天翼云媒体存储

Android SDK 使用指导书

天翼云科技有限公司

2026年3月27日

天翼云媒体存储Android SDK使用指导书

接入方式

方式一：使用官网下载的SDK

在官网下载SDK，下载地址：Android SDK

下载完成之后，解压到项目根路径下，修改gradle配置文件

```
dependencies {
    implementation 'com.google.code.gson:gson:2.2.4'
    implementation 'com.fasterxml.jackson.core:jackson-core:2.1.1'
    implementation files('oss-android-sdk/aws-android-sdk-core.jar', 'oss-
android-sdk/aws-android-sdk-kms.jar', 'oss-android-sdk/aws-android-sdk-s3.jar')
```

方式二：直接使用aws的SDK

直接修改gradle配置文件

```
dependencies {
    implementation "com.amazonaws:aws-android-sdk-s3:2.16.8"
```

环境设置

android sdk支持android api level 21

```
defaultConfig {
    minSdkVersion 21
    targetSdkVersion 29
}
```

配置读写sd卡和网络权限

```
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

初始化SDK

注意：直接在客户端上使用主账号存在账号泄露的风险，在客户端上必须使用sts功能生成的临时账号，此初始化流程只能用于测试。如何使用sts初始化参考 本章节的安全凭证服务(STS)章节。

通过sdk使用s3服务的时候主要需要设置3个配置参数，accessKey，secretKey和endpoint，使用以下方法进行设置，完成sdk的初始化。

```
private void CreateS3Client() {
    if (sS3Client == null) {
        String accessKey = "填入你的accesskey";
        String secretKey = "填入你的secretkey";
        BasicAWSCredentials credentials = new BasicAWSCredentials(accessKey,
secretKey);
        sS3Client = new AmazonS3Client(credentials);
        sS3Client.setEndpoint("http://填入S3的地址和端口");
    }
}
```

参数	说明
accessKey	用户账号 access key
secretKey	用户账号 secret key
endpoint	天翼云资源池的地址，必须指定http或https前缀

桶相关接口

创建桶

Bucket是用于存储对象（Object）的容器，所有的对象都必须隶属于某个Bucket。本章节介绍如何创建桶（Bucket）。

接口定义

```
// 简化接口
public Bucket createBucket(String bucketName)
// 完整接口
public Bucket createBucket(CreateBucketRequest createBucketRequest)
```

参数说明

参数名	类型	说明
bucketName	String	bucket名称
region	String	如果非NULL，则是用于授权签名的AWS区域
cannedAcl	CannedAccessControlList	设定的权限

代码示例

```
public void CreateBuckets(String bucketName, OnS3ResponseListener<Bucket>
listener) {
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            Bucket bucket = sS3Client.createBucket(bucketName);

            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onResponse(bucket));
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
        }
    });
}
```

获取桶列表

Bucket是用于存储对象（Object）的容器，所有的对象都必须隶属于某个Bucket。本章节介绍如何获取桶（Bucket）列表。

接口定义

```
public List<Bucket> listBuckets()
```

代码示例

```
public void ListBuckets(OnS3ResponseListener<List<Bucket>> listener){
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            List<Bucket> list = sS3Client.listBuckets();

            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onResponse(list));
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
        }
    });
}
```

判断桶是否存在

Bucket是用于存储对象（Object）的容器，所有的对象都必须隶属于某个Bucket。本章节介绍如何判断桶（Bucket）是否存在。

接口定义

```
public boolean doesBucketExist(String bucketName)
```

参数说明

参数名	类型	说明
bucketName	String	bucket名

代码示例

```
public void DoesBucketExist(String bucketName, OnS3ResponseListener<Boolean>
listener){
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            boolean exist = sS3Client.doesBucketExist(bucketName);

            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onResponse(exist));
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
        }
    });
}
```

删除桶

Bucket是用于存储对象（Object）的容器，所有的对象都必须隶属于某个Bucket。本章节介绍如何删除桶（Bucket）。

注意：待删除的bucket必须是空的，否则会报错。

接口：

```
// 简化接口
public void deleteBucket(String bucketName)
// 完整接口
public void deleteBucket(DeleteBucketRequest deleteBucketRequest)
```

参数说明

参数名	类型	说明
bucketName	String	要删除的bucket名

代码示例

