阿里云开放存储服务

ALIYUN OPEN STORAGE SERVICE

Android SDK参考手册

(Release 0.2.1)

2014.12.17

1. 前言

本文档介绍了OSS Android SDK的使用方式。

OSS Android SDK是为了方便移动开发者在Android端能更简易地使用OSS云存储服务，而基于OSSRESTful接口(阿里云开放存储服务)实现的一个存储组件。通过该SDK，开发者开发的app可以直接从终端向OSS服务端进行数据存取、数据删除、数据拷贝等操作，并且这些操作同时提供了同步和异步两种使用模式。

详细API和部分类结构见 **(API文档)** 。

需要注意的一点是，本SDK是基于OSS开发的一个旨在满足移动端开发者数据存储需求的一个无线端基础组件， 提供的是让移动端应用简便地对接**OSS**进行数据存取的能力，而并没有提供进行**OSS**控制台管理的功能 ，比如申请Bucket、管理Bucket、域名绑定、开通静态网站托管等。也不会提供全局浏览Bucket内数据的功能，数据之间的映射关系还需要开发者自己进行维护。

---

2. 关于OSS

开放存储服务（Open Storage Service，OSS），是阿里云对外提供的海量、安全和高可靠的云存储服务。本SDK是OSS为Android移动开发者量身订做的一套Android平台的API接口，所以，在使用本SDK前，你需要先到阿里云官网开通OSS服务，以及了解OSS的基本用法。

OSS官网页面： **http://www.aliyun.com/product/oss**

OSS手册： **OSS API 手册**

---

3. 安装OSS Android SDK

3.1 直接在Eclipse中使用

当你下载了OSS Android SDK的jar包后，进行以下步骤：

- 在官网下载OSS Android SDK
- 解压后得到jar包

- 将此jar包复制到你的Android工程的lib目录下
- 在Eclipse右键工程 -> Properties -> Java Build Path -> Add JARs，导入你刚才复制的jar包

经过以上几步之后，你就可以在工程中使用OSS Android SDK了。

---

4. 应用程序初始化

在你使用OSS Android SDK进行OSS上的数据操作以前，你需要对SDK的上下文进行一些初始化配置，如权限声明、实现加签方法、传递程序Context、设置数据中心域名等。其中权限声明在AndroidManifest.xml文件中进行，其余动作你需要在代码中，通过 `OSSClient` 类的全局静态方法来完成。

在整个应用程序的生命周期，这些初始化行为只需要你在使用OSS Android SDK前进行一次。

4.1 权限声明

以下是OSS Android SDK所需要的Android权限，请把这些权限配置到你的AndroidManifest.xml文件，否则，SDK将无法正常工作。

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

4.2 应用程序Context传入

OSS Android SDK工作时需要使用应用程序的Context，所以，你需要在初始化时对此进行设置。这个动作通过 `OSSClient` 类完成，代码如下：

```
OSSClient.setApplicationContext(getApplicationContext());
```

需要指出，这个操作只在第一次生效，后续的重复设置将会被忽略。

4.3 访问控制

当你在阿里云官网开通OSS服务并创建自己的存储空间---Bucket以后，你就可以用OSS Android SDK在终端进行数据的存取了。为了保证你的数据安全，OSS已经在服务端做了妥善的安全设置，相应的，你需要实现对应的一些鉴权步骤，才能畅通无阻地访问到你的数据。

你在注册OSS时，系统会给你分配一对 Access Key ID 和 Access Key Secret ，称为 ID 对，本文中简称为 AK/SK ，用于你在访问OSS时做签名验证。AK/SK具体的使用方法可参考OSS API手册第4章节。简而言之，你向OSS服务端发起的每个请求，都需要携带上 AK/SK 对请求内容加签得到的Token，鉴权通过后，OSS服务端才会处理你的请求。

由于你进行的是终端上的开发，AK/SK保管的安全问题不容忽视。所以，OSS Android SDK并不会索取你的 AK/SK ，而是由你来实现加签方法，SDK会在发起对OSS的请求前，调用你的加签方法生成Token。

你需要调用的方法是：

```
OSSClient.setGlobalDefaultTokenGenerator(TokenGenerator tokenGen);
```

