
                while ((read_len = in.read(read_buf)) > 0) {
                    outputStream.write(read_buf, 0, read_len);
                }
                in.close();
                outputStream.close();
            } catch (IOException e){
                e.printStackTrace();
            }
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

上传对象

功能说明

将一个文件（Object）上传至指定 Bucket。

方法原型

`PutObjectResult putObject(PutObjectRequest putObjectRequest)`

参数说明

- PutObjectRequest

设定参数方法	是否必选	描述
PutObjectRequest(String bucketName, String key, File file)	是	构造函数设置桶名、对象名、待上传文件
void setBucketName(String bucketName)	是	设置桶名
void setKey(String key)	是	设置对象名
void setCannedAcl(CannedAccessControlList cannedAcl)	可选	设置对象CannedAccessControlList： CannedAccessControlList.Private，私有（默认）； CannedAccessControlList.PublicRead，公共读，但只有对象所有者可以写入和更改ACL； CannedAccessControlList.BucketOwnerRead，对象所有者和存储桶所有者可以读取该对象，只有对象所有者可以写入和更改ACL； CannedAccessControlList.BucketOwnerFullControl，对象所有者和存储桶所有者可以读取和写入该对象以及更改ACL。
void setAccessControlList(AccessControlList accessControlList)	可选	设置对象AccessControlList 。
void setMetadata(ObjectMetadata metadata)	可选	设置对象元数据
void setObjectLockMode(String objectLockMode)	可选	设置合规保留模式： "COMPLIANCE"，设置后不允许修改； "GOVERNANCE"，设置后允许修改。
void setObjectLockRetainUntilDate(Date objectLockRetainUntilDate)	可选	设置保留到期时间
void setTagging(ObjectTagging tagging)	可选	设置对象标签，格式："key=value"
void setStorageClass(StorageClass storageClass)	可选	设置桶存储类型，可选项： StorageClass.Standard，标准存储； StorageClass.StandardInfrequentAccess，低频存储； StorageClass.Glacier，归档存储。
void setAppendEnabled(Boolean appendEnabled)	可选	设置追加模式上传Object
void setAppendPosition(long appendPosition)	可选	设置追加上传的起始位置，单位Byte

- AccessControlList

设定参数方法	是否必选	描述
void setOwner(Owner owner)	是	设置桶所有者
void grantPermission(Grantee grantee, Permission permission)	是	设置授权角色及权限： Permission.FullControl，所有权限； Permission.Read，读权限； Permission.ReadACP，读ACL权限； Permission.Write，写权限； Permission.WriteAcp，写ACL权限。

- Owner

设定参数方法	是否必选	描述
void setId(String id)	是	设置ID
void setDisplayName(String name)	否	设置名字

- Grantee，接口类，两种实现：CanonicalGrantee、EmailAddressGrantee

设定参数方法	是否必选	描述
CanonicalGrantee(String identifier)	可选	构造函数，设置被授权用户id
EmailAddressGrantee(String emailAddress)	可选	构造函数，设置被授权用户邮箱

- ObjectMetadata

设定参数方法	是否必选	描述
void addUserMetadata(String key, String value)	可选	添加用户元数据

返回结果说明

- PutObjectResult

获取结果方法	描述
String getETag()	获取对象ETag
String getVersionId()	获取版本号
long getAppendPosition()	如果设置了追加上传，获取下一次追加的位置

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.PutObjectResult;

import java.io.File;

public class putObject {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String filepath = "<your-path>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            // 上传 Object
            File file= new File(filepath);
            PutObjectRequest request = new PutObjectRequest(Bucket, Key, file);
            PutObjectResult result = s3Client.putObject(request);
            System.out.format("Etag: %s, VersionId: %s\n",result.getETag(),
result.getVersionId());
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

上传对象-Post上传

功能说明

将一个文件（Object）通过Post方式上传至指定 Bucket。

方法原型

`String postObject(PostObjectRequest postObjectRequest)`

参数说明

- PostObjectPolicy

设定参数方法	是否必选	描述
PostObjectPolicy()	是	构造函数
void setExpiration(int expiredSecond)	否	设置policy失效的时间，默认为900
void addEqualCondition(String property, String value)	否	设置精确匹配条件
void addMatchCondition(String property, String value)	否	设置前缀匹配条件

Policy是使用 UTF-8 和 Base64 编码的 JSON 文档，它指定了请求必须满足的条件并且用于对内容进行身份验证。

POST Policy 示例

```
{ "expiration": "2022-04-14T12:00:00.000Z",
  "conditions": [
    {"acl": "private" },
    {"bucket": "ctyun" },
    ["starts-with", "$key", "file/archive/"],
  ]
}
```

PostObjectPolicy总是包含 expiration 和 conditions 字段。

expiration 字段表示该policy失效的时间，例如例子中的policy将在 GMT 2022-04-14 12:00:00 失效。

conditions 包含了一组约束条件，POST请求需要满足conditions中给出的条件才能请求成功。在form中出现的每一个字段（除去 x-amz-signature, file, policy 和 x-ignore-开头的字段），都需要在conditions中指明约束。

- PostObjectRequest

设定参数方法	是否必选	描述
PostObjectRequest generatePostObjectRequest(String bucketName, String objectKey, String filePath, PostObjectPolicy policy)	是	创建Post上传请求，设置桶名、对象名、待上传文件路径、policy
void addFormFields(String key, String value)	否	添加form字段

返回结果说明

- PostObjectResult

获取结果方法	描述
int getResponseCode()	获取请求的状态码
String getMessage()	获取请求的返回消息

代码示例

```
import java.util.Map;
import java.util.Map.Entry;

public class postObject {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String filePath = "<your-path>";

    public static void main(String[] args) throws Exception {
        PostObjectSDK sdk = new PostObjectSDK(ACCESS_KEY, SECRET_KEY,
END_POINT);

        PostObjectSDK.PostObjectPolicy policy = new
PostObjectSDK.PostObjectPolicy();
        policy.setExpiration(900);
        policy.addEqualCondition("acl", "private");
        System.out.println("Policy:" + policy.getPolicy());

        PostObjectSDK.PostObjectRequest request =
sdk.generatePostObjectRequest(Bucket, Key, filePath, policy);
        request.addFormFields("acl", "private");

        Map<String, String> formFields = request.getFormFields();
        System.out.println("formFields:");
        for (Entry<String, String> entry : formFields.entrySet()) {
            System.out.println("    " + entry.getKey() + ":" +
entry.getValue());
        }

        // post上传 object
```

```
        PostObjectSDK.PostObjectResult result =
PostObjectSDK.postObject(request);
        System.out.println("Post Object " + Key + " to bucket " + Bucket);
        System.out.println("Response code: " + result.getResponseCode());
        System.out.println("Response message: " + result.getMessage());
    }
}
```

PostObjectSDK

```
import org.json.JSONException;
import org.json.JSONObject;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.*;

public class PostObjectSDK {
    private final String accessKeyId;
    private final String accessKeySecret;
    private final String endpoint;

    public PostObjectSDK(String ak, String sk, String endpoint) {
        this.accessKeyId = ak;
        this.accessKeySecret = sk;
        this.endpoint = endpoint;
    }

    public static class PostObjectRequest {
        private String Url;
        private String FilePath;
        private Map<String, String> FormFields;

        public PostObjectRequest(String url, String filePath, Map<String,
String> formFields) {
            this.Url = url;
            this.FilePath = filePath;
            this.FormFields = formFields;
        }

        public String getUrl() {
            return Url;
        }

        public String getFilePath() {
            return FilePath;
        }

        public Map<String, String> getFormFields() {
```

```
            return FormFields;
        }

        public void addFormFields(String key, String value) {
            FormFields.put(key, value);
        }
    }

    public static class PostObjectResult {
        private final int responseCode;
        private final String message;

        public PostObjectResult(int responseCode, String message) {
            this.responseCode = responseCode;
            this.message = message;
        }

        public int getResponseCode() {
            return responseCode;
        }

        public String getMessage() {
            return message;
        }
    }

    public static class PostObjectPolicy {
        private String expiration;
        private List<Object> conditions = new ArrayList<>();

        public PostObjectPolicy() {
            setExpiration(900);
        }

        public void addEqualCondition(String property, String value) {
            conditions.add(Arrays.asList("eq", "$" + property, value));
        }

        public void addMatchCondition(String property, String value) {
            conditions.add(Arrays.asList("starts-with", "$" + property, value));
        }

        public void setExpiration(int expiredSecond) {
            Date now = new Date();
            Date expire = new Date(now.getTime() + expiredSecond * 1000L);

            DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss'Z'");
            dateFormat.setTimeZone(TimeZone.getTimeZone("UTC"));
            expiration = dateFormat.format(expire);
        }

        public String getPolicy() throws JSONException {
            JSONObject jsonObject = new JSONObject();
            jsonObject.put("expiration", expiration);
            jsonObject.put("conditions", conditions);
            return jsonObject.toString();
        }
```

```
    }

    static class Signature {
        private final static char[] hexArray = "0123456789ABCDEF".toCharArray();

        public static String bytesToHex(byte[] bytes) {
            char[] hexChars = new char[bytes.length * 2];
            for (int j = 0; j < bytes.length; j++) {
                int v = bytes[j] & 0xFF;
                hexChars[j * 2] = hexArray[v >>> 4];
                hexChars[j * 2 + 1] = hexArray[v & 0x0F];
            }
            return new String(hexChars).toLowerCase();
        }

        public static byte[] HmacSHA256(byte[] key, String data) throws
Exception {
            String algorithm = "HmacSHA256";
            Mac mac = Mac.getInstance(algorithm);
            mac.init(new SecretKeySpec(key, algorithm));
            return mac.doFinal(data.getBytes("UTF8"));
        }

        public static String compute(String currentDate, String Region,
                                     String accessKeySecret, String
encodePolicy) throws Exception {
            String serviceName = "s3";
            byte[] kSecret = ("AWS4" + accessKeySecret).getBytes("UTF8");
            byte[] kDate = Signature.HmacSHA256(kSecret, currentDate);
            byte[] kRegion = Signature.HmacSHA256(kDate, Region);
            byte[] kService = Signature.HmacSHA256(kRegion, serviceName);
            byte[] kSigning = Signature.HmacSHA256(kService, "aws4_request");
            byte[] signature = Signature.HmacSHA256(kSigning, encodePolicy);
            return Signature.bytesToHex(signature);
        }
    }

    private String getDate(Date date) {
        DateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");
        dateFormat.setTimeZone(TimeZone.getTimeZone("UTC"));//server timezone
        return dateFormat.format(date);
    }

    private String getTimeStamp(Date date) {
        DateFormat dateFormat = new SimpleDateFormat("yyyyMMdd'T'HHmmss'Z'");
        dateFormat.setTimeZone(TimeZone.getTimeZone("UTC"));//server timezone
        return dateFormat.format(date);
    }

    public PostObjectRequest generatePostObjectRequest(String bucketName, String
objectKey, String filePath, PostObjectPolicy policy) throws Exception {
        String urlStr = endpoint + "/" + bucketName;

        Date date = new Date();
        String currentDate = getDate(date);
        String amzDate = getTimeStamp(date);
        String region = "cn";
        String success_action_status = "201";
```

```
        String credential = accessKeyId + "/" + currentDate + "/" + region +
"/s3/aws4_request";

        Map<String, String> formFields = new HashMap<>();
        formFields.put("key", objectKey);
        formFields.put("x-amz-algorithm", "AWS4-HMAC-SHA256");
        formFields.put("x-amz-credential", credential);
        formFields.put("x-amz-date", amzDate);
        formFields.put("success_action_status", success_action_status);

        policy.addEqualCondition("bucket", bucketName);
        policy.addMatchCondition("key", objectKey);
        policy.addEqualCondition("x-amz-algorithm", "AWS4-HMAC-SHA256");
        policy.addEqualCondition("x-amz-credential", credential);
        policy.addEqualCondition("x-amz-date", amzDate);
        policy.addEqualCondition("success_action_status",
success_action_status);
        String encodePolicy = new
String(Base64.getEncoder().encodeToString(policy.getPolicy().getBytes()));
        formFields.put("Policy", encodePolicy);

        // Signature
        String signaturecom = Signature.compute(currentDate, region,
accessKeySecret, encodePolicy);
        formFields.put("x-amz-signature", signaturecom);

        return new PostObjectRequest(urlStr, filePath, formFields);
    }

    public static PostObjectResult postObject(PostObjectRequest request) throws
Exception {
        HttpURLConnection conn = null;
        int responseCode = -1;
        String message = "";
        try {
            String boundary = "9431149156168";
            URL url = new URL(request.getUrl());
            conn = (HttpURLConnection) url.openConnection();
            conn.setConnectTimeout(5000);
            conn.setReadTimeout(30000);
            conn.setDoOutput(true);
            conn.setDoInput(true);
            conn.setRequestMethod("POST");
            conn.setRequestProperty("User-Agent", "Mozilla/5.0 (Windows; U;
Windows NT 6.1; zh-CN; rv:1.9.2.6)");
            conn.setRequestProperty("Content-Type", "multipart/form-data;
boundary=" + boundary);

            OutputStream out = new DataOutputStream(conn.getOutputStream());

            Map<String, String> formFields = request.getFormFields();
            if (formFields != null) {
                StringBuilder strBuf = new StringBuilder();
                int i = 0;
                for (Map.Entry<String, String> entry : formFields.entrySet()) {
                    String inputName = entry.getKey();
                    String inputValue = entry.getValue();
                    if (inputValue == null) continue;
```

```
                    if (i > 0) {
                        strBuf.append("\r\n");
                    }
                    strBuf.append("--").append(boundary).append("\r\n");
                    strBuf.append("Content-Disposition: form-data;
name=\"").append(inputName).append("\"\r\n\r\n");
                    strBuf.append(inputValue);
                    i++;
                }
                out.write(strBuf.toString().getBytes());
            }

            String filePath = request.getFilePath();
            File file = new File(filePath);
            String filename = file.getName();
            Path path = Paths.get(filePath);
            String contentType = Files.probeContentType(path);
            if (contentType == null || contentType.equals("")) {
                contentType = "application/octet-stream";
            }
            StringBuilder strBuf = new StringBuilder();
            strBuf.append("\r\n--").append(boundary).append("\r\n");
            strBuf.append("Content-Disposition: form-data; name=\"file\";
filename=\"").append(filename).append("\"\r\n");
            strBuf.append("Content-Type:
").append(contentType).append("\r\n\r\n");
            out.write(strBuf.toString().getBytes());
            DataInputStream in = new DataInputStream(new FileInputStream(file));
            int bytes = 0;
            byte[] bufferOut = new byte[1024];
            while ((bytes = in.read(bufferOut)) != -1) {
                out.write(bufferOut, 0, bytes);
            }
            in.close();

            byte[] endData = ("\r\n--" + boundary + "--\r\n").getBytes();
            out.write(endData);
            out.flush();
            out.close();

            strBuf = new StringBuilder();
            BufferedReader reader = null;
            responseCode = conn.getResponseCode();
            if (responseCode == HttpURLConnection.HTTP_OK || responseCode ==
HttpURLConnection.HTTP_CREATED || responseCode ==
HttpURLConnection.HTTP_NO_CONTENT) {
                reader = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
            } else {
                reader = new BufferedReader(new
InputStreamReader(conn.getErrorStream()));
            }
            String line = null;
            while ((line = reader.readLine()) != null) {
                strBuf.append(line).append("\n");
            }
            message = strBuf.toString();
            reader.close();
```

```
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (conn != null) {
                conn.disconnect();
            }
        }
        return new PostObjectResult(responseCode, message);
    }
}
```