```
public void DeleteBuckets(String bucketName, OnS3ResultListener listener) {
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            sS3Client.deleteBucket(bucketName);

            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onSuccess());
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
        }
    });
}
```

对象相关接口

获取对象列表

对象是存储数据的基本单元。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由桶内部唯一的Key来标识。本章节介绍如何获取对象列表。

接口定义

```
// 简化接口
public ObjectListing listObjects(String bucketName)
// 完整接口
public ObjectListing listObjects(ListObjectsRequest listObjectsRequest)
```

参数说明

参数名	类型	说明
bucketName	String	包含bucket及相关的请求参数
prefix	String	如果非NULL，则仅列举以指定的prefix作为前缀的对象
marker	String	如果非NULL，指定一个标识符，在列举桶内对象列表时，返回的对象列表将仅是按照字典顺序排序后位于这个标识符之后的对象
delimiter	String	如果非NULL，则是用来对桶内对象进行分组的字符串。所有名称包含指定的前缀且第一次出现delimiter字符之间的对象将作为一组元素，在返回信息的CommonPrefixes节点显示
maxkeys	int	指定返回对象的最大数量，若为0则列举所有对象

代码示例

```
public void ListObjects(String bucketName,
OnS3ResponseListener<List<S3ObjectSummary>> listener) {
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            List<S3ObjectSummary> list =
sS3Client.listObjects(bucketName).getObjectSummaries();

            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onResponse(list));
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
        }
    });
}
```

上传对象

对象是存储数据的基本单元。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由桶内部唯一的Key来标识。本章节介绍如何上传对象。

接口定义

```
// 简化接口
public PutObjectResult putObject(String bucketName, String key, File file)
// 完整接口
public PutObjectResult putObject(PutObjectRequest putObjectRequest)
```

参数说明

参数名	类型	说明
bucketName	String	包含bucket及相关的请求参数
key	String	将要上传的对象的文件名
file	FILE	将要上传的文件对象，和inputStream二选一
inputStream	InputStream	将要上传的文件流，和file二选一
metadata	ObjectMetadata	可选参数，元数据，可以设置ContentType和自定义元数据
cannedAcl	CannedAccessControlList	可选参数，权限控制参数
accessControlList	AccessControlList	可选参数，权限控制参数

代码示例

```
public void PutObject(String bucketName, String objectKey, File file,
                              OnS3ResponseListener<PutObjectResult> listener) {
    GlobalThreadPool.getInstance().execute(() -> {
        try {
```

```
            PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName,
objectKey, file);
            ObjectMetadata metadata = new ObjectMetadata();
            //metadata.setContentType("image/jpeg");        // content-type
            //metadata.addUserMetadata("my-metadata-key", "my-metadata-value");
    // 自定义元数据
            putObjectRequest.setMetadata(metadata);
            PutObjectResult result = sS3Client.putObject(putObjectRequest);
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onResponse(result));
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
        }
    });
}
```

注意：putObject对文件大小有限制，最大能上传5GB大小的文件，超过5GB需要使用分片上传。

下载对象

对象是存储数据的基本单元。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由桶内部唯一的Key来标识。本章节介绍如何下载对象。

接口定义

```
// 简化接口
public S3Object getObject(String bucketName, String key)
// 完整接口
public S3Object getObject(GetObjectRequest getObjectRequest)
```

参数说明

参数名	类型	说明
s3ObjectIdBuilder	S3ObjectIdBuilder	包含bucketName和key
range	long[]	获取文件的区间
generalProgressListener	ProgressListener	进度回调

代码示例

```
public void GetObjects(String bucketName, String objectKey,
                       OnS3ResponseListener<S3Object> listener) {
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            S3Object result = sS3Client.getObject(bucketName, objectKey);
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onResponse(result));
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
        }
    });
}
```

复制对象

对象是存储数据的基本单元。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由桶内部唯一的Key来标识。本章节介绍如何复制对象。

接口定义

```
// 简化接口
public CopyObjectResult copyObject(String sourceBucketName, String sourceKey,
String destinationBucketName, String destinationKey)
// 完整接口
public CopyObjectResult copyObject(CopyObjectRequest copyObjectRequest)
```

参数说明

参数名	类型	说明
sourceBucketName	String	源bucket及请求参数
sourceKey	String	源对象名
destinationBucketName	String	目的bucket
destinationKey	String	目的对象名

代码示例