TokenGenerator是一个抽象类，内部有一个抽象方法是你需要实现的。它的参数与OSS API手册4.2节中Authorization字段计算方法是一一对应的，你只需要将这些参数用AK/SK加签生成Token后返回。

```
public abstract String generateToken(
            String httpMethod,
            String md5,
            String type,
            String date,
            String ossHeaders,
            String resource);
```

如果不考虑 AK/SK 的安全性，一个sample是这样的：

```
static final String accessKey = "Your accessKey"; // 实际使用中，AK/SK不应明文保存在代码中
```

```
static final String secretKey = "Your secretKey";
OSSClient.setGlobalDefaultTokenGenerator(new TokenGenerator() {
    @Override
    public String generateToken(String httpMethod, String md5, String type, String date, String ossHea
ders,
            String resource) {
        String content = httpMethod + "\n" + md5 + "\n" + type + "\n" + date
                + "\n" + ossHeaders + resource;

        return OSSToolKit.generateToken(accessKey, secretKey, content);
    }
});
```

以上代码中使用到的工具类函数已经保留在SDK中，方便你用之进行测试。其中 OSSToolKit.generateToken(...) 的实现细节是这样的:

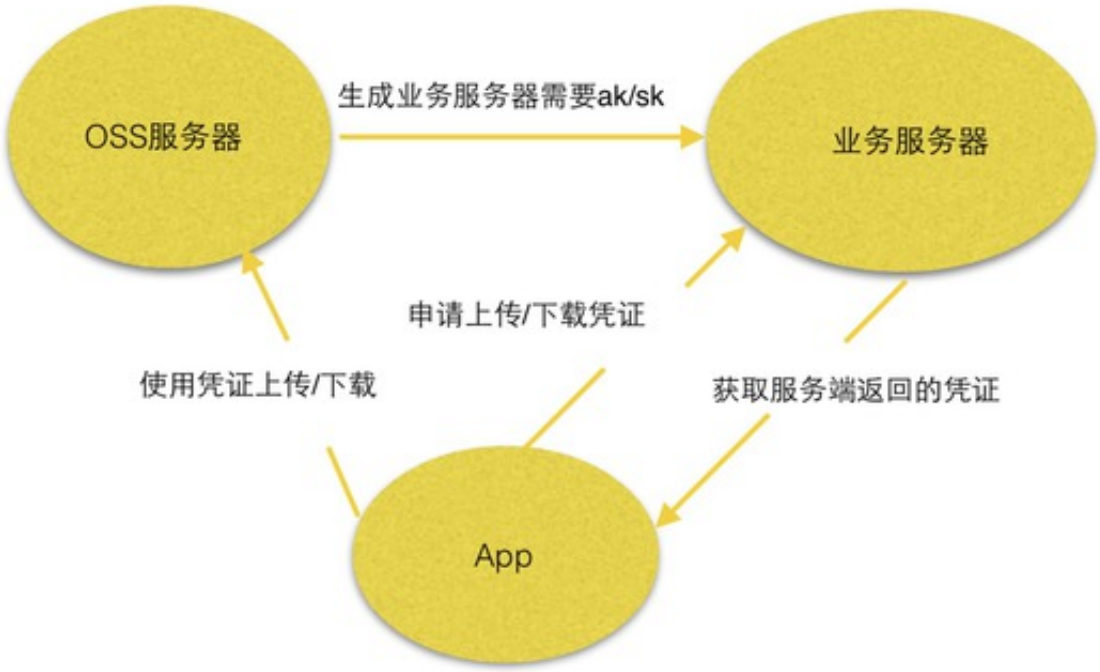
```
public static String generateToken(String accessKey, String secretKey, String content) {
    String signature = null;

    try {
        signature = OSSToolKit.getHmacSha1Signature(content, secretKey);
        signature = signature.trim();
    } catch (Exception e) {
        OSSLog.logD(e.toString());
    }

    return "OSS " + accessKey + ":" + signature; // 注意"OSS"和accessKey之间存在一个空格！！！
}
```

但需要再次强调， AK/SK 不应明文保存在代码中，你应该设计一套安全性可以接受的 AK/SK 保管方案。

一个建议的编程模型是这样的：



如图中所示，你可以将 AK/SK 保存在你的业务服务器中，然后，你将 TokenGenerator 中的 generateToken() 方法实现为，把这些参数通过http请求post到你的业务服务器上，在你的业务服务器上进行加签，返回token，你再把这个token返回给 TokenGenerator 。