断点续传

功能说明

实现文件的断点续传功能，通过传入Bucket、Key、文件名以及回调函数，能够在上传大文件但因各种原因中断时，再次调用该接口，通过本地保存的checkpoint文件信息，继续上传未上传的分段。

注意：

1. 文件小于8M不开启断点续传，文件partSize默认为8M，自行设置分段大小需要大于等于8M
2. 同一个断点续传任务需要保证partSize前后一致
3. 断点续传需要手动开启setEnableCheckpoint功能

方法原型

`ResumeUploadResult resumeUploadFile(ResumeUploadRequest resumeUploadRequest, UploadProcessCallback callback)`

参数说明

- ResumeUploadRequest

设定参数方法	是否必选	描述
ResumeUploadRequest(String bucketName, String key, String filePath)	是	构造函数设置桶名，对象名和文件路径
void setBucketName(String bucketName)	是	设置桶名
void setKey(String key)	是	设置对象名
void setFilePath(String filePath)	是	设置文件路径
void setPartSize(long partSize)	否	设置分段大小
void setEnableCheckpoint(bool enableCheckpoint)	否	设置是否开启断点续传的checkpoint文件保留
void setCheckpointPath(String checkpointPath)	否	设置checkpoint文件保留路径
void setStorageClass(StorageClass storageClass)	否	设置上传文件存储类型
void setCannedAcl(CannedAccessControlList cannedAcl)	否	设置对象CannedAccessControlList
void setAccessControlList(AccessControlList accessControlList)	否	设置对象AccessControlList
void setMetadata(ObjectMetadata metadata)	否	设置对象元数据
void setObjectLockMode(String objectLockMode)	否	设置合规保留模式："COMPLIANCE"，设置后不允许修改；"GOVERNANCE"，设置后允许修改。
void setObjectLockRetainUntilDate(Date objectLockRetainUntilDate)	否	设置保留到期时间
void setObjectLockLegalHoldStatus(String objectLockLegalHoldStatus)	否	设置依法保留状态
void setTagging(ObjectTagging tagging)	否	设置对象标签，格式："key=value"

- UploadProcessCallback

设定参数方法	是否必选	描述
Function(long currentBytes, long totalBytes)	否	回调函数，获取当前已上传字节数和总字节数

返回结果说明

- ResumeUploadResult

获取结果方法	描述
String getETag()	获取整个文件ETag
String getVersionId()	获取整个文件版本号

代码示例

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.*;

public class UploadTest {
    public static String ACCESS_KEY = "<your-access_key>";
    public static String SECRET_KEY = "<your-secret_key>";
    public static String END_POINT = "<your-end_point>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String FilePath = "<your-filePath>";
    public static long PartSize = "<your-partSize>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try{
            AWSCredentials credentials = new
BasicAWSCredentials(ACCESS_KEY,SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
```

```
            .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
            .disableChunkedEncoding()
            .enablePathStyleAccess()
            .build();
            System.out.print("=====connect success=====\n");
            //开启文件断点续传
            ResumeUploadRequest resumeUploadRequest = new ResumeUploadRequest();
            resumeUploadRequest.setBucketName(Bucket);
            resumeUploadRequest.setKey(Key);
            resumeUploadRequest.setEnableCheckpoint(true);
            resumeUploadRequest.setCheckpointPath(".");
            resumeUploadRequest.setFilePath(FilePath);
            resumeUploadRequest.setPartSize(PartSize);
            ResumeUploadResult result  =
s3Client.resumeUploadFile(resumeUploadRequest, (currentBytes, totalBytes) -> {
                    // 注意：这里将long类型转换为double进行除法，避免整数除法导致结果向下
取整
                    double percentage = ((double) currentBytes / totalBytes) *
100;
                    String processStr = String.format("\rUploaded: %10d / %10d
(%.2f%%)", currentBytes, totalBytes, percentage);
                    System.out.print(processStr);
                    System.out.flush();
                });
            System.out.print("ETag: " + result.getETag() + "\n");
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

删除对象

功能说明

将一个文件（Object）删除。

方法原型

`void deleteObject(DeleteObjectRequest deleteObjectRequest)`

参数说明

- DeleteObjectRequest

设定参数方法	是否必选	描述
DeleteObjectRequest(String bucketName, String key)	是	构造函数设置桶名和对象名
void setBucketName(String bucketName)	是	设置桶名
void setKey(String key)	是	设置对象名

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.DeleteObjectRequest;

import java.io.File;
import java.io.FileOutputStream;

public class deleteObject {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            DeleteObjectRequest request = new DeleteObjectRequest(Bucket, Key);
            s3Client.deleteObject(request);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

删除对象某个版本

功能说明

将一个对象的特定版本删除。

方法原型

`void deleteVersion(DeleteVersionRequest deleteVersionRequest )`

参数说明

- DeleteVersionRequest

设定参数方法	是否必选	描述
DeleteVersionRequest(String bucketName, String key, String versionId)	是	构造函数设置桶名、对象名、版本号
void setBucketName(String bucketName)	是	设置桶名
void setKey(String key)	是	设置对象名
void setBypassGovernanceRetention(boolean bypassGovernanceRetention	可选	设置是否绕过Governance模式锁的限制

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.Bucket;
import com.amazonaws.services.s3.model.DeleteVersionRequest;

public class deleteVersion {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
```

```
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            DeleteVersionRequest request = new DeleteVersionRequest(Bucket, Key,
"<your-versionId>");
            s3Client.deleteVersion(request);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

批量删除对象

功能说明

批量删除文件，最大支持单次删除 1000 个文件。对于返回结果，ZOS 提供 Verbose 和 Quiet 两种结果模式。Verbose 模式将返回每个 Object 的删除结果；Quiet 模式只返回报错的 Object 信息。

方法原型

`DeleteObjectsResult deleteObjects(DeleteObjectsRequest deleteObjectsRequest )`

参数说明

- DeleteObjectsRequest

设定参数方法	是否必选	描述
DeleteObjectRequest(String bucketName)	是	构造函数设置桶名
void setBucketName(String bucketName)	是	设置桶名
void setKeys(List<DeleteObjectsRequest.KeyVersion> keys)	是	设置对象名
void setBypassGovernanceRetention(boolean bypassGovernanceRetention)	可选	设置是否绕过GOVERNANCE对象锁的限制
void setQuiet(boolean quiet)	可选	是否开启quiet模式

- DeleteObjectsRequest.KeyVersion

设定参数方法	是否必选	描述
KeyVersion(String key)	可选	构造函数设置对象名
KeyVersion(String key, String version)	可选	构造函数设置对象名、版本号

返回结果说明

- DeleteObjectsResult

获取结果方法	描述
List<DeletedObject> getDeletedObjects()	获取已删除的对象列表

- DeletedObject

获取结果方法	描述
String getKey()	获取对象名
String getVersionId()	获取对象版本号

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
import com.amazonaws.services.s3.model.DeleteObjectsResult;
import com.amazonaws.services.s3.model.MultiObjectDeleteException;

import java.util.Arrays;
import java.util.List;

public class deleteObjects {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
```

```
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            DeleteObjectsRequest request = new DeleteObjectsRequest(Bucket);
            List<DeleteObjectsRequest.KeyVersion> keyVersions =
                    Arrays.asList(new DeleteObjectsRequest.KeyVersion(Key), new
DeleteObjectsRequest.KeyVersion(Key, "<your-versionId>"));
            request.setKeys(keyVersions);
            request.setQuiet(false);

            DeleteObjectsResult result = s3Client.deleteObjects(request);
            System.out.print("=====request success=====\n");
            result.getDeletedObjects().forEach((deletedObj)->
System.out.format("%s versionId: %s deleted\n",deletedObj.getKey(),
deletedObj.getVersionId()));
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

列出对象

功能说明

列出该 Bucekt 下部分或者所有Object，一次最多返回1000个。

方法原型

`ObjectListing listObjects(ListObjectsRequest listObjectsRequest )`

参数说明

- ListObjectsRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名
void setPrefix(String prefix)	可选	只列出特定前缀的对象
void setMaxKeys(Integer maxKeys)	可选	设置每次最多返回的对象数，默认1000
void setMarker(String marker)	可选	列举对象的起始位置，返回的对象列表将是对象名按照字典序排序后该参数以后的所有对象
void setDelimiter(String delimiter)	可选	对象名按照此标识符进行分组。通常与prefix参数搭配使用，如果指定了prefix，从prefix到第一次出现delimiter间具有相同字符串的对象名会被分成一组

返回结果说明

- ObjectListing

获取结果方法	描述
List<S3ObjectSummary> getObjectSummaries()	获取对象集合
boolean isTruncated()	获取是否返回了所有满足要求的Key
String getNextMarker()	下次列举对象请求的起始位置。如果本次没有返回全部结果，响应请求中将包含该元素，用于标明接下来请求的Marker值
int getMaxKeys()	获取请求每次最大返回数
String getBucketName()	获取桶名
String getPrefix()	获取前缀
String getDelimiter()	获取分组标识符

- S3ObjectSummary

获取结果方法	描述
String getETag()	获取ETag
String getKey()	获取对象名
Date getLastModified()	获取修改时间
Owner getOwner()	获取所有者
long getSize()	获取大小
String getStorageClass()	获取存储类型
String getType()	获取对象类型： "Appendable"，可追加写对象； "Symlink"，软链接对象； "Normal"，普通对象

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.ListObjectsRequest;
import com.amazonaws.services.s3.model.ObjectListing;

public class listObjects {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
```

```
            // 列出 Object
            ListObjectsRequest request = new ListObjectsRequest();
            request.setBucketName(Bucket);

            boolean istruncated = true;
            String marker = "";
            // 循环请求直到获取全部对象
            while(istruncated) {
                request.setMarker(marker);
                ObjectListing result = s3Client.listObjects(request);
                for (S3ObjectSummary summary : result.getObjectSummaries()){
                    System.out.println(summary.toString());
                }
                istruncated = result.isTruncated();
                marker = result.getNextMarker();
            }
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

列出对象V2

功能说明

列出该 Bucekt 下部分或者所有Object，一次最多返回1000个。

方法原型

`ListObjectsV2Result listObjectsV2(ListObjectsV2Request listObjectsV2Request)`

参数说明

- ListObjectsV2Request

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名
void setPrefix(String prefix)	可选	只列出特定前缀的对象
void setMaxKeys(Integer maxKeys)	可选	设置每次最多返回的对象数，默认1000
void setContinuationToken(String continuationToken)	可选	列举对象的起始位置，返回的对象列表将是对象名按照字典序排序后该参数以后的所有对象
void setDelimiter(String delimiter)	可选	对象名按照此标识符进行分组。通常与prefix参数搭配使用，如果指定了prefix，从prefix到第一次出现delimiter间具有相同字符串的对象名会被分成一组

返回结果说明

- ListObjectsV2Result

获取结果方法	描述
List<S3ObjectSummary> getObjectSummaries()	获取对象集合
boolean isTruncated()	获取是否返回了所有满足要求的Key
String getNextContinuationToken()	下次列举对象请求的起始位置。如果本次没有返回全部结果，响应请求中将包含该元素，用于标明接下来请求的token值
int getMaxKeys()	获取请求每次最大返回数
String getBucketName()	获取桶名
String getPrefix()	获取前缀
String getDelimiter()	获取分组标识符

- S3ObjectSummary

获取结果方法	描述
String getETag()	获取ETag
String getKey()	获取对象名
Date getLastModified()	获取修改时间
Owner getOwner()	获取所有者
long getSize()	获取大小
String getStorageClass()	获取存储类型
String getType()	获取对象类型： "Appendable"，可追加写对象； "Symlink"，软链接对象； "Normal"，普通对象

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

public class listObjects {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
```

```
            // 列出 Object
            ListObjectsV2Request request = new ListObjectsV2Request();
            request.setBucketName(Bucket);

            boolean istruncated = true;
            String token = "";
            // 循环请求直到获取全部对象
            while(istruncated) {
                request.setContinuationToken(token);
                ListObjectsV2Result result = s3Client.listObjectsV2(request);
                for (S3ObjectSummary summary : result.getObjectSummaries()){
                    System.out.println(summary.toString());
                }
                istruncated = result.isTruncated();
                token = result.getNextContinuationToken();
            }
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

列出版本

功能说明

列出该 Bucekt 下部分或者所有Object的版本，一次最多返回1000个。

方法原型

`VersionListing listVersions(ListVersionsRequest listVersionsRequest)`

参数说明

- ListVersionsRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名
void setPrefix(String prefix)	可选	只列出特定前缀的对象
void setMaxResults(Integer maxResults)	可选	设置每次最多返回的对象数，默认1000
void setKeyMarker(String keyMarker)	可选	列举多版本对象的起始位置，返回的对象列表将是对象名按照字典序排序后该参数以后的所有对象
void setVersionIdMarker(String versionIdMarker)	可选	与keyMarker配合使用，返回的对象列表将是对象名和版本号按照字典序排序后该参数以后的所有对象
void setDelimiter(String delimiter)	可选	对象名按照此标识符进行分组。通常与prefix参数搭配使用，如果指定了prefix，从prefix到第一次出现delimiter间具有相同字符串的对象名会被分成一组