```
public void CopyObjects(String sourceBucketName, String sourceObjectKey,
                        String destBucketName, String destObjectKey,
                        OnS3ResponseListener<CopyObjectResult> listener) {
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            CopyObjectResult result = sS3Client.copyObject(sourceBucketName,
sourceObjectKey, destBucketName, destObjectKey);

            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onResponse(result));
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
```

```
        }
    });
}
```

删除对象

对象是存储数据的基本单元。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由桶内部唯一的Key来标识。本章节介绍如何删除对象。

接口定义

```
// 简化接口
public void deleteObject(String bucketName, String key)
// 完整接口
public void deleteObject(DeleteObjectRequest deleteObjectRequest)
```

参数

参数名	类型	说明
bucketName	String	包含bucket及相关的请求参数
key	String	要删除的对象名称

代码示例

```
public void DeleteObjects(String bucketName, String objectKey,
                          OnS3ResultListener listener) {
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            sS3Client.deleteObject(bucketName, objectKey);
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onSuccess());
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
        }
    });
}
```

获取对象元数据

对象是存储数据的基本单元。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由桶内部唯一的Key来标识。本章节介绍如何获取对象元数据。

接口定义

```
// 简化接口
public ObjectMetadata getObjectMetadata(String bucketName, String key)
// 完整接口
public ObjectMetadata getObjectMetadata(GetObjectMetadataRequest
getObjectMetadataRequest)
```

参数

参数名	类型	说明
bucketName	String	包含bucket及相关的请求参数
key	String	对象名称

代码示例

```
public void GetObjectMetadata(String bucketName, String objectKey,
                        OnS3ResponseListener<ObjectMetadata> listener) {
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            ObjectMetadata result = sS3Client.getObjectMetadata(bucketName,
objectKey);
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onResponse(result));
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
        }
    });
}
```

分片上传

融合接口

SDK提供封装好的融合接口，方便用户实现分片上传的功能。

接口定义

```
public TransferObserver upload(String bucket, String key, File file)

public TransferObserver upload(String bucket, String key, File file,
CannedAccessControlList cannedAcl)

public TransferObserver upload(String bucket, String key, File file,
ObjectMetadata metadata)

public TransferObserver upload(String bucket, String key, File file,
ObjectMetadata metadata, CannedAccessControlList cannedAcl)

public TransferObserver upload(String bucket, String key, File file,
ObjectMetadata metadata, CannedAccessControlList cannedAcl, TransferListener
listener)
```

参数说明

参数名	类型	说明
bucketName	String	bucket名
key	String	要上传的对象名称
file	FILE	上传的文件对象
metadata	ObjectMetadata	可选参数，元数据，可以设置ContentType和自定义元数据
cannedAcl	CannedAccessControlList	文件操作权限(Private|PublicRead)
listener	TransferListener	上传文件回调

代码示例

```
private void beginUpload(File file) {
    TransferUtility transferUtility = TransferUtility.builder()
            .context(context)
            .s3Client(S3Client)
            .build();

    TransferObserver observer = transferUtility.upload(
            bucketName,
            key,
            file
    );
}
```

关于Content-Type的配置

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

在android sdk中，如果用户没有设置Content-Type，会根据对象的key后缀扩展名自动生成Content-Type。

创建分片上传任务

创建分片上传任务，返回分片上传任务的ID。

接口定义

```
public InitiateMultipartUploadResult initiateMultipartUpload(
            InitiateMultipartUploadRequest initiateMultipartUploadRequest)
```

参数说明

参数名	类型	说明
bucketName	String	bucket名
key	String	要上传的对象名
cannedACL	CannedAccessControlList	权限设置
objectMetadata	ObjectMetadata	可选参数，元数据，可以设置ContentType和自定义元数据

代码示例