于是，整个过程中 AK/SK 不会暴露在终端上，有效地规避了泄密的风险。

安全性与易用性、效率难以兼得，请你在确定AK/SK的保管方案前，谨慎评估其中的风险与收益。

4.4 设置全局默认bucket访问权限

在OSS官网上创建一个Bucket时，你可以为之指定公共读写、公共读、私有三种访问权限。所以你在SDK中，也需要指明你想访问的Bucket被设置了哪种权限。你可以在全局初始化时为所有Bucket设置一个默认值，在实例化Bucket时，你可以通过显式指定来覆盖这个设置。

```
ossBucket.setBucketACL(AccessControlList.PRIVATE);
```

如果你没有调用这个接口进行设置，那么OSS Android SDK将默认为你设置全局Bucket访问权限为 PRIVATE

4.5 设置数据中心域名

你在OSS官网上创建一个 Bucket 时，可以根据费用单价、请求来源分布、响应延迟等方面的考虑，为该 bucket 选择所在的数据中心。如果在创建 Bucket 时未指定所属的数据中心，OSS 将会为其自动分配一个默认的数据中心，目前默认的数据中心为 oss-cn-hangzhou。参考 **OSS API 文档** 。

于是，在进行对OSS进行数据相关的操作时，你需要通过一个域名来指明你的bucket的所在的数据中心，通过以下接口完成：

```
OSSClient.setGlobalDefaultHostId("oss-cn-qingdao.aliyuncs.com"); // 指明你的bucket是放在青岛数据中心
```

如果你 没有 调用这个接口进行设置，那么OSS Android SDK将默认为你设置hostId为 `oss-cn-hangzhou.aliyuncs.com` 。

4.6 CNAME域名绑定设置

OSS 支持用户将自定义的域名绑定在属于自己的 bucket 上面，这个操作必须通过 OSS 控制台（http://www.aliyun.com/product/oss）来实现。已经绑定CNAME的bucket在使用OSS Android SDK访问时，需要在OSSCliet初始化时将该CNAME域名作为setOSSHostId()方法的参数传入。如：

```
OSSClient.setGlobalDefaultHostId("your.cname.host.com"); // 将OSS Android SDK的请求Host设置为bucket绑定的cname域名
```

4.7 初始化总览

如果你在初始化时进行了所有设置，那么初始化部分的代码可能是这样：

```
static final String accessKey = "Your accessKey"; // 实际使用中，AK/SK不应明文保存在代码中
static final String secretKey = "Your secretKey";
OSSClient.setGlobalDefaultTokenGenerator(new TokenGenerator() {
    @Override
    public String generateToken(String httpMethod, String md5, String type, String date, String ossHeaders,
            String resource) {
```

```
        String content = httpMethod + "\n" + md5 + "\n" + type + "\n" + date
                + "\n" + ossHeaders + resource;

        return OSSToolKit.generateToken(accessKey, secretKey, content);
    }
});
OSSClient.setGlobalDefaultACL(AccessControlList.PRIVATE);
OSSClient.setGlobalDefaultHostId("oss-cn-hangzhou.aliyuncs.com");
OSSClient.setApplicationContext(getApplicationContext());
```

其中， TokenGenerator 如果留空，可以等到构造Bucket的时候再进行相应设置。其他设置项如果你没有进行设置，都将采用默认值。

---

5. 建立Bucket以及设置

Bucket是OSS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体，自然的，在你进行数据相关的操作前，你需要先行构造一个Bucket类实例并进行设置，以便在后面进行数据操作用该Bucket实例来指明数据存放的位置。

构造一个Bucket非常简单：

```
OSSBucket sampleBucket = new OSSBucket("oss-example");
```

然后进行对一些设置选项的设置：（如果不设置，部分选项将从OSSClient继承）

```
sampleBucket.setBucketACL(AccessControlList.PRIVATE); // 指明该Bucket的访问权限
sampleBucket.setBucketHostId("oss-cn-hangzhou.aliyuncs.com"); // 指明该Bucket所在数据中心的域名
sampleBucket.setBucketTokenGen(tokenGenerator); // 指明该Bucket访问时应该采用的加签方法，如果Bucket访问权限为Public，那么可以为null
```