返回结果说明

- VersionListing

获取结果方法	描述
List<S3VersionSummary> getVersionSummaries()	获取对象集合
boolean isTruncated()	获取是否返回了所有满足要求的版本
String getNextKeyMarker()	下次列举多版本对象请求的起始位置。如果本次没有返回全部结果，响应请求中将包含该元素，用于标明接下来请求的keyMarker值
String getNextVersionIdMarker()	下次列举多版本对象请求的起始位置，与nextKeyMarker配合使用。如果本次没有返回全部结果，响应请求中将包含该元素，用于标明接下来请求的versionIdMarker值。
int getMaxKeys()	获取请求每次最大返回数
String getBucketName()	获取桶名
String getPrefix()	获取前缀
String getDelimiter()	获取分组标识符

- S3VersionSummary

获取结果方法	描述
String getETag()	获取ETag
String getVersionId()	获取VersionId
String getKey()	获取对象名
Date getLastModified()	获取修改时间
Owner getOwner()	获取所有者
long getSize()	获取大小
String getStorageClass()	获取存储类型
boolean isDeleteMarker()	获取是否是删除标记
String getType()	获取对象类型： "Appendable"，可追加写对象； "Symlink"，软链接对象； "Normal"，普通对象

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.OutputStreamWriter;

public class listVersions {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
```

```
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            // 列出 Version
            ListVersionsRequest request = new ListVersionsRequest();
            request.setBucketName(Bucket);

            boolean istruncated = true;
            String keyMarker = "";
            String versionIdMarker = "";
            // 循环请求直到获取全部版本
            while(istruncated) {
                request.setVersionIdMarker(versionIdMarker);
                request.setKeyMarker(keyMarker);
                VersionListing result = s3Client.listVersions(request);
                for (S3VersionSummary summary : result.getVersionSummaries()){
                    System.out.println(summary.getKey() + ": " +
summary.getVersionId());
                }
                istruncated = result.isTruncated();
                keyMarker = result.getNextKeyMarker();
                versionIdMarker = result.getNextVersionIdMarker();
            }
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取对象元数据

功能说明

获取对应 Object 的元数据。

方法原型

`ObjectMetadata getObjectMetadata(GetObjectMetadataRequest request)`

参数说明

- GetObjectMetadataRequest

设定参数方法	是否必选	描述
GetObjectMetadataRequest(String bucketName, String key)	是	构造函数设置桶名、对象名
void setVersionId(String versionId)	可选	设置版本号

返回结果说明

- ObjectMetadata

获取结果方法	描述
Map<String, String> getUserMetadata()	获取用户自定义元数据
String getETag()	获取ETag
String getVersionId()	获取VersionId
Date getLastModified()	获取修改时间
String getContentType()	获取文件类型
long getContentLength()	获取大小
String getStorageClass()	获取存储类型

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.GetObjectMetadataRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;

public class getObjectMetadata {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);
```

```
            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            GetObjectMetadataRequest var = new
GetObjectMetadataRequest(Bucket,Key);

            ObjectMetadata meta = s3Client.getObjectMetadata(var);
            System.out.print("content-type: " + meta.getContentType() + "\n");
            System.out.print("content-length: " + meta.getContentLength()+
"\n");
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

设置对象ACL

功能说明

设置对象的ACL，控制对对象的访问权限。该操作需要用户具有WRITE_ACP权限。

设置方法有三种：

1. CannedAccessControlList，整体的权限设置；
2. AccessControlList ，根据给出的配置，具体的给某些用户授予某些权限；

方法原型

`void setObjectAcl(SetObjectAclRequest setObjectAclRequest)`

参数说明

- SetObjectAclRequest

设定参数方法	是否必选	描述
SetObjectAclRequest(String bucketName, String key, AccessControlList acl)	构造函数选其一	构造函数设置桶名、对象名、AccessControlList
SetObjectAclRequest(String bucketName, String key, CannedAccessControlList acl)	构造函数选其一	CannedAccessControlList.Private，私有（默认）；CannedAccessControlList.PublicRead，公共读，但只有对象所有者可以写入和更改ACL；CannedAccessControlList.BucketOwnerRead，对象所有者和存储桶所有者可以读取该对象，只有对象所有者可以写入和更改ACL；CannedAccessControlList.BucketOwnerFullControl，对象所有者和存储桶所有者可以读取和写入该对象以及更改ACL。
SetObjectAclRequest(String bucketName, String key, String versionId, AccessControlList acl)	构造函数选其一	构造函数设置桶名、对象名、版本号、AccessControlList
SetObjectAclRequest(String bucketName, String key, String versionId, CannedAccessControlList acl)	构造函数选其一	构造函数设置桶名、对象名、版本号、CannedAccessControlList CannedAccessControlList.Private，私有（默认）；CannedAccessControlList.PublicRead，公共读，但只有对象所有者可以写入和更改ACL；CannedAccessControlList.BucketOwnerRead，对象所有者和存储桶所有者可以读取该对象，只有对象所有者可以写入和更改ACL；CannedAccessControlList.BucketOwnerFullControl，对象所有者和存储桶所有者可以读取和写入该对象以及更改ACL。

- AccessControlList

设定参数方法	是否必选	描述
void setOwner(Owner owner)	是	设置桶所有者
void grantPermission(Grantee grantee, Permission permission)	是	设置授权角色及权限： Permission.FullControl，所有权限； Permission.Read，读权限； Permission.ReadACP，读ACL权限； Permission.Write，写权限； Permission.WriteAcp，写ACL权限。

- Owner

设定参数方法	是否必选	描述
void setId(String id)	是	设置ID
void setDisplayName(String name)	否	设置名字

- Grantee，接口类，两种实现：CanonicalGrantee、EmailAddressGrantee

设定参数方法	是否必选	描述
CanonicalGrantee(String identifier)	可选	构造函数，设置被授权用户id
EmailAddressGrantee(String emailAddress)	可选	构造函数，设置被授权用户邮箱

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import java.util.List;

public class setObjectAcl {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
```

```
    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            // 第一种方法
            SetObjectAclRequest req = new SetObjectAclRequest(Bucket, Key,
CannedAccessControlList.Private);
            // 第二种方法
//            AccessControlList acl = new AccessControlList();
//            Owner owner = new Owner("test-1", "test-1");
//            acl.setOwner(owner);
//            CanonicalGrantee canonicalGrantee = new CanonicalGrantee("test-
2");
//            acl.grantPermission(canonicalGrantee, Permission.Write);
//            SetObjectAclRequest req = new SetObjectAclRequest(Bucket, Key,
acl);

            s3Client.setObjectAcl(req);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取对象ACL

功能说明

获取指定Object的 ACL。该操作需要READ_ACP权限。

方法原型

`AccessControlList getObjectAcl(GetObjectAclRequest getObjectAclRequest)`

参数说明

- GetObjectAclRequest

设定参数方法	是否必选	描述
GetObjectAclRequest(String bucketName, String key)	构造函数选其一	构造函数设置桶名、对象名
GetObjectAclRequest(String bucketName, String key, String versionId)	构造函数选其一	构造函数设置桶名、对象名、版本号

返回结果说明

- AccessControlList

获取结果方法	描述
Owner getOwner()	获取桶Owner
List<Grant> getGrantsAsList()	获取授权列表

- Owner

设定参数方法	描述
String getId()	获取ID
String getDisplayName()	获取名字

- Grant

获取结果方法	描述
Grantee getGrantee()	获取授权用户信息
Permission getPermission()	获取权限类型: Permission.FullControl，所有权限； Permission.Read，读权限； Permission.ReadACP，读ACL权限； Permission.Write，写权限； Permission.WriteAcp，写ACL权限。

- Grantee

获取结果方法	描述
String getTypeIdentifier()	获取用户类型： id，id用户； emailAddress，邮箱用户； uri，组用户
String getIdentifier();	获取用户标识

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import java.util.List;

public class getObjectAcl {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            AccessControlList accessControlList = s3Client.getObjectAcl(Bucket,
Key);
            for (Grant grant: accessControlList.getGrantsAsList()) {
                System.out.println(grant.toString());
            }
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

设置对象标签

功能说明

将提供的标签集设置为存储桶中已存在的对象。标签是一个键值对。请注意，标签的最大数量限制为每个对象 10 个标签，key最大128字节，value最大256字节，value可以为空。

方法原型

```
void setObjectTagging(SetObjectTaggingRequest setObjectTaggingRequest )
```

参数说明

- SetObjectTaggingRequest

设定参数方法	是否必选	描述
SetObjectTaggingRequest(String bucketName, String key, ObjectTagging tagging)	构造函数选其一	构造函数设置桶名、对象名、标签集合
SetObjectTaggingRequest(String bucketName, String key, String versionId, ObjectTagging tagging)	构造函数选其一	构造函数设置桶名、对象名、版本号、标签集合
void setBucketName(String bucketName)	可选	设置桶名
void setKey(String key)	可选	设置对象名
void setVersionId(String versionId)	可选	设置版本号
setTagging(ObjectTagging tagging)	可选	设置标签

- ObjectTagging

设定参数方法	是否必选	描述
ObjectTagging(List<Tag> tagSet)	是	构造函数设置标签

- Tag

设定参数方法	是否必选	描述
Tag(String key, String value)	是	构造函数设置标签k、v

返回结果说明

无

代码示例

```
import java.util.ArrayList;
import java.util.List;

import com.amazonaws.ClientConfiguration;
```

```java
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ObjectTagging;
import com.amazonaws.services.s3.model.SetObjectTaggingRequest;
import com.amazonaws.services.s3.model.Tag;

public class setObjectTagging {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            List<Tag> tagSet = new ArrayList<Tag>();
            tagSet.add(new Tag("tagKey", "tagValue"));

            ObjectTagging tagging = new ObjectTagging(tagSet);
            SetObjectTaggingRequest request = new
SetObjectTaggingRequest(Bucket, Key, tagging);

            s3Client.setObjectTagging(request);
            System.out.println("=====Put Object Tagging success!=====");
        }catch (Exception e) {
            System.out.print(e.getMessage());
        }
    }
}
```

获取对象标签

功能说明

返回对象的标签集。

方法原型

`GetObjectTaggingResult getObjectTagging(GetObjectTaggingRequest GetObjectTaggingRequest)`

参数说明

- GetObjectTaggingRequest

设定参数方法	是否必选	描述
GetObjectTaggingRequest(String bucketName, String key)	构造函数选其一	构造函数设置桶名、对象名
GetObjectTaggingRequest(String bucketName, String key, String versionId)	构造函数选其一	构造函数设置桶名、对象名、版本号
void setBucketName(String bucketName)	可选	设置桶名
void setKey(String key)	可选	设置对象名
void setVersionId(String versionId)	可选	设置版本号

返回结果说明

- GetObjectTaggingResult

获取结果方法	描述
List<Tag> getTagSet();	获取标签集

- Tag

设定参数方法	描述
String getKey();	获取标签key
String getValue()	获取标签value

代码示例

```
import java.util.List;
import com.amazonaws.ClientConfiguration;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

```
import com.amazonaws.services.s3.model.GetObjectTaggingRequest;
import com.amazonaws.services.s3.model.GetObjectTaggingResult;
import com.amazonaws.services.s3.model.Tag;

public class getObjectTagging {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            GetObjectTaggingRequest request = new
GetObjectTaggingRequest(Bucket, Key);
            GetObjectTaggingResult result = s3Client.getObjectTagging(request);

            if (result == null) {
                System.out.println("There are not tagging for the object: " +
Key);
            } else {
                System.out.println("=====Get object Tagging=====");
                System.out.println(result.getVersionId());
                List<Tag> tagSet = result.getTagSet();

                for(int i=0;i<tagSet.size();i++) {
                    System.out.println("Key: " + tagSet.get(i).getKey());
                    System.out.println("Value: " + tagSet.get(i).getValue());
                }
                System.out.println("============================");
            }
        }catch (Exception e) {
            System.out.print(e.getMessage());
        }
    }
}
```

删除对象标签

功能说明

从指定的对象中删除整个标记集。

方法原型

`void deleteObjectTagging(DeleteObjectTaggingRequest deleteObjectTaggingRequest)`

参数说明

- DeleteObjectTaggingRequest

设定参数方法	是否必选	描述
DeleteObjectTaggingRequest(String bucketName, String key)	是	构造函数设置桶名、对象名
void setBucketName(String bucketName)	可选	设置桶名
void setKey(String key)	可选	设置对象名
void setVersionId(String versionId)	可选	设置版本号

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.DeleteObjectTaggingRequest;

public class deleteObjectTagging {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);
```

```
            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            DeleteObjectTaggingRequest request = new
DeleteObjectTaggingRequest(Bucket, Key);
            s3Client.deleteObjectTagging(request);
            System.out.println("=====Delete Object Tagging success!=====");
        }catch (Exception e) {
            System.out.print(e.getMessage());
        }
    }
}
```

设置对象合规保留配置

功能说明

在指定的对象上增加合规保留配置，桶在创建时需开启对象锁。

方法原型

`SetObjectRetentionResult setObjectRetention(SetObjectRetentionRequest setObjectRetentionRequest)`

参数说明

- SetObjectRetentionRequest

设定参数方法	是否必选	描述
void setBucketName(String bucket)	是	设置桶名
void setKey(String key)	是	设置对象名
void setVersionId(String versionId)	可选	设置版本号
void setRetention(ObjectLockRetention retention)	可选	设置合规保留规则
void setBypassGovernanceRetention(boolean bypassGovernanceRetention)	可选	设置是否忽视Governance模式

- ObjectLockRetention

设定参数方法	是否必选	描述
void setRetainUntilDate(Date retainUntilDate)	是	设置到期时间
void setMode(String mode)	是	设置模式： "COMPLIANCE"，设置后不允许修改； "GOVERNANCE"，设置后允许修改。

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import java.util.*;
import com.amazonaws.services.s3.model.*;

public class setObjectRetention {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
```

```
            // 年份从1900年开始计数，月份从0开始。2024年6月18日9点11分5秒。
            Date date = new Date(124, 05, 18, 9, 11, 5);
            ObjectLockRetention retention = new ObjectLockRetention();
            retention.setMode("GOVERNANCE");
            retention.setRetainUntilDate(date);