```
public void InitiateMultipartUpload(String bucketName, String objectKey,
    OnS3ResponseListener<InitiateMultipartUploadResult> listener) {
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            InitiateMultipartUploadRequest initiateMultipartUploadRequest =
                    new InitiateMultipartUploadRequest(bucketName, objectKey);
            InitiateMultipartUploadResult result =
sS3Client.initiateMultipartUpload(initiateMultipartUploadRequest);
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onResponse(result));
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
        }
    });
}
```

上传一个分片

获取到分片任务ID之后，通过ID来上传分片内容到S3服务器。

接口定义

```
public UploadPartResult uploadPart(UploadPartRequest uploadPartRequest)
```

参数说明

参数名	类型	说明
bucketName	String	bucket名
key	String	要上传的对象名称
file	FILE	上传的文件对象
uploadId	String	上传任务ID
partSize	long	上传内容长度
partNum	int	分片ID（1-10000）
fileOffset	long	文件起始位置

代码示例

```
public void UploadPart(String bucketName, String objectKey, String uploadId,
    int partNum, File file, int filePosition, int partSize,
    OnS3ResponseListener<UploadPartResult> listener) {
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            UploadPartRequest uploadPartRequest = new UploadPartRequest()
                    .withBucketName(bucketName)
                    .withKey(objectKey)
                    .withUploadId(uploadId)
                    .withPartNumber(partNum)
                    .withFileOffset(filePosition)
                    .withFile(file)
                    .withPartSize(partSize);
            UploadPartResult result = sS3Client.uploadPart(uploadPartRequest);
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onResponse(result));
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
        }
    });
}
```

完成分片上传任务

完成所有分片的上传之后，调用完成接口，服务端会把所有分片合并成对象保存。

接口定义

```
public CompleteMultipartUploadResult completeMultipartUpload(
            CompleteMultipartUploadRequest completeMultipartUploadRequest)
```

参数说明

参数名	类型	说明
bucketName	String	bucket名
key	String	要上传的对象名
uploadId	String	上传任务ID
partETags	List	上传的分片信息列表

代码示例

```
public void CompleteMultipartUpload(String bucketName,
    String objectKey,
    String uploadId,
    List<PartETag> partETags,
    OnS3ResponseListener<CompleteMultipartUploadResult> listener) {
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            CompleteMultipartUploadRequest completeMultipartUploadRequest = new
CompleteMultipartUploadRequest(
                    bucketName, objectKey, uploadId, partETags);
            CompleteMultipartUploadResult result =
sS3Client.completeMultipartUpload(completeMultipartUploadRequest);
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onResponse(result));
        } catch (Exception e) {
            Handler mainHandler = new Handler(Looper.getMainLooper());
            mainHandler.post(() -> listener.onError());
        }
    });
}
```

终止分片上传任务

上传失败的时候调用此接口，服务器会清除残留的分片数据。

接口定义

```
public void abortMultipartUpload(AbortMultipartUploadRequest
abortMultipartUploadRequest)
```

参数说明

参数名	类型	说明
bucketName	String	bucket名
key	String	要上传的对象名
uploadId	String	上传任务ID

代码示例

```
public void AbortMultipartUpload(String bucketName, String objectKey, String
uploadId) {
    GlobalThreadPool.getInstance().execute(() -> {
        try {
            AbortMultipartUploadRequest abortMultipartUploadRequest = new
AbortMultipartUploadRequest(
                    bucketName, objectKey, uploadId);
            sS3Client.abortMultipartUpload(abortMultipartUploadRequest);
        } catch (Exception e) {
        }
    });
}
```

安全凭证服务(STS)

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

获取临时token

在服务端生成临时token，可参考java、python、nodejs、CPP、donet、go、php SDK说明，请从SDK概览 页面选择进入对应的开发指南查阅。

使用临时token

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

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

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

    public synchronized AWSCredentials getCredentials() {
        return credentials;
    }

    public synchronized void refresh() {
    }

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

使用临时token

```
String accessKey = "<your-access-key>";
String secretKey = "<your-secret-access-key>";
String endPoint = "<your-endpoint>";
String sessionToken = "<your-session-token>";

MyCredentialsProvider credProvider = new MyCredentialsProvider(accessKey,
secretKey, sessionToken);
ClientConfiguration clientConfig = new ClientConfiguration();
clientConfig.setProtocol(Protocol.HTTP);
AmazonS3Client mS3Client = new AmazonS3Client(credProvider, clientConfig);
mS3Client.setEndpoint(endPoint);
```