通常来说，Bucket可以认为是一个全局概念，因为大部分情况下，你所有的数据操作，可能都跟某一个Bucket是相关的。因此，你应该也是在全局初始化时构造好这些Bucket，后面进行数据操作的时候，你

只需把对应Bucket传入指明数据位置即可。

---

6. 数据的存储

OSS Android SDK对数据的存储相关操作由 OSSData 类实现。这里的数据指的是在应用程序运行时在内存中的数据。所以，如果你要在程序运行时上传内存中的一段数据，或者期望把OSS上的一条数据下载到本地内存中作为byte[]数组进行处理，那你应该通过这个类进行操作。

构造 OSSData 类实例时，你需要指定 OSSBucket 和 ObjectKey ，指明你进行的操作将与OSS上的哪条数据相关。实例的获得如下：

```
OSSData ossData = new OSSData(OSSBucket ossBucket, String ObjectKey);
```

6.1 数据下载

在你构造出 OSSData 实例后，如果你指定的 OSSBucket 和 ObjectKey 对应的数据已经存在于OSS上，你就可以将它下载下来了。代码如下：

```
OSSData ossData = new OSSData(sampleBucket, "sample-data"); // 构造OSSData实例
byte[] data = ossData.get(); // 下载失败将会抛出异常
```

这里需要注意：

- 上面只是简单的示例代码，实际应用时，下载可能因为各种情况导致失败，所以你应该编写捕捉异常的代码，并对异常进行处理。异常的解释请参考后面章节。
- 这是一个同步方法，将会阻塞线程直到方法返回。
- 由于进行数据操作时，OSS Android SDK需要与OSS服务端进行网络交互，这些代码将被认为是耗时的阻塞性操作。而在 Android 平台上，大部分的代码是在主线程(也称为UI线程)上运行的，如果在主线程上进行耗时的阻塞性操作，你的代码可能会无法正常运行。所以，你应该在非UI线程中执行这些代码，或者，你可以使用SDK提供的异步接口 getInBackgroud() ，参考5.2节。

另外， 如果你的文件是放在文件夹中 ， 只需要在 ObjectKey 同样写明路径即可：

```
OSSData ossData = new OSSData(sampleBucket, "fileFolder/sample-data"); // 构造OSSData实例
byte[] data = ossData.get(); // 下载失败将会抛出异常
```

在数据下载结束后，你可以调用 getMeta() 方法来获取meta数据：

```
List<BasicNameValuePair> metaList = OSSData.getMeta();
```

这个方法只有在数据下载完毕后才能拿到结果（如果是异步下载需要在 onSuccess() 触发以后），如果你希望直接能获取meta数据，请参考第8节。

6.2 数据下载的异步版本

实际上，OSS Android SDK为所有耗时的方法提供了异步版本，允许你传入自己实现好的回调方法，在新的线程中进行相关操作，并异步回调你的处理逻辑。

对于数据下载，异步接口的使用方式是这样的：

```
OSSData ossData = new OSSData(sampleBucket, "sample-data");
ossData.getInBackground(new GetBytesCallback() {
    @Override
    public void onSuccess(String objectKey, byte[] data) {
        // 在这里对下载下来的数据进行处理
    }

    @Override
    public void onProgress(String objectKey, int byteWirtten, int totalSize) {
        // 在这里可以根据下载进度进行相关操作
    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {
        // 下载失败时，从这里获取异常信息和进行异常处理
    }
```

```
});
```

请你在使用异步接口时， 务必传入回调方法 。如果你对某个回调方法并不关心，可以保留空实现。

需要指出，你实现的这些回调方法将会在任务处理线程中被OSS Android SDK调用，也就是说，是在非UI线程中执行。在Android中，非UI线程是不能进行对UI元素的控制的，所以，如果你有在回调方法中控制UI元素的需求，需要通过Android提供的消息传递机制来实现。这方面已经有许多资料可以参考，这里不再赘述。

我们不在SDK中预先实现把回调方法传递到UI线程执行的理由是，如果所有回调方法都这么做，将会极大限制SDK使用场景的灵活性。

6.3 数据上传

如果在应用程序运行时你需要把内存中的一段数据上传到OSS，那么你需要先指定 `OSSBucket` 和 `ObjectKey` 构造出 `OSSData` 实例，然后调用 `setData` 接口指定需要上传的数据和它的类型，最后再调用上传方法进行上传。