            SetObjectRetentionRequest req = new SetObjectRetentionRequest();
            req.setBucketName(Bucket);
            req.setKey(Key);
            req.setRetention(retention);
            req.setBypassGovernanceRetention(true);

            s3Client.setObjectRetention(req);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取对象合规保留配置

功能说明

获取对象的合规保留配置。

方法原型

`GetObjectRetentionResult getObjectRetention(GetObjectRetentionRequest getObjectRetentionRequest)`

参数说明

- GetObjectRetentionRequest

设定参数方法	是否必选	描述
void setBucketName(String bucket)	是	设置桶名
void setKey(String key)	是	设置对象名
void setVersionId(String versionId)	可选	设置版本号

返回结果说明

- GetObjectRetentionResult

获取结果方法	描述
ObjectLockRetention getRetention()	获取合规保留配置

- ObjectLockRetention

获取结果方法	描述
String getMode()	获取保留模式
Date getRetainUntilDate()	获取到期时间

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import java.util.*;
import com.amazonaws.services.s3.model.*;

public class getObjectRetention {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            GetObjectRetentionRequest req = new GetObjectRetentionRequest();
            req.setBucketName(Bucket);
            req.setKey(Key);

            GetObjectRetentionResult resp = s3Client.getObjectRetention(req);
            System.out.print("=====request success=====\n");
            ObjectLockRetention retention = resp.getRetention();
            String mode = retention.getMode();
            System.out.println(mode);
            Date date = retention.getRetainUntilDate();
            System.out.println(date);
```

```
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

设置对象依法保留配置

功能说明

在指定的对象上增加依法保留配置，桶在创建时需开启对象锁。该配置仅可开启或关闭，没有时限的设置，如果同时设置了合规保留配置，则以合规保留配置为准，**建议使用合规保留。**

方法原型

`SetObjectLegalHoldResult setObjectLegalHold(SetObjectLegalHoldRequest request)`

参数说明

- SetObjectLegalHoldRequest

设定参数方法	是否必选	描述
void setBucketName(String bucket)	是	设置桶名
void setKey(String key)	是	设置对象名
void setVersionId(String versionId)	可选	设置版本号
void setLegalHold(ObjectLockLegalHold legalHold)	可选	设置依法保留规则

- ObjectLockLegalHold

设定参数方法	是否必选	描述
void setStatus(String status)	是	设置是否开启，"ON"或"OFF"

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import java.util.*;
import com.amazonaws.services.s3.model.*;

public class setObjectLegalHold {
    public static String ACCESS_KEY = "<your-access-key>";
```

```
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            ObjectLockLegalHold legal_hold = new ObjectLockLegalHold();
            legal_hold.setStatus("ON");

            SetObjectLegalHoldRequest Req = new SetObjectLegalHoldRequest();
            Req.setBucketName(Bucket);
            Req.setKey(Key);
            Req.setLegalHold(legal_hold);
//            Req.setVersionId("");

            SetObjectLegalHoldResult resp = s3Client.setObjectLegalHold(Req);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取对象依法保留配置

功能说明

获取对象的依法保留配置。

方法原型

`GetObjectLegalHoldResult getObjectLegalHold(GetObjectLegalHoldRequest request)`

参数说明

- GetObjectLegalHoldRequest

设定参数方法	是否必选	描述
void setBucketName(String bucket)	是	设置桶名
void setKey(String key)	是	设置对象名
void setVersionId(String versionId)	可选	设置版本号

返回结果说明

- GetObjectLegalHoldResult

获取结果方法	描述
ObjectLockLegalHold getLegalHold()	获取依法保留配置

- ObjectLockLegalHold

获取结果方法	描述
String getStatus()	获取依法保留开启状态

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import java.util.*;
import com.amazonaws.services.s3.model.*;

public class getObjectLegalHold {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
```

```
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            GetObjectLegalHoldRequest Req = new GetObjectLegalHoldRequest();
            Req.setBucketName(Bucket);
            Req.setKey(Key);
//            Req.setVersionId("");

            GetObjectLegalHoldResult resp = s3Client.getObjectLegalHold(Req);
            System.out.print("=====request success=====\n");
            ObjectLockLegalHold legal_hold = resp.getLegalHold();
            String status = legal_hold.getStatus();
            System.out.println(status);
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

复制对象

功能说明

将一个文件从源路径复制到目标路径。

方法原型

`CopyObjectResult copyObject(CopyObjectRequest copyObjectRequest)`

参数说明

- CopyObjectRequest

设定参数方法	是否必选	描述
void setDestinationBucketName(String destinationBucketName)	是	设置目的桶名
void setDestinationKey(String destinationKey)	是	设置目的对象名
void setSourceBucketName(String sourceBucketName)	是	设置源桶名
void setSourceKey(String sourceKey)	是	设置源对象名
void setSourceVersionId(String sourceVersionId)	可选	设置源对象版本号
void setMatchingETagConstraints(List<String> eTagList)	可选	仅当指定的Etag和Copy Source指定的object的Etag匹配时才复制
void setNonmatchingETagConstraints(List<String> eTagList)	可选	仅当指定的Etag和Copy Source指定的object的Etag不匹配时才复制
void setModifiedSinceConstraint(Date date)	可选	仅当CopySource在指定时间后更新过才复制
void setUnmodifiedSinceConstraint(Date date)	可选	仅当CopySource在指定时间后未更新过才复制
void setCannedAccessControlList(CannedAccessControlList cannedACL)	可选	设置对象CannedAccessControlList，可选项： CannedAccessControlList.Private，私有（默认）； CannedAccessControlList.PublicRead，公共读，但只有对象所有者可以写入和更改ACL； CannedAccessControlList.BucketOwnerRead，对象所有者和存储桶所有者可以读取该对象，只有对象所有者可以写入和更改ACL； CannedAccessControlList.BucketOwnerFullControl，对象所有者和存储桶所有者可以读取和写入该对象以及更改ACL。
void setAccessControlList(AccessControlList accessControlList)	可选	设置对象AccessControlList。
void setNewObjectMetadata(ObjectMetadata newObjectMetadata)	可选	设置对象元数据
void setMetadataDirective(String metadataDirective)	可选	是否沿用元数据： "COPY"，复制源对象的元数据； "REPLACE"，替换新的元数据。
void setNewObjectTagging(ObjectTagging newObectTagging)	可选	设置标签
void setObjectLockMode(String objectLockMode)	可选	设置合规保留模式： "COMPLIANCE"，设置后不允许修改； "GOVERNANCE"，设置后允许修改。
void setObjectLockRetainUntilDate(Date objectLockRetainUntilDate)	可选	设置保留到期时间
void setStorageClass(StorageClass storageClass)	可选	设置桶存储类型，可选项： StorageClass.Standard，标准存储； StorageClass.StandardInfrequentAccess，低频存储； StorageClass.Glacier，归档存储。

- AccessControlList

设定参数方法	是否必选	描述
void setOwner(Owner owner)	是	设置桶所有者
void grantPermission(Grantee grantee, Permission permission)	是	设置授权角色及权限： Permission.FullControl，所有权限； Permission.Read，读权限； Permission.ReadACP，读ACL权限； Permission.Write，写权限； Permission.WriteAcp，写ACL权限。

- Owner

设定参数方法	是否必选	描述
void setId(String id)	是	设置ID
void setDisplayName(String name)	否	设置名字

- Grantee，接口类，两种实现：CanonicalGrantee、EmailAddressGrantee

设定参数方法	是否必选	描述
CanonicalGrantee(String identifier)	可选	构造函数，设置被授权用户id
EmailAddressGrantee(String emailAddress)	可选	构造函数，设置被授权用户邮箱

- ObjectTagging

设定参数方法	是否必选	描述
ObjectTagging(List<Tag> tagSet)	是	构造函数设置标签

- Tag

设定参数方法	是否必选	描述
Tag(String key, String value)	是	构造函数设置标签k、v

- ObjectMetadata

设定参数方法	是否必选	描述
void addUserMetadata(String key, String value)	可选	添加用户元数据

返回结果说明

- CopyObjectResult

获取结果方法	描述
String getETag()	获取对象ETag
Date getLastModifiedDate()	获取对象最后修改时间
String getVersionId()	获取复制后对象的版本号

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CopyObjectRequest;
import com.amazonaws.services.s3.model.CopyObjectResult;

import java.io.File;
import java.io.FileOutputStream;

public class copyObject {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            CopyObjectRequest request = new CopyObjectRequest();
            request.setDestinationBucketName("bucket2");
```

```
            request.setDestinationKey("key2");
            request.setSourceBucketName(Bucket);
            request.setSourceKey(Key);
            CopyObjectResult result = s3Client.copyObject(request);
            System.out.print("=====request success=====\n");

            System.out.format("object copyed, etag: %s\n", result.getETag());

        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

生成预签名链接

功能说明

生成一个临时的预签名的Url，没有权限访问集群的用户可以通过该Url访问集群，包括上传文件、下载文件等等。

方法原型

```
URL generatePresignedUrl(GeneratePresignedUrlRequest generatePresignedUrlRequest )
```

参数说明

- GeneratePresignedUrlRequest

设定参数方法	是否必选	描述
GeneratePresignedUrlRequest(String bucketName, String key, HttpMethod method)	是	构造函数设置桶名、对象名、HTTP方法
void setBucketName(String bucketName)	可选	设置桶名
void setKey(String key)	可选	设置对象名
void setMethod(HttpMethod method)	可选	设置HTTP方法： HttpMethod.GET HttpMethod.POST HttpMethod.PUT HttpMethod.DELETE HttpMethod.HEAD HttpMethod.PATCH
void setExpiration(Date expiration)	可选	设置有效时间，最多7天

返回结果说明

预签名链接字符串。

代码示例

- 仅创建链接

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.HttpMethod;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;

import java.net.URL;
import java.time.Instant;
import java.util.Date;

public class generatePresignedUrl {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            //获取预签名Url
            GeneratePresignedUrlRequest var = new
GeneratePresignedUrlRequest(Bucket, Key);

            Date expiration = new Date();
            long expTimeMillis = Instant.now().toEpochMilli();
            expTimeMillis += 1000 * 60 * 60;
            // 设置有效期60分钟
            expiration.setTime(expTimeMillis);
            var.setExpiration(expiration);
            var.setMethod(HttpMethod.GET);
            URL url = s3Client.generatePresignedUrl(var);
```

```
            System.out.print(url.toString() + "\n");
            System.out.print(url.getProtocol() + "\n");
            System.out.print(url.getHost() + "\n");
            System.out.print(url.getPort() + "\n");
            System.out.print(url.getPath() + "\n");
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

- 创建上传链接并上传对象

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.HttpMethod;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import com.amazonaws.services.s3.model.S3Object;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Date;

public class uploadPresign {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String filePath = "<your-filePath>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
```

```
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            Date expiration = new Date();
            long expTimeMills = expiration.getTime();
            expTimeMills += 1000 * 60 * 60;
            expiration.setTime(expTimeMills);
            //生成预签名链接
            System.out.println("Generating pre-signed URL.");
            GeneratePresignedUrlRequest generatePresignedUrlRequest =
                    new GeneratePresignedUrlRequest(Bucket, Key)
                            .withMethod(HttpMethod.PUT)
                            .withExpiration(expiration);
            URL url =
s3Client.generatePresignedUrl(generatePresignedUrlRequest);
            System.out.println("The pre-signed URL is  : " + url.toString());

            //创建HTTP连接并上传对象
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setDoOutput(true);
            connection.setRequestMethod("PUT");
            //设置acl，"private"|"public-read"
            //connection.setRequestProperty("x-amz-acl", "private");
            FileInputStream in = new FileInputStream(filePath);
            OutputStream out = connection.getOutputStream();
            int len;
            int bufSize = 50 * 1024 * 1024;
            byte[] bs = new byte[bufSize];
            if ((len = in.read(bs)) != -1) {
                out.write(bs, 0, len);
            }
            in.close();
            out.close();
            System.out.println("HTTP response code: " +
connection.getResponseCode());
            connection.disconnect();
            S3Object object = s3Client.getObject(Bucket, Key);
            System.out.println("Object " + object.getKey() + " created in bucket
" + object.getBucketName());
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

- 创建下载链接并下载对象

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.HttpMethod;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
```

```java
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import com.amazonaws.services.s3.model.S3Object;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Date;

public class downloadPresign {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String filePath = "<your-filePath>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            Date expiration = new Date();
            long expTimeMills = expiration.getTime();
            expTimeMills += 1000 * 60 * 60;
            expiration.setTime(expTimeMills);
            //生成预签名链接
            System.out.println("Generating pre-signed URL.");
            GeneratePresignedUrlRequest generatePresignedUrlRequest =
                    new GeneratePresignedUrlRequest(Bucket, Key)
                            .withMethod(HttpMethod.GET)
                            .withExpiration(expiration);
            URL url =
s3Client.generatePresignedUrl(generatePresignedUrlRequest);
            System.out.println("The pre-signed URL is  : " + url.toString());
            //创建HTTP连接并下载对象
```

```
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            FileOutputStream out = new FileOutputStream(filePath);
            InputStream in = connection.getInputStream();
            int len;
            int bufSize = 50 * 1024 * 1024;
            byte[] bs = new byte[bufSize];
            if ((len = in.read(bs)) != -1) {
                out.write(bs, 0, len);
            }
            in.close();
            out.close();
            System.out.println("HTTP response code: " +
connection.getResponseCode());
            connection.disconnect();
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

上传软链接

功能说明

使用接口对目标对象创建软链接，可以通过访问软链接来获取目标对象。删除软链接时，仅删除软链接本身，不会影响指
向的目标对象。

方法原型

`PutSymlinkResult putSymlink(PutSymlinkRequest request)`

参数说明

- PutSymlinkRequest

设定参数方法	是否必选	描述
PutSymlinkRequest(String bucket, String key, String symlinkTarget)	是	构造函数设置桶名、软链接名、链接的对象名
void setForbidOverwrite(boolean forbidOverwrite)	可选	设置是否禁止重名覆盖，默认不禁止。

返回结果说明

- PutSymlinkResult

获取结果方法	描述
String getVersionId()	获取软链接版本号

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

public class putSymlink {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String Target = "<your-symlinkTarget>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            PutSymlinkRequest request = new PutSymlinkRequest(Bucket, Key,
Target);
            PutSymlinkResult result = s3Client.putSymlink(request);
            System.out.format("VersionId: %s\n",result.getVersionId());
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