你还可以在上传时选择是否开启MD5校验。

代码如下：

```
OSSData ossData = new OSSData(sampleBucket, "sample-data");
ossData.setData(data, "raw"); // 指定需要上传的数据和它的类型
ossData.enableUploadCheckMd5sum(); // 开启上传MD5校验
ossData.upload(); // 上传失败将会抛出异常
```

上传成功后，OSS上 `sampleBucket` 对应的bucket将会多了一条名为 `sampleData` 的数据，其内容就是你上传的 `data` 。

同样的，数据上传也有异步版本：

```
OSSData ossData = new OSSData(sampleBucket, "sample-data");
ossData.setData(data, "raw"); // 指定需要上传的数据和它的类型
```

```
ossData.enableUploadCheckMd5sum(); // 开启上传MD5校验
ossData.uploadInBackground(new SaveCallback() {
    @Override
    public void onSuccess(String objectKey) {

    }

    @Override
    public void onProgress(String objectKey, int byteCount, int totalSize) {

    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

6.4 数据删除

在构造出代表你指定的 OSSBucket 和 ObjectKey 对应数据的 OSSData 以后，你可以调用删除方法将这条数据删除。代码如下：

```
OSSData ossData = new OSSData(sampleBucket, "sample-data"); // 构造OSSData实例
ossData.delete(); // 删除失败将会抛出异常
```

同样有异步版本：

```
OSSData ossData = new OSSData(sampleBucket, "sample-data"); // 构造OSSData实例
data.deleteInBackground(new DeleteCallback() {
    @Override
    public void onSuccess(String objectKey) {
```

```
    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

删除操作是没有进度的，你不需要为之实现进度回调方法。下面的拷贝方法也类似。

6.5 数据拷贝

你可以构造一个OSS还不存在的指定 OSSBucket 和 ObjectKey 的 OSSData ，然后调用拷贝方法将别的已经存在于OSS上的数据拷贝过来。拷贝的源可以是同一个bucket的数据，也可以是不同bucket的数据。

代码如下：

```
OSSData ossData = new OSSData(sampleBucket, "sample-data");
ossData.copyFrom("sourceData"); // 拷贝自同一个bucket
```

//或者：

```
ossData.copyFrom("sourceBucket", "sourceData"); // 拷贝自别的bucket
```

异步版本：

```
OSSData ossData = new OSSData(sampleBucket, "sample-data");
ossData.copyFromInBackgroud("sourceData", new CopyCallback() {
    @Override
    public void onSuccess(String objectKey) {

    }
```

```
    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

//或者：

```
ossData.copyFromInBackgroud("sourceBucket", "sourceData", new CopyCallback() {
    @Override
    public void onSuccess(String objectKey) {

    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

6.6 下载时指定范围

在下载数据前，可以通过设置 Range 来指定下载范围：

```
ossData.setRange(222, 888)); // 只对下载操作有效，且必须在下载前调用才生效
```

6.7 上传时添加自定义meta属性

在OSS上的数据都有自己的meta属性，除了系统自动添加的之外，用户也可以指定以"x-oss-meta-"为前缀的自定义属性，具体含义请参考OSS API手册的2.1节和5.4.5节。

这些你在上传时指定的属性可以通过SDK中的获取meta属性方法来单独获取。见本文档第7章节。

上传前添加自定义meta属性的代码如下：

```
ossData.addXOSSHeader("x-oss-meta-key1", "value1"); // 只对上传操作有效，且必须在上传前调用才生效
ossData.addXOSSHeader("x-oss-meta-key2", "value2"); // 自定义的meta属性必须以"x-oss-meta-"为前缀
ossData.addXOSSHeader("x-oss-meta-key3", "value3"); // 不支持同名的meta属性键值对
```

注意，如果添加的meta属性不是以"x-oss-meta-"为前缀的，将会被忽略。

6.8 生成数据的URL

在构造指定 OSSBucekt 和 objectKey 的数据后，你可以调用 getResourceURL() 接口生成一个访问URL，以实现对第三方的授权URL访问。 如果数据所在的Bucket没有开放 公共读 权限，你需要为该接口传入 accessKey 和一个有效时长，这个URL将在这个有效时长后失效。

```
OSSData.getResourceURL(String accessKey, int availablePeriodInSeconds); // 生成的URL将在availablePeriodInSeconds秒后失效
```

如果你的Bucket开放了 公共读 权限，你只需要调用：

```
OSSData.getResourceURL(); // 由于Bucket为公共读权限，不再需要传入accessKey和有效期
```

6.9 异步上传、下载任务的取消

对移动端来说，特别是在2G/3G网络下，一个较大数据的上传、下载的操作可能会耗费比较长的时间。而在这任务处理期间，你可能有某种原因需要放弃这次任务。 因此，OSS Android SDK为这种场景提供了数据异步上传/下载，文件异步上传/下载接口的取消功能。在你开启一个异步任务后，接口会返回一个 TaskHandler ，你可以调用 cancel() 接口取消这次任务。如果任务被成功取消，任务将进入 onFailure() 回调，抛出一个 InterruptIOException 异常。

```
TaskHandler tHandler = OSSData.getInBackground(new GetBytesCallback() {...}); // 开启一个异步下载任务
// doSomething
tHandler.cancel(); // 因为某种原因需要取消任务
```

---

7. 文件操作

在OSS Android SDK里，文件操作的集合放在 OSSFile 类中，这个类的用法和 OSSData 大同小异。事实上，两个类的 区别只在于， OSSData 用以存取放在内存中的数据，而 OSSFile ，用于终端本地文件的直接上传和OSS数据直接下载到本地文件中。

考虑到上传大文件耗时较长，中途可能会发生各种异常导致上传失败，所以，OSS Android SDK提供了大文件上传的断点续传接口。

7.1 下载到文件

下载到文件的操作和5.1节的数据下载操作类似，唯一不同的是，下载时指定目标文件路径。下载方法不会返回数据。

```
OSSFile ossFile = new OSSFile(sampleBucket, "sample-data");
ossFile.downloadTo("/your/path/to/file"); // 下载失败将抛出异常
```

异步版本：

```
ossFile.downloadToInBackground("/your/path/to/file", new GetFileCallback() {
    @Override
    public void onSuccess(String objectKey, String filePath) {

    }

    @Override
    public void onProgress(String objectKey, int byteCount, int totalSize) {

    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {
```

```
    }
});
```

可以看到，异步接口的 onSuccess 回调方法也不再有byte[]数组作为参数，取而代之的是下载成功后的文件保存路径。

7.2 从文件上传

从文件上传的接口也改变为，在上传前，指定要上传文件的路径。

```
OSSFile ossFile = new OSSFile(sampleBucket, "sample-data");
ossFile.setUploadFilePath("/your/path/to/file", "your content type"); // 指明需要上传的文件的路径，和文件内容类型
ossFile.enableUploadCheckMd5sum(); // 开启上传md5校验
ossFile.upload(); // 上传失败将会抛出异常
```

异步版本：

```
ossFile.uploadInBackground(new SaveCallback() {
    @Override
    public void onSuccess(String objectKey) {

    }

    @Override
    public void onProgress(String objectKey, int byteCount, int totalSize) {

    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
```

```
});
```

7.3 其余常规操作

上面两小节已经说明 OSSFile 和 OSSData 的区别，可以看出，从OSS的角度看，它们是没有区别的，都是存储于OSS上的一段数据。所以，两个类除了涉及数据形式的上传下载接口有区别外， 其他接口的用法都是一致的，比如拷贝、删除、指定下载范围、添加自定义**meta**属性、生成**URL**、任务取消功能等 。

这里不再赘述。

7.4 断点续传

由于大文件的上传将会有比较长的耗时，这个过程中失败的风险很高，为此，OSS Android SDK提供了断点续传的接口。当你使用此接口上传文件时，如果途中因为各种原因导致上传失败，你下次上传只要指定的是和上次一样的 OSSbucket ， ObjectKey ，同时要上传的文件未曾改动，那么，此次上传将会从上次失败的进度继续进行。

同时，在你调用这个接口时，接口也会返回一个任务 handler ，在上传过程中你随时可以用这个 handler 取消掉这次上传任务。任务取消后进入 onFailure() 逻辑。