获取软链接

功能说明

获取软链接，得到版本号和链接所指向的对象名。

方法原型

`GetSymlinkResult getSymlink(GetSymlinkRequest request)`

参数说明

- GetSymlinkRequest

设定参数方法	是否必选	描述
GetSymlinkRequest(String bucket, String key)	是	构造函数设置桶名、对象名
void setVersionId(String versionId)	可选	设置版本号

返回结果说明

- GetSymlinkResult

获取结果方法	描述
String getVersionId()	获取版本号
String getSymlinkTarget()	获取链接的对象名

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

public class getSymlink {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);
```

```
            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            GetSymlinkRequest request = new GetSymlinkRequest(Bucket, Key);
            GetSymlinkResult result = s3Client.getSymlink(request);
            System.out.format("VersionId: %s\n",result.getVersionId());
            System.out.format("SymlinkTarget: %s\n",result.getSymlinkTarget());
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

解冻对象

功能说明

用于解冻归档对象，归档对象无法直接读取，需要先解冻。解冻操作会生成一份带有生存期的标准类型副本数据，副本数据可设置有效期1~31天。

方法原型

`RestoreObjectResult restoreObjectV2(RestoreObjectRequest request)`

参数说明

- RestoreObjectRequest

设定参数方法	是否必选	描述
RestoreObjectRequest(String bucketName, String key)	是	构造函数设置桶名、对象名
void setExpirationInDays(int expirationInDays)	可选	设置有效时间（单位/天，默认1天）
void setVersionId(String versionId)	可选	设置版本号

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.RestoreObjectRequest;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.RestoreObjectResult;

public class restoreObjectV2 {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            RestoreObjectRequest request = new RestoreObjectRequest(Bucket,
Key);
            request.setExpirationInDays(2);
            RestoreObjectResult result = s3Client.restoreObjectV2(request);
            System.out.print("=====request success=====\n");

            // 确认解冻进度（）
            ObjectMetadata meta = s3Client.getObjectMetadata(Bucket, Key);
            Boolean restoreFlag = meta.getOngoingRestore();
            System.out.format("Restoration status: %s.\n",
                    restoreFlag ? "in progress" : "not in progress (finished or
failed)");
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
```

```
            System.out.print(e.getMessage());
        }
    }
}
```

分段上传

创建分段上传

功能说明

请求实现初始化分片上传，成功执行此请求以后会返回 Upload ID 用于后续的上传分段请求。

方法原型

`InitiateMultipartUploadResult initiateMultipartUpload(InitiateMultipartUploadRequest initiateMultipartUploadRequest)`

参数说明

- initiateMultipartUploadRequest

设定参数方法	是否必选	描述
InitiateMultipartUploadRequest(String bucketName, String key)	是	构造函数设置桶名、对象名
void setBucketName(String bucketName)	是	设置桶名
void setKey(String key)	是	设置对象名
void setStorageClass(StorageClass storageClass)	可选	设置桶存储类型，可选项： StorageClass.Standard，标准存储； StorageClass.StandardInfrequentAccess，低频存储； StorageClass.Glacier，归档存储。
void setTagging(ObjectTagging tagging)	可选	设置标签
void setCannedAcl(CannedAccessControlList cannedAcl)	可选	设置对象CannedAccessControlList ，可选项： CannedAccessControlList.Private，私有（默认）； CannedAccessControlList.PublicRead，公共读，但只有对象所有者可以写入和更改ACL； CannedAccessControlList.BucketOwnerRead，对象所有者和存储桶所有者可以读取该对象，只有对象所有者可以写入和更改ACL； CannedAccessControlList.BucketOwnerFullControl，对象所有者和存储桶所有者可以读取和写入该对象以及更改ACL。
void setAccessControlList(AccessControlList accessControlList)	可选	设置对象AccessControlList。
void setObjectMetadata(ObjectMetadata objectMetadata)	可选	设置对象元数据
void setObjectLockMode(String objectLockMode)	可选	设置对象合规保留模式： "COMPLIANCE"，设置后不允许修改； "GOVERNANCE"，设置后允许修改。
void setObjectLockRetainUntilDate(Date objectLockRetainUntilDate)	可选	设置对象过期时间

- AccessControlList

设定参数方法	是否必选	描述
void setOwner(Owner owner)	是	设置桶所有者
void grantPermission(Grantee grantee, Permission permission)	是	设置授权角色及权限： Permission.FullControl，所有权限； Permission.Read，读权限； Permission.ReadACP，读ACL权限； Permission.Write，写权限； Permission.WriteAcp，写ACL权限。

- Owner

设定参数方法	是否必选	描述
void setId(String id)	是	设置ID
void setDisplayName(String name)	否	设置名字

- Grantee，接口类，两种实现：CanonicalGrantee、EmailAddressGrantee

设定参数方法	是否必选	描述
CanonicalGrantee(String identifier)	可选	构造函数，设置被授权用户id
EmailAddressGrantee(String emailAddress)	可选	构造函数，设置被授权用户邮箱

- ObjectTagging

设定参数方法	是否必选	描述
ObjectTagging(List<Tag> tagSet)	是	构造函数设置标签

- Tag

设定参数方法	是否必选	描述
Tag(String key, String value)	是	构造函数设置标签k、v

- ObjectMetadata

设定参数方法	是否必选	描述
void addUserMetadata(String key, String value)	可选	添加用户元数据

返回结果说明

- InitiateMultipartUploadResult

获取结果方法	描述
String getUploadId()	获取upload id

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.InitiateMultipartUploadRequest;
import com.amazonaws.services.s3.model.InitiateMultipartUploadResult;
```

```
public class initiateMultipartUpload {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            //初始化分段上传
            InitiateMultipartUploadRequest var = new
InitiateMultipartUploadRequest(Bucket, Key);
            InitiateMultipartUploadResult result =
s3Client.initiateMultipartUpload(var);
            System.out.print("Upload Id: " + result.getUploadId() + "\n");
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

上传分段

功能说明

求实现在初始化以后的分块上传，支持的块的数量为 1 到 10000，除了最后一块，其他每块大小都必须**大于或等于5M**。在每次请求 Upload Part 时，需要携带 partNumber 和 uploadID，partNumber 为块的编号，支持乱序上传。

方法原型

`UploadPartResult uploadPart(UploadPartRequest uploadPartRequest)`

参数说明

- UploadPartRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名
void setKey(String key)	是	设置对象名
void setUploadId(String uploadId)	是	设置Upload ID
void setFile(File var)	与数据流选其一	设置上次文件
void setInputStream(InputStream var)	与文件输入选其一	设置上传数据流
void setPartNumber(int partNumber)	是	设置分段号，从1开始
void setPartSize(long partSize)	是	设置分段大小

返回结果说明

- UploadPartResult

获取结果方法	描述
String getETag()	获取分段ETag

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.InputStream;

public class uploadPart {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String UploadId = "<your-uploadId>";
    public static String filePath = "<your-path>";

    public static void main(String[] args) {
```

```
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            //上传分段数据
            File file = new File(FilePath);
            int partNumber = 1;         //初始分段号
            long filePosition = 0;
            long fileLength = file.length();
            long partSize = 5 * 1024 * 1024;    //分段大小（字节）,大于5M
            UploadPartRequest var = new UploadPartRequest();
            var.setUploadId(UploadId);
            var.setBucketName(Bucket);
            var.setKey(Key);
            var.setFile(file);
            while (filePosition < fileLength) {
                long actualSize = Math.min(partSize, fileLength - filePosition);
                var.setPartNumber(partNumber);
                var.setPartSize(actualSize);
                var.setFileOffset(filePosition);
                if (partSize >= fileLength - filePosition){
                    var.setLastPart(true);
                }

                UploadPartResult res = s3Client.uploadPart(var);
                System.out.println("part" + partNumber + " upload success, ETag:
" + res.getETag());

                filePosition += actualSize;
                partNumber++;
            }
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

完成分段上传

功能说明

完成整个分块上传。当您已经使用 Upload Parts 上传所有块以后，你可以用该 API 完成上传。在使用该 API 时，您必须在 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。

方法原型

`CompleteMultipartUploadResult completeMultipartUpload(CompleteMultipartUploadRequest completeMultipartUploadRequest)`

参数说明

- CompleteMultipartUploadRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketName)	是	设置桶名
void setKey(String key)	是	设置对象名
void setUploadId(String uploadId)	是	设置upload id
void setPartETags(List<PartETag> partETags)	是	设置各分段信息

- PartETag

设定参数方法	是否必选	描述
PartETag(int partNumber, String eTag)	是	构造函数设置分段号和ETag
void setPartNumber(int partNumber)	可选	设置分段号
void setETag(String eTag)	可选	设置分段Etag

返回结果说明

- CompleteMultipartUploadResult

获取结果方法	描述
String getETag()	获取整个文件Etag
String getVersionId()	获取整个文件版本号

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

```
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import java.util.ArrayList;
import java.util.List;

public class completeMultipartUpload {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String UploadId = "<your-uploadId>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            //结束分段上传
            CompleteMultipartUploadRequest var = new
CompleteMultipartUploadRequest();
            var.setBucketName(Bucket);
            var.setKey(Key);
            var.setUploadId(UploadId);
            List<PartETag> partETags = new ArrayList();
            ListPartsRequest listPartsRequest = new ListPartsRequest(Bucket,
Key, UploadId);
            PartListing parts = s3Client.listParts(listPartsRequest);
            List<PartSummary> summary = parts.getParts();
            for(PartSummary T:summary) {
                partETags.add(new PartETag(T.getPartNumber(), T.getETag()));
            }
            System.out.println(partETags.size());
            var.setPartETags(partETags);
            CompleteMultipartUploadResult result =
s3Client.completeMultipartUpload(var);
            System.out.print("ETag: " + result.getETag() + "\n");
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
```

```
}
```

列出分段

功能说明

用来查询特定分段上传中的已上传的分段的信息。

方法原型

`PartListing listParts(ListPartsRequest listPartsRequest)`

参数说明

- ListPartsRequest

设定参数方法	是否必选	描述
ListPartsRequest(String bucketName, String key, String uploadId)	是	构造函数设置桶名、对象名、uploadId
void setBucketName(String bucketName)	可选	设置桶名
void setKey(String key)	可选	设置对象名
void setUploadId(String uploadId)	可选	设置upload id
void setMaxParts(int maxParts)	可选	设置每次返回的最多分段数
void setPartNumberMarker(Integer partNumberMarker)	可选	列举分段的起始位置，返回的分段列表是分段号大于此参数的分段

返回结果说明

- PartListing

获取结果方法	描述
List<PartSummary> getParts()	获取分段信息
Integer getPartNumberMarker()	获取本次请求的起始分段位置
boolean isTruncated()	是否返回全部分段
Integer getNextPartNumberMarker()	下次列举分段请求的起始位置。如果本次没有返回全部结果，响应请求中将包含该元素，用于标明接下来请求的分段标记
Integer getMaxParts()	获取当前list返回的最大分段数目
String getStorageClass()	获取文件的存储类型

- PartSummary

获取结果方法	描述
int getPartNumber()	获取当前分段的分段号
String getETag()	获取该分段的Etag
long getSize()	获取当前分段的大小，单位字节

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;
import java.util.List;

public class listParts {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String UploadId = "<your-uploadId>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);
```

```
            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            //list分段信息
            ListPartsRequest var = new ListPartsRequest(Bucket, Key, UploadId);
            PartListing result = s3Client.listParts(var);
            List<PartSummary> summary = result.getParts();
            for(PartSummary T:summary) {
                System.out.println("Part" + T.getPartNumber());
                System.out.println("ETag: " + T.getETag());
                System.out.println("Size: " + T.getSize());
            }
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

列出分段上传

功能说明

查询正在进行中的分段上传，也就是已经 Created但是还没有Aborted或者Completed的分段上传数据，单次最多列出 1000 个正在进行中的分段上传。

方法原型

`MultipartUploadListing listMultipartUploads(ListMultipartUploadsRequest listMultipartUploadsRequest)`

参数说明

- ListMultipartUploadsRequest

设定参数方法	是否必选	描述
ListMultipartUploadsRequest(String bucketName)	是	构造函数设置桶名
void setBucketName(String bucketName)	是	设置桶名
void setPrefix(String prefix)	可选	只有以Prefix为开头的Key的分段上传数据才会返回
void setMaxUploads(Integer maxUploads)	可选	单次最多返回的分段上传数据，大小是1-1000
void setKeyMarker(String keyMarker)	可选	列举分段上传的起始位置，返回的对象列表将是对象名按照字典序排序后该参数以后的所有对象
void setUploadIdMarker(String uploadIdMarker)	可选	与keyMarker配合使用，返回的对象列表将是对象名和uploadId按照字典序排序后该参数以后的所有对象
void setDelimiter(String delimiter)	可选	对象名按照此标识符进行分组。通常与prefix参数搭配使用，如果指定了prefix，从prefix到第一次出现delimiter间具有相同字符串的对象名会被分成一组

返回结果说明

- MultipartUploadListing

获取结果方法	描述
List<MultipartUpload> getMultipartUploads()	获取分段上传数据
boolean isTruncated()	获取是否返回了所有满足要求的分段上传
String getNextKeyMarker()	下次列举分段上传请求的起始位置。如果本次没有返回全部结果，响应请求中将包含该元素，用于标明接下来请求的keyMarker值
String getNextUploadIdMarker()	下次列举分段上传请求的起始位置，与nextKeyMarker配合使用。如果本次没有返回全部结果，响应请求中将包含该元素，用于标明接下来请求的uploadIdMarker值。
String getPrefix()	获取请求设置的前缀
String getDelimiter()	获取请求设置的分隔符

- MultipartUpload

获取结果方法	描述
String getUploadId()	获取分段上传uploadId
String getKey()	获取文件key
String getStorageClass()	获取文件存储类型
Owner getOwner()	获取分段上传所属用户
Date getInitiated()	获取分段上传的初始化时间

- Owner

设定参数方法	描述
String getId()	获取ID
String getDisplayName()	获取名字

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;
import java.util.List;

public class listMultipartUploads {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
```

```
            System.out.print("=====connect success=====\n");
            //获取分段上传信息
            ListMultipartUploadsRequest var = new
ListMultipartUploadsRequest(Bucket);
            MultipartUploadListing result = s3Client.listMultipartUploads(var);

            List<MultipartUpload> summary = result.getMultipartUploads();
            for(MultipartUpload T:summary) {
                System.out.print(T.getUploadId() + "\n");
                System.out.print(T.getKey() + "\n");
            }
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

取消分段上传

功能说明

舍弃一个分块上传并删除已上传的块。当您调用 Abort Multipart Upload 时，如果有正在使用这个 Upload Parts 上传块的请求，则 Upload Parts 会返回失败。

方法原型

`void abortMultipartUpload(AbortMultipartUploadRequest abortMultipartUploadRequest)`

参数说明

- AbortMultipartUploadRequest

设定参数方法	是否必选	描述
AbortMultipartUploadRequest(String bucketName, String key, String uploadId)	是	构造函数设置桶名、对象名、uploadId

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;
```

```
public class abortMultipartUpload {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String UploadId = "<your-uploadId>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            //结束分段上传
            AbortMultipartUploadRequest  var = new AbortMultipartUploadRequest
(Bucket,Key,UploadId);
            s3Client.abortMultipartUpload(var);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

分段复制

功能说明

请求适用于将大文件分段复制到目标路径（桶）。

方法原型

`CopyPartResult copyPart(CopyPartRequest copyPartRequest )`

参数说明

- CopyPartRequest

设定参数方法	是否必选	描述
void setDestinationBucketName(String destinationBucketName)	是	设置目的桶名
void setDestinationKey(String destinationKey)	是	设置目的对象名
void setSourceBucketName(String sourceBucketName)	是	设置源桶名
void setSourceKey(String sourceKey)	是	设置源对象名
void setFirstByte(Long firstByte)	是	设置当前分段起始字节
void setLastByte(Long lastByte)	是	设置当前分段结束字节
void setPartNumber(int partNumber)	是	设置分段编号
void setUploadId(String uploadId)	是	设置uploadId
void setSourceVersionId(String sourceVersionId)	可选	设置源对象版本号
void setMatchingETagConstraints(List<String> eTagList)	可选	仅当指定的Etag和Copy Source指定的object的Etag匹配时才复制
void setNonmatchingETagConstraints(List<String> eTagList)	可选	仅当指定的Etag和Copy Source指定的object的Etag不匹配时才复制
void setModifiedSinceConstraint(Date date)	可选	仅当CopySource在指定时间后更新过才复制
void setUnmodifiedSinceConstraint(Date date)	可选	仅当CopySource在指定时间后未更新过才复制

返回结果说明

- CopyPartResult

获取结果方法	描述
String getETag()	获取分段ETag
Date getLastModifiedDate()	获取对象最后修改时间
String getVersionId()	获取复制后对象的版本号
int getPartNumber()	获取分段号

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;
import java.util.ArrayList;
import java.util.List;

public class copyPart {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String srcBucket = "<your-bucket>";
    public static String dstBucket = "<your-bucket>";
    public static String srcKey = "<your-objectKey>";
    public static String dstKey = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            // 获取要复制的object的信息
            GetObjectMetadataRequest metadataRequest = new
GetObjectMetadataRequest(srcBucket, srcKey);
            ObjectMetadata metadataResult =
s3Client.getObjectMetadata(metadataRequest);
            long objectSize = metadataResult.getContentLength();

            // 初始化multiPartUpload
            InitiateMultipartUploadRequest initRequest = new
InitiateMultipartUploadRequest(dstBucket, dstKey);
            InitiateMultipartUploadResult initResult =
s3Client.initiateMultipartUpload(initRequest);
            String uploadId = initResult.getUploadId();
```

```
            // 以5M为一段复制Object
            long partSize = 5 * 1024 * 1024;
            long bytePosition = 0;
            int partNum = 1;

            List<CopyPartResult> copyResponses = new ArrayList<>();
            while (bytePosition < objectSize) {
                long lastByte = Math.min(bytePosition + partSize - 1, objectSize
- 1);

                // 复制一段
                CopyPartRequest copyRequest = new CopyPartRequest()
                        .withSourceBucketName(srcBucket)
                        .withSourceKey(srcKey)
                        .withDestinationBucketName(dstBucket)
                        .withDestinationKey(dstKey)
                        .withUploadId(uploadId)
                        .withFirstByte(bytePosition)
                        .withLastByte(lastByte)
                        .withPartNumber(partNum++);
                copyResponses.add(s3Client.copyPart(copyRequest));
                bytePosition += partSize;
            }
            // 完成 multiPartUpload
            CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest(
                    dstBucket,
                    dstKey,
                    uploadId,
                    getETags(copyResponses));
            s3Client.completeMultipartUpload(completeRequest);
            System.out.println("Multipart copy complete.");

        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }

    private static List<PartETag> getETags(List<CopyPartResult> responses) {
        List<PartETag> etags = new ArrayList<>();
        for (CopyPartResult response : responses) {
            etags.add(new PartETag(response.getPartNumber(),
response.getETag()));
        }
        return etags;
    }
}
```

获取临时凭证

该操作涉及的前置操作：创建角色、创建策略、绑定角色策略等，请参考官网指引文档：https://www.ctyun.cn/document/10026735/10029128，在控制台进行操作。

承接角色

功能说明

承接一个角色，获得临时凭证（AK、SK、Token），拥有该角色的策略权限。

方法原型

`AssumeRoleResult assumeRole(AssumeRoleRequest assumeRoleRequest)`

参数说明

- AssumeRoleRequest

获取结果方法	是否必选	描述
void setRoleArn(String roleArn)	是	设置角色标识符，"arn:aws:iam:::role/<your-role-name>"
void setRoleSessionName(String roleSessionName)	是	设置任务名（任意）
void setDurationSeconds(Integer durationSeconds)	可选	设置有效期（900秒至角色设置的最大有效期，默认3600秒）

返回结果说明

- AssumeRoleResult

获取结果方法	描述
Credentials getCredentials()	获取临时凭证

- Credentials

获取结果方法	描述
String getAccessKeyId()	获取AK
String getSecretAccessKey()	获取SK
String getSessionToken()	获取Token
java.util.Date getExpiration()	获取过期时间

代码示例

承接角色

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
```

```
import com.amazonaws.services.securitytoken.model.AssumeRoleRequest;
import com.amazonaws.services.securitytoken.model.AssumeRoleResult;
import
com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
import com.amazonaws.services.securitytoken.model.Credentials;

public class assumeRole {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String roleArn = "<your-roleArn>";
    public static String sessionName = "<your-sessionname>";

    public static void main(String[] args) {
        AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);

        AWSSecurityTokenService stsClient =
AWSSecurityTokenServiceClientBuilder.standard()
                .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                .withCredentials(new AWSStaticCredentialsProvider(credentials))
                .build();

        assumeRole(stsClient, roleArn, sessionName);
    }
    public static void assumeRole(AWSSecurityTokenService sts, String roleArn,
String sessionName){
        AssumeRoleRequest request = new AssumeRoleRequest();
        request.setRoleArn(roleArn);
        request.setRoleSessionName(sessionName);
        request.setDurationSeconds(60*60);

        AssumeRoleResult result = sts.assumeRole(request);
        Credentials myCreds = result.getCredentials();
        System.out.println("AK: " + myCreds.getAccessKeyId());
        System.out.println("SK: " + myCreds.getSecretAccessKey());
        System.out.println("token: " + myCreds.getSessionToken());
    }
}
```

使用临时凭证访问对象存储（列出所有桶）

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.Bucket;

import java.util.List;

public class deleteObject {
    public static String ACCESS_KEY = "<your-access-key>";
```

```
    public static String SECRET_KEY = "<your-secret-key>";
    public static String TOKEN = "<your-token>";
    public static String END_POINT = "<your-endpoint>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicSessionCredentials(ACCESS_KEY,
SECRET_KEY, TOKEN);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            List<Bucket> buckets  = s3Client.listBuckets();
            for(int i = 0; i < buckets.size();i++){
                System.out.println(buckets.get(i).getName() + ", " +
buckets.get(i).getOwner().toString());
            }
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

客户端加密

只支持AES256加密算法。

生成加密密钥

```
// 生成Key的要妥善保管，如果该key丢失了，那么意味着通过该key加密的数据将没法解密
public static SecretKey genSecretKey() {
    try {
        KeyGenerator generator = KeyGenerator.getInstance("AES");
        generator.init(256);
        SecretKey secretKey = generator.generateKey();
        return secretKey;
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        return null;
    }
}
```

保存加密密钥

```
public static void save_key(String dir, SecretKey secretKey) {
    try {
        byte[] bytes = encode(secretKey.getEncoded());
        FileOutputStream fos = new FileOutputStream(dir);
        fos.write(bytes);
        fos.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

加载加密密钥

```
public static SecretKey load_key(String dir) {
    try {
        File fileKey = new File(dir);
        FileInputStream fis = new FileInputStream(fileKey);
        byte[] encodedKey = new byte[(int) fileKey.length()];
        fis.read(encodedKey);
        fis.close();
        SecretKeySpec secretKey = new SecretKeySpec(decode(encodedKey), "AES");
        return secretKey;
    } catch(IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

加密上传示例

请求参数与返回和普通上传相同。

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.kms.AWSKMSClient;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3EncryptionClientV2Builder;
import com.amazonaws.services.s3.AmazonS3EncryptionV2;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.io.*;
import java.security.NoSuchAlgorithmException;

public class putObjectEncryption {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
```

```
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String filePath = "<your-path>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            SecretKey secretKey = genSecretKey();
            save_key("<your-key-path>", secretKey);
            AWSKMSClient awskmsClient = new AWSKMSClient();
            AmazonS3EncryptionV2 s3EncryptionClient =
AmazonS3EncryptionClientV2Builder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .withCryptoConfiguration(new
CryptoConfigurationV2().withCryptoMode((CryptoMode.StrictAuthenticatedEncryption
)))
                    .withEncryptionMaterialsProvider(new
StaticEncryptionMaterialsProvider(new EncryptionMaterials(secretKey)))
                    .withKmsClient(awskmsClient)
                    .build();
            System.out.print("=====connect success=====\n");

            File file= new File(filePath);
            PutObjectRequest request = new PutObjectRequest(Bucket, Key, file);
            PutObjectResult result = s3EncryptionClient.putObject(request);
            System.out.format("Etag: %s, VersionId: %s\n",result.getETag(),
result.getVersionId());
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }

    public static SecretKey genSecretKey() {
        try {
            KeyGenerator generator = KeyGenerator.getInstance("AES");
            generator.init(256);
            SecretKey secretKey = generator.generateKey();
            return secretKey;
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }
    public static void save_key(String dir, SecretKey secretKey) {
        try {
            byte[] bytes = encode(secretKey.getEncoded());
```

```
            FileOutputStream fos = new FileOutputStream(dir);
            fos.write(bytes);
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

解密下载示例

请求参数与返回和普通下载相同。

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.kms.AWSKMSClient;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3EncryptionClientV2Builder;
import com.amazonaws.services.s3.AmazonS3EncryptionV2;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.io.*;
import java.security.NoSuchAlgorithmException;

public class getObjectDecryption {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String filePath = "<your-path>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            SecretKey secretKey = load_key("<your-key-path>");
            AWSKMSClient awskmsClient = new AWSKMSClient();
            AmazonS3EncryptionV2 s3EncryptionClient =
AmazonS3EncryptionClientV2Builder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
```

```
                    .withCryptoConfiguration(new
CryptoConfigurationV2().withCryptoMode((CryptoMode.StrictAuthenticatedEncryption
)))
                    .withEncryptionMaterialsProvider(new
StaticEncryptionMaterialsProvider(new EncryptionMaterials(secretKey)))
                    .withKmsClient(awskmsClient)
                    .build();
            System.out.print("=====connect success=====\n");

            GetObjectRequest getObjectRequest = new GetObjectRequest(Bucket,
Key);
            S3Object s3Object = s3EncryptionClient.getObject(getObjectRequest);

            try {
                InputStream in = s3Object.getObjectContent();
                File outputFile = new File(filePath);
                FileOutputStream outputStream = new
FileOutputStream(outputFile);

                byte[] read_buf = new byte[1024 * 1024];
                int read_len = 0;
                while ((read_len = in.read(read_buf)) > 0) {
                    outputStream.write(read_buf, 0, read_len);
                }
                in.close();
                outputStream.close();
            } catch (IOException e){
                e.printStackTrace();
            }
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
    public static SecretKey load_key(String dir) {
        try {
            File fileKey = new File(dir);
            FileInputStream fis = new FileInputStream(fileKey);
            byte[] encodedKey = new byte[(int) fileKey.length()];
            fis.read(encodedKey);
            fis.close();
            SecretKeySpec secretKey = new SecretKeySpec(decode(encodedKey),
"AES");
            return secretKey;
        } catch(IOException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

图片处理

Post请求处理图片

功能说明

该功能是对存储桶中的图片进行处理，并持久化到指定的存储桶中。

方法原型

```
void ProcessObject(ProcessObjectRequest request)
```

参数说明

- ProcessObjectRequest

设定参数方法	是否必选	描述
void setBucketName(String bucketname)	是	设置持久化的目的桶
void setKey(String key)	是	设置持久化存储的目的名称（不能和原图相同）
void setProcessSource(String processource)	是	设置待处理的图片所在桶和对象名，格式："bucket/key"
void setZosProcess(String zosprocess)	是	设置图片处理方式，每一项参数格式均为"参数名称_取值"，之间用逗号分隔具体见下表

- 图片缩放(e.g. "image/resize,w_300,h_200,m_fixed")

参数名称	参数用途	取值	是否必须
w	指定目标缩放图宽度	[1,4096]	使用按百分比缩放可不指定宽高
h	指定目标缩放图高度	[1,4096]	使用按百分比缩放可不指定宽高
m	指定缩放模式	lfit（默认值）：等比缩放，目标缩放图为指定w和h矩形框内的最大图形； mfit：等比缩放，目标缩放图为延伸出指定w和h矩形框外的最小图形； fill：将原图等比缩放为延伸出指定w与h的矩形框外的最小图片，之后将超出的部分进行居中裁剪； pad: 将原图等比缩放为指定w和h矩形框内最大的图形，然后使用color指定的颜色将矩形框内剩余部分进行填充； fixed: 固定宽高，强制缩放。	否
color	缩放模式为pad时，指定填充颜色	RGB颜色值，默认FFFFFF(白色)	否（仅当m为pad模式时使用）
p	按百分比进行缩放	[1,1000]，小于100缩小，大于100放大	否
limit	指定目标缩放图大于原图时是否缩放	1(默认)：目标缩放图大于原图时返回原图； 0：按指定参数缩放	否

- 图片裁剪(e.g. "image/crop,w_100,h_100,x_10,y_10,g_se")

参数名称	参数用途	取值	是否必须
w	指定裁剪宽度。	[0,图片宽度] 默认为最大值。	否
h	指定裁剪高度。	[0,图片高度] 默认为最大值。	否
x	指定裁剪起点，相对原点的横坐标（默认左上角为原点）。	[0,图片边界]	否
y	指定裁剪起点，相对原点的纵坐标（默认左上角为原点）。	[0,图片边界]	否
g	设置裁剪的原点位置。原点按照整幅图的九宫格形式分布，一共有九个位置可以设置，为每个九宫格的左上角顶点。	nw：左上(默认) north：中上 ne：右上 west：左中 center：中部 east：右中 sw：左下 south：中下 se：右下	否

- 图片旋转(e.g. "image/rotate,45")

参数名称	参数用途	取值	是否必须
[value]	图片按顺时针旋转的角度。	[0,360] 默认值：0，表示不旋转。	是

- 水印

图片水印(e.g. image/watermark, image_aHVkaWUuanBnP3gtem9zLXByb2Nlc3M9aW1hZ2UvcmVzaXplLHBfMzAvcm90YXRlLDE4MA==, g_north,t_40)

文字水印(e.g. image/watermark,text_Q2hpbmF0ZWxlY29t,type_heiti,color_FF0000,size_40,g_se,t_80)

参数名称	参数用途	取值	是否必须
t	图片水印或文字水印的透明度	[0, 100]	否
x	文字水印距离图片边界的水平距离	[0, 4096] 默认值：10	否
y	文字水印距离图片边界的垂直距离	[0, 4096] 默认值：10	否
text	指定文字水印内容	Base64编码后的字符串，编码结果字符串中‘/’要替换为‘_’	否
color	指定文字水印的颜色	RGB颜色值。 默认：FFFFFF（白色）	否
size	指定文字水印的字体大小	默认值：40	否
type	指定文字水印的字体类型	如Airal, Helvetica, 支持中文字体包括yahei(微软雅黑)，heiti(黑体)，kaishu（楷书）,youyuan(幼圆)	否
rotate	指定文字水印顺时针旋转角度	[0, 360] 默认值：0	否
image	指定图片水印名称，水印图片需要和原图存放在相同存储空间	水印图片可以进行预处理（e.g. 水印图片缩放为30%并旋转180度，hudie.jpg?x-zos-process=image/resize,p_30/rotate,180 ），需要转换成base64编码，编码结果字符串中‘/’要替换为‘_’	否
g	指定水印在图片中的位置	nw：左上 north：中上 ne：右上 west：左中 center：中部 east：右中 sw：左下 south：中下 se：右下 (默认)	否

- 格式转化(e.g. "image/format,png")

参数名称	参数用途	取值	是否必须
无	将原图转换成指定格式	jpg、png、webp、bmp、tiff	是

返回结果说明

无

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.ProcessObjectRequest;
import java.util.Base64;


public class ProcessObject {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String srcBucket = "<your-srcBucket>";
    public static String dstBucket = "<your-dstBucket>";
    public static String srcKey = "<your-srcKey>";
    public static String dstKey = "<your-dstKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");


            ProcessObjectRequest request = new ProcessObjectRequest();
            request.setBucketName(dstBucket);
            request.setKey(dstKey);
            request.setProcessSource(srcBucket + "/" + srcKey);
            request.setZosProcess("image/crop,w_100,h_100,x_10,y_10,g_se");
            s3Client.ProcessObject(request);
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
```

```
            System.out.print(e.getMessage());
        }
    }
}
```

Get请求获取图片

功能说明

图片处理是在get_object基础上进行的扩展，用来对存储桶中的图片对象在线进行处理。

方法原型

`S3Object getObject(GetObjectRequest request)`

参数说明

- GetObjectRequest

设定参数方法	是否必选	描述
GetObjectRequest(String bucketName, String key)	可选	构造函数设置桶名、对象名
GetObjectRequest(String bucketName, String key, String versionId)	可选	构造函数设置桶名、对象名、版本号)
void setZosProcess(String zosprocess)	可选	设置图片处理方式，包含缩放(resize)，裁剪(crop)，旋转(rotate)，水印(watermark)，格式转换(format)，具体见上节

返回结果说明

- S3Object

获取结果方法	描述
S3ObjectInputStream getObjectContent()	获取对象内容
String getBucketName()	获取对象所在桶
String getKey()	获取对象key
ObjectMetadata getObjectMetadata()	获取对象元数据

- ObjectMetadata

获取结果方法	描述
Map<String, String> getUserMetadata()	获取用户元数据
Date getLastModified()	获取对象创建时间
String getETag()	获取对象ETag
String getVersionId()	获取对象版本号
Long[] getContentRange()	若请求时指定了Range或partnumber，则返回结果中包含content range
String getObjectLockMode()	获取对象的合规保留模式
Date getObjectLockRetainUntilDate()	获取Retention 过期日期
String getStorageClass()	获取对象存储类型

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.S3Object;

import java.io.*;

public class getObject {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String savePath = "<your-path>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
```

```
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            GetObjectRequest request = new GetObjectRequest(Bucket, Key);
            request.setZosProcess("image/crop,w_100,h_100,x_10,y_10,g_se");
            S3Object result = s3Client.getObject(request);
            System.out.print("=====request success=====\n");
            // 保存到本地
            try {
                InputStream in = result.getObjectContent();
                File outputFile = new File(savePath);
                FileOutputStream outputStream = new
FileOutputStream(outputFile);

                byte[] read_buf = new byte[1024 * 1024];
                int read_len = 0;
                while ((read_len = in.read(read_buf)) > 0) {
                    outputStream.write(read_buf, 0, read_len);
                }
                in.close();
                outputStream.close();
            } catch (IOException e){
                e.printStackTrace();
            }
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

特定场景

批量碎片清理

功能说明

清除桶内的所有文件碎片（分段上传的未完成部分）。

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.Protocol;
import com.amazonaws.services.s3.model.*;
import java.util.List;

public class defragment {
```

```
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");
            //获取分段上传信息
            ListMultipartUploadsRequest var = new
ListMultipartUploadsRequest(Bucket);

            MultipartUploadListing result = s3Client.listMultipartUploads(var);
            List<MultipartUpload> summary = result.getMultipartUploads();
            for(MultipartUpload T:summary) {
                AbortMultipartUploadRequest abortReq = new
AbortMultipartUploadRequest (Bucket,T.getKey(),T.getUploadId());
                s3Client.abortMultipartUpload(abortReq);
            }
            System.out.print("=====request success=====\n");
        }catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
}
```

批量解冻

功能说明

解冻桶内特定前缀路径（文件夹）下的所有归档对象。

代码示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
```

```java
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.model.*;
import com.amazonaws.Protocol;

public class restoreObjects {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Prefix = "<your-prefix>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
            System.out.print("=====connect success=====\n");

            ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
            listObjectsRequest.setBucketName(Bucket);
            listObjectsRequest.setPrefix(Prefix);

            boolean istruncated = true;
            String marker = "";
            // 每次返回最多1000条，循环请求直到获取全部对象
            while(istruncated) {
                listObjectsRequest.setMarker(marker);
                ObjectListing objects =
s3Client.listObjects(listObjectsRequest);
                for (S3ObjectSummary summary : objects.getObjectSummaries()){
                    String key = summary.getKey();
                    ObjectMetadata meta = s3Client.getObjectMetadata(Bucket,
key);
                    if (meta.getStorageClass() != null &&
meta.getStorageClass().equals("GLACIER")){
                        System.out.println("restoring: " + key);
                        RestoreObjectRequest restoreObjectRequest = new
RestoreObjectRequest(Bucket, key);
//                          restoreObjectRequest.setExpirationInDays(2);  //设置解冻
有效期，默认1天
                        s3Client.restoreObjectV2(restoreObjectRequest);
                    }
                }
```

```
                istruncated = objects.isTruncated();
                marker = objects.getNextMarker();
            }
        } catch (AmazonServiceException e) {
            System.out.println(e.getMessage());
        } catch (SdkClientException e) {
            e.printStackTrace();

        }
    }
}
```

预签名分段上传

功能说明

生成临时的预签名的Url，没有权限访问集群的用户可以通过预签名Url创建分段上传、上传分段、列出分段、完成分段上传、取消分段上传。

代码示例

- 创建初始化分段上传链接并初始化分段上传

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.HttpMethod;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

import java.io.IOException;
import java.net.URL;
import java.util.Date;

public class presignedUrlInitiateMultipartUpload {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
```

```
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();

            // 指定生成的预签名URL的过期时间，单位为毫秒，示例设置过期时间为12小时
            Date expiration = new Date();
            long expTimeMills = expiration.getTime();
            expTimeMills += 12 * 60 * 60 * 1000;
            expiration.setTime(expTimeMills);

            // 生成预签名的创建分段上传URL
            GeneratePresignedUrlRequest presignInitiateMultipartUploadRequest =
                    new GeneratePresignedUrlRequest(Bucket, Key)
                            .withMethod(HttpMethod.POST)
                            .withExpiration(expiration)
                            .withZeroByteContent(true);
            presignInitiateMultipartUploadRequest.addRequestParameter("uploads",
null);

            // 设置请求头参数需在预签名请求和用户HttpPost请求同时设置
            // 可设置acl: "private"|"public-read"
            presignInitiateMultipartUploadRequest.putCustomRequestHeader("x-amz-
acl", "private");
            // 可设置 storage-class: "STANDARD"|"STANDARD_IA"|"GLACIER"
            presignInitiateMultipartUploadRequest.putCustomRequestHeader("x-amz-
storage-class", "STANDARD");
            // 可设置 tagging: "tag1=value1&tag2=value2"
            presignInitiateMultipartUploadRequest.putCustomRequestHeader("x-amz-
tagging", "tag1=value1&tag2=value2");
            URL signedInitiateMultipartUploadUrl =
s3Client.generatePresignedUrl(presignInitiateMultipartUploadRequest);
            System.out.println("signedInitiateMultipartUploadUrl: " +
signedInitiateMultipartUploadUrl);

            // 其他用户得到预签名链接后，进行初始化分段上传

 initiateMultipartUploadWithSignedUrl(signedInitiateMultipartUploadUrl);
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }

    public static void initiateMultipartUploadWithSignedUrl(URL
signedInitiateMultipartUploadUrl) throws IOException {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        CloseableHttpResponse response = null;
        try {
            // 创建分段上传
```

```
            HttpPost post = new
HttpPost(signedInitiateMultipartUploadUrl.toString());

            // 可设置 acl: "private"|"public-read"
            post.addHeader("x-amz-acl", "private");
            // 可设置 storage-class: "STANDARD"|"STANDARD_IA"|"GLACIER"
            post.addHeader("x-amz-storage-class", "STANDARD");
           // 可设置 tagging: "tag1=value1&tag2=value2"
            post.addHeader("x-amz-tagging", "tag1=value1&tag2=value2");
            response = httpClient.execute(post);
            int responseCode = response.getStatusLine().getStatusCode();
            if (responseCode == 200) {
                System.out.print("InitiateMultipartUpload success, Content: ");
            } else {
                System.out.print("InitiateMultipartUpload failed, response
code: " + responseCode + ", Content: ");
            }
            String responseBody = EntityUtils.toString(response.getEntity());
            System.out.println(responseBody);
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        } finally {
            if (response != null) {
                response.close();
            }
            if (httpClient != null) {
                httpClient.close();
            }
        }
    }
}
```

- 创建上传分段链接并上传分段

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.HttpMethod;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import com.amazonaws.services.s3.model.PartETag;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.Header;
import org.apache.http.entity.BufferedHttpEntity;
import org.apache.http.entity.InputStreamEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.IOException;
```

```
import java.io.InputStream;
import java.net.URL;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class presignedUrlUploadPart {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String UploadId = "<your-uploadId>";
    public static int PartCounts;
    public static String filePath = "<your-filePath>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();

            // 指定生成的预签名URL的过期时间，单位为毫秒，示例设置过期时间为12小时
            Date expiration = new Date();
            long expTimeMills = expiration.getTime();
            expTimeMills += 12 * 60 * 60 * 1000;
            expiration.setTime(expTimeMills);

            // 生成预签名的上传分段URL
            // 分段数量PartCounts由需要进行预签名分段上传的用户提供
            List<URL> signeUploadPartUrls = new ArrayList<URL>();
            for (int partNum = 1; partNum <= PartCounts; ++partNum) {
                GeneratePresignedUrlRequest presignUploadPartRequest =
                        new GeneratePresignedUrlRequest(Bucket, Key)
                                .withMethod(HttpMethod.PUT)
                                .withExpiration(expiration);
                presignUploadPartRequest.addRequestParameter("partNumber",
String.valueOf(partNum));
                presignUploadPartRequest.addRequestParameter("uploadId",
UploadId);
                URL signeUploadPartUrl =
s3Client.generatePresignedUrl(presignUploadPartRequest);
                signeUploadPartUrls.add(signeUploadPartUrl);
```

```
                System.out.println("signeUploadPartUrl part " + partNum + ": " +
signeUploadPartUrl);
            }

            // 其他用户得到预签名链接后，进行上传分段
            uploadPartWithSignedUrl(signeUploadPartUrls);
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }

    public static void uploadPartWithSignedUrl(List<URL> signeUploadPartUrls)
throws IOException {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        CloseableHttpResponse response = null;
        httpClient = HttpClients.createDefault();
        try {
            // 上传分段
            File file = new File(FilePath);
            long fileLength = file.length();
            InputStream inStream = Files.newInputStream(file.toPath());
            int partNumber = 1; //初始分段号
            long partSize = 5 * 1024 * 1024; // 分段大小（字节），大于5M
            List<PartETag> partETags = new ArrayList<PartETag>();
            for (URL signedUrl : signeUploadPartUrls) {
                long startPos = (partNumber - 1) * partSize;
                long curPartSize = (partNumber == PartCounts) ? (fileLength -
startPos) : partSize;
                byte[] buffer = new byte[(int) curPartSize];
                inStream.read(buffer);
                InputStreamEntity entity = new InputStreamEntity(new
ByteArrayInputStream(buffer), curPartSize);
                BufferedHttpEntity byteArrayEntity = new
BufferedHttpEntity(entity);

                HttpPut put = new HttpPut(signedUrl.toString());
                put.setEntity(byteArrayEntity);
                response = httpClient.execute(put);
                int responseCode = response.getStatusLine().getStatusCode();
                if (responseCode == 200) {
                    Header eTagHeader = response.getFirstHeader("ETag");
                    if (eTagHeader != null) {
                        String eTag = eTagHeader.getValue().replace("\"", "");
                        partETags.add(new PartETag(partNumber, eTag));
                        System.out.println("UploadPart success, part " +
partNumber + ", eTag: " + eTag);
                    }
                } else {
                    System.out.println("part " + partNumber + " upload failed,
response code: " + responseCode);
                    put.releaseConnection();
                    return;
                }
                ++partNumber;
            }
            System.out.println("partETags:");
            for (PartETag partETag : partETags) {
```

```
                System.out.print("partNumber: " + partETag.getPartNumber());
                System.out.println(", eTag: " + partETag.getETag());
            }
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        } finally {
            if (response != null) {
                response.close();
            }
            if (httpClient != null) {
                httpClient.close();
            }
        }
    }
}
```

- 创建列出分段链接并列出分段

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.HttpMethod;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

import java.io.IOException;
import java.net.URL;
import java.util.Date;

public class presignedUrlListPart {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String UploadId = "<your-uploadId>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
```

```
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();

            // 指定生成的预签名URL的过期时间，单位为毫秒，示例设置过期时间为12小时
            Date expiration = new Date();
            long expTimeMills = expiration.getTime();
            expTimeMills += 12 * 60 * 60 * 1000;
            expiration.setTime(expTimeMills);

            // 生成预签名的列出分段URL
            GeneratePresignedUrlRequest presignListPartRequest =
                    new GeneratePresignedUrlRequest(Bucket, Key)
                            .withMethod(HttpMethod.GET)
                            .withExpiration(expiration);
            presignListPartRequest.addRequestParameter("uploadId", UploadId);
            URL signedListPartsUrl =
s3Client.generatePresignedUrl(presignListPartRequest);
            System.out.println("signedListPartsUrl: " + signedListPartsUrl);

            // 其他用户得到预签名链接后，进行列出分段
            listPartWithSignedUrl(signedListPartsUrl);
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }

    public static void listPartWithSignedUrl(URL signedListPartsUrl) throws
IOException {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        CloseableHttpResponse response = null;
        try {
            // 列出分段
            HttpGet get = new HttpGet(signedListPartsUrl.toString());
            response = httpClient.execute(get);
            int responseCode = response.getStatusLine().getStatusCode();
            if (responseCode == 200) {
                System.out.print("ListParts success, Content: ");
            } else {
                System.out.print("ListParts failed, response code: " +
responseCode + ", Content: ");
            }
            String responseBody = EntityUtils.toString(response.getEntity());
            System.out.println(responseBody);
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        } finally {
            if (response != null) {
                response.close();
            }
            if (httpClient != null) {
```

```
                httpClient.close();
            }
        }
    }
}
```

- 创建完成分段上传链接并完成分段上传

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.HttpMethod;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;
import com.amazonaws.services.s3.model.transform.RequestXmlFactory;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.ByteArrayEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;

import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class presignedUrlCompleteMultipartUpload {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String UploadId = "<your-uploadId>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
```

```
                    .build();

            // 指定生成的预签名URL的过期时间，单位为毫秒，示例设置过期时间为12小时
            Date expiration = new Date();
            long expTimeMills = expiration.getTime();
            expTimeMills += 12 * 60 * 60 * 1000;
            expiration.setTime(expTimeMills);

            // 生成预签名的完成分段上传URL
            GeneratePresignedUrlRequest presignCompleteMultipartUploadRequest =
                    new GeneratePresignedUrlRequest(Bucket, Key)
                            .withMethod(HttpMethod.POST)
                            .withExpiration(expiration)
                            .withZeroByteContent(true);

 presignCompleteMultipartUploadRequest.addRequestParameter("uploadId",
UploadId);
            URL signedCompleteMultipartUploadUrl =
s3Client.generatePresignedUrl(presignCompleteMultipartUploadRequest);
            System.out.println("signedCompleteMultipartUploadUrl: " +
signedCompleteMultipartUploadUrl);

            // 其他用户得到预签名链接后，进行完成分段上传

 completeMultipartUploadWithSignedUrl(signedCompleteMultipartUploadUrl);
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }

    public static void completeMultipartUploadWithSignedUrl(URL
signedCompleteMultipartUploadUrl) throws IOException {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        CloseableHttpResponse response = null;
        try {
            // 完成分段上传
            List<PartETag> partETags = new ArrayList();
            // 以下partETags为示例代码，需要在UploadPart时通过响应结果拼装出partETags，
或通过解析ListPart的响应结果拼装出partETags
            partETags.add(new PartETag(1, "ha6a0d097e307ac52ed9b4ad551801k3"));
            partETags.add(new PartETag(2, "dc6a0d097e307ac52ed9b4ad551801rt"));
            HttpPost completePost = new
HttpPost(signedCompleteMultipartUploadUrl.toString());
            byte[] xml = RequestXmlFactory.convertToXmlByteArray(partETags);
            HttpEntity entity = new ByteArrayEntity(xml);
            completePost.setEntity(entity);
            response = httpClient.execute(completePost);
            int responseCode = response.getStatusLine().getStatusCode();
            if (responseCode == 200) {
                System.out.println("CompleteMultipartUpload success");
            } else {
                System.out.println("CompleteMultipartUpload failed, response
code: " + responseCode);
                System.out.println(response);
            }
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
```

```
            System.out.print(e.getMessage());
        } finally {
            if (response != null) {
                response.close();
            }
            if (httpClient != null) {
                httpClient.close();
            }
        }
    }
}
```

- 创建取消分段上传链接并取消分段上传

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.HttpMethod;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpDelete;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;

import java.io.IOException;
import java.net.URL;
import java.util.Date;

public class presignedUrlAbortMultipartUpload {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String UploadId = "<your-uploadId>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
```

```
                    .enablePathStyleAccess()
                    .build();

            // 指定生成的预签名URL的过期时间，单位为毫秒，示例设置过期时间为12小时
            Date expiration = new Date();
            long expTimeMills = expiration.getTime();
            expTimeMills += 12 * 60 * 60 * 1000;
            expiration.setTime(expTimeMills);

            // 生成预签名的取消分段上传URL
            GeneratePresignedUrlRequest presignAbortMultipartUploadRequest =
                    new GeneratePresignedUrlRequest(Bucket, Key)
                            .withMethod(HttpMethod.DELETE)
                            .withExpiration(expiration);
            presignAbortMultipartUploadRequest.addRequestParameter("uploadId",
UploadId);
            URL signedAbortMultipartUploadUrl =
s3Client.generatePresignedUrl(presignAbortMultipartUploadRequest);
            System.out.println("signedAbortMultipartUploadUrl: " +
signedAbortMultipartUploadUrl);

            // 其他用户得到预签名链接后，进行取消分段上传
            multipartUploadWithSignedUrl(signedAbortMultipartUploadUrl);
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }

    public static void multipartUploadWithSignedUrl(URL
signedAbortMultipartUploadUrl) throws IOException {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        CloseableHttpResponse response = null;
        try {
            // 取消分段上传
            HttpDelete delete = new
HttpDelete(signedAbortMultipartUploadUrl.toString());
            response = httpClient.execute(delete);
            int responseCode = response.getStatusLine().getStatusCode();
            if (responseCode == 204) {
                System.out.println("AbortMultipartUpload success");
            } else {
                System.out.println("AbortMultipartUpload failed, response code: "
+ responseCode);
                System.out.println(response);
            }
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        } finally {
            if (response != null) {
                response.close();
            }
            if (httpClient != null) {
                httpClient.close();
            }
        }
    }
```

```
}
```

- 使用预签名链接分段上传文件示例

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.HttpMethod;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;
import com.amazonaws.services.s3.model.transform.RequestXmlFactory;
import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.*;
import org.apache.http.entity.BufferedHttpEntity;
import org.apache.http.entity.ByteArrayEntity;
import org.apache.http.entity.InputStreamEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;

import java.io.*;
import java.net.URL;
import java.nio.file.Files;
import java.util.*;

public class presignedUrlMultipartUpload {
    public static String ACCESS_KEY = "<your-access-key>";
    public static String SECRET_KEY = "<your-secret-key>";
    public static String END_POINT = "<your-endpoint>";
    public static String Bucket = "<your-bucket>";
    public static String Key = "<your-objectKey>";
    public static String UploadId = "<your-uploadId>";
    public static int PartCounts;
    public static String filePath = "<your-filePath>";

    public static void main(String[] args) {
        AmazonS3 s3Client = null;
        try {
            AWSCredentials credentials = new BasicAWSCredentials(ACCESS_KEY,
SECRET_KEY);
            ClientConfiguration awsClientConfig = new ClientConfiguration();
            awsClientConfig.setSignerOverride("AWSS3V4SignerType");
            awsClientConfig.setProtocol(Protocol.HTTP);

            s3Client = AmazonS3ClientBuilder.standard()
                    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
                    .withClientConfiguration(awsClientConfig)
                    .withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(END_POINT, ""))
                    .disableChunkedEncoding()
                    .enablePathStyleAccess()
                    .build();
```

```
            // 指定生成的预签名URL的过期时间，单位为毫秒，示例设置过期时间为12小时
            Date expiration = new Date();
            long expTimeMills = expiration.getTime();
            expTimeMills += 12 * 60 * 60 * 1000;
            expiration.setTime(expTimeMills);

            // 初始化分段上传
            InitiateMultipartUploadRequest initRequest = new
InitiateMultipartUploadRequest(Bucket, Key);
            initRequest.setStorageClass(StorageClass.Standard);
            InitiateMultipartUploadResult result =
s3Client.initiateMultipartUpload(initRequest);
            String uploadId = result.getUploadId();
            System.out.println("UploadId: " + uploadId);

            // 生成预签名的上传分段URL
            // 分段数量由需要进行预签名分段上传的用户提供
            List<URL> signeUploadPartUrls = new ArrayList<URL>();
            for (int partNum = 1; partNum <= partCounts; ++partNum) {
                GeneratePresignedUrlRequest presignUploadPartRequest =
                        new GeneratePresignedUrlRequest(Bucket, Key)
                                .withMethod(HttpMethod.PUT)
                                .withExpiration(expiration);
                presignUploadPartRequest.addRequestParameter("partNumber",
String.valueOf(partNum));
                presignUploadPartRequest.addRequestParameter("uploadId",
uploadId);
                URL signeUploadPartUrl =
s3Client.generatePresignedUrl(presignUploadPartRequest);
                signeUploadPartUrls.add(signeUploadPartUrl);
                System.out.println("signeUploadPartUrl part " + partNum + ": " +
signeUploadPartUrl);
            }

            // 生成预签名的完成分段上传URL
            GeneratePresignedUrlRequest presignCompleteMultipartUploadRequest =
                    new GeneratePresignedUrlRequest(Bucket, Key)
                            .withMethod(HttpMethod.POST)
                            .withExpiration(expiration)
                            .withZeroByteContent(true);

 presignCompleteMultipartUploadRequest.addRequestParameter("uploadId",
uploadId);
            URL signedCompleteMultipartUploadUrl =
s3Client.generatePresignedUrl(presignCompleteMultipartUploadRequest);
            System.out.println("signedCompleteMultipartUploadUrl: " +
signedCompleteMultipartUploadUrl);

            // 需要进行预签名分段上传的用户得到链接后，进行上传分段和完成分段上传
            multipartUploadWithSignedUrl(signeUploadPartUrls,
signedCompleteMultipartUploadUrl);
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        }
    }
```

```
    public static void multipartUploadWithSignedUrl(List<URL>
signeUploadPartUrls, URL signedCompleteMultipartUploadUrl) throws IOException {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        CloseableHttpResponse response = null;
        try {
            // 上传分段
            File file = new File(filePath);
            long fileLength = file.length();
            InputStream inStream = Files.newInputStream(file.toPath());
            int partNumber = 1; //初始分段号
            long partSize = 5 * 1024 * 1024; // 分段大小（字节），大于5M
            List<PartETag> partETags = new ArrayList<PartETag>();
            int responseCode;
            for (URL signedUrl : signeUploadPartUrls) {
                long startPos = (partNumber - 1) * partSize;
                long curPartSize = (partNumber == partCounts) ? (fileLength -
startPos) : partSize;

                byte[] buffer = new byte[(int) curPartSize];
                inStream.read(buffer);
                InputStreamEntity entity = new InputStreamEntity(new
ByteArrayInputStream(buffer), curPartSize);
                BufferedHttpEntity byteArrayEntity = new
BufferedHttpEntity(entity);

                HttpPut put = new HttpPut(signedUrl.toString());

                put.setEntity(byteArrayEntity);
                response = httpClient.execute(put);
                responseCode = response.getStatusLine().getStatusCode();
                if (responseCode == 200) {
                    Header eTagHeader = response.getFirstHeader("ETag");
                    if (eTagHeader != null) {
                        String etag = eTagHeader.getValue().replace("\"", "");
                        partETags.add(new PartETag(partNumber, etag));
                        System.out.println("UploadPart success, part " +
partNumber + ", etag: " + etag);
                    }
                } else {
                    System.out.println("part " + partNumber + " upload failed,
response code: " + responseCode);
                    put.releaseConnection();
                    return;
                }
                ++partNumber;
            }

            // 完成分段上传
            HttpPost completePost = new
HttpPost(signedCompleteMultipartUploadUrl.toString());
            byte[] xml = RequestXmlFactory.convertToXmlByteArray(partETags);
            HttpEntity entity = new ByteArrayEntity(xml);
            completePost.setEntity(entity);
            response = httpClient.execute(completePost);
            responseCode = response.getStatusLine().getStatusCode();
            if (responseCode == 200) {
                System.out.println("CompleteMultipartUpload success");
            } else {
```

```
                System.out.println("CompleteMultipartUpload failed, response
code: " + responseCode);
                System.out.println(response);
            }
        } catch (Exception e) {
            System.out.print("=====request fail=====\n");
            System.out.print(e.getMessage());
        } finally {
            if (response != null) {
                response.close();
            }
            if (httpClient != null) {
                httpClient.close();
            }
        }
    }
}
```