这个接口只有异步版本。示例代码如下：

```
OSSFile ossFile = new OSSFile(sampleBucket, "large.data");
ossFile.setUploadFilePath("/your/path/to/file", "your file content type"); // 指定要上传的文件，和文件内容的类型
TaskHandler tk = ossFile.ResumableUploadInBackground(new SaveCallback() {
    @Override
    public void onSuccess(String objectKey) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onProgress(String objectKey, int byteCount, int totalSize) {
```

```
        // TODO Auto-generated method stub
    }

    @Override
    public void onFailure(String objectKey, OSSException ossException) {
        // TODO Auto-generated method stub
    }
});

// doSomething

tk.cancel(); // 上传任务执行过程中，你可以随时放弃这次上传
```

---

8. 只获取Meta

可以用 OSSMeta 类来单独获取数据的meta属性。获取得到的meta属性集合将以 List<BasicNameValuePair> 的形式返回给你。

示例代码如下：

```
OSSMeta meta = new OSSMeta(sampleBucket, "sample-data");
List<BasicNameValuePair> namevalue = meta.getMeta();
```

异步版本：

```
OSSMeta meta = new OSSMeta(sampleBucket, "sample-data");
meta.getMetaInBackground(new GetMetaCallback() {
    @Override
    public void onSuccess(String objectKey, List<BasicNameValuePair> meta) {

    }
```

```
    @Override
    public void onFailure(String objectKey, OSSException ossException) {

    }
});
```

---

9. 异常处理

OSS Android SDK中很多接口是需要和OSS server端进行网络交互的，每个请求还要经过独立的鉴权，或者涉及本地文件的操作等，这些动作可能会遇到各种异常情况。为此，SDK提供了专门的异常类---`OSSException` 来负责异常信息的收集和反馈。

所有发生的异常都会被包裹在 `OSSException` 中。在使用同步接口时，异常将会以抛出的形式反馈；而在使用异步接口时，异常将会传递给失败时需要执行的回调方法--- `onFailure(String objectKey, OSSException ossException)` 进行处理。

异常将会被分类：本地异常和OSS异常。前者指系统发生的异常，如网络连接失败、IO异常、文件异常、参数异常、状态异常等。后者指OSS系统无法处理请求，如资源不存在、鉴权失败、文件过大等。

在你处理异常前，应该首先对异常的类别进行判断。

9.1 本地异常

假设你已经抓获异常 `ossException` ，你可以这样获取异常信息，并进行相应处理：

```
try {
    //...
} catch (OSSException ossException) {
    //处理前务必进行异常类别的判断，不同类别处理方法不同
    if (ossException.getExceptionType() == ExceptionType.OSS_EXCEPTION) {
        String objectKey = ossException.getObjectKey(); // 获取该任务对应的ObjectKey
        String mesString = ossException.getMessage(); // 异常信息
        String info = ossException.toString();
```

```
        Exception localException = ossException.getException(); // 取得原始的异常
        ossException.printStackTrace(); // 打印栈
    }
}
```

9.2 OSS异常

假设你已经抓获异常 ossException ，你可以这样获取异常信息，并进行相应处理：

```
try {
    //...
} catch (OSSException ossException) {
    //处理前务必进行异常类别的判断，不同类别处理方法不同
    if (ossException.getExceptionType() == ExceptionType.OSS_EXCEPTION) {
        String objectKey = ossException.getObjectKey(); // 获取该任务对应的ObjectKey
        OSSResponseInfo resp = ossException.getOssRespInfo(); // 获取根据OSS响应的内容构造的数据结构
        int statusCode = resp.getStatusCode(); // OSS响应的http状态码
        Document dom = resp.getResponseInfoDom(); // 根据OSS响应内容解析得到的文档结构，你可以通过它获取更详细的信息
        String errorCode = resp.getCode(); // OSS反馈的错误码
        String requestId = resp.getRequestId(); // 该次任务的请求ID
        String hostId = resp.getHostId(); // 该次任务请求的主机
        String message = resp.getMessage(); // OSS反馈的错误信息
        String info = ossException.toString(); // 本次异常的信息汇总
        ossException.printStackTrace(); // 打印栈
    }
}
```

这些异常的信息是根据OSS API文档的标准错误响应构造出来的，你可以利用这些信息定位问题，并进行处理。参考OSS API文档第6章节。

---

10. 完整示例

以下是在Android程序中使用OSS Android SDK下载数据的完整流程：

```
package com.example.testoss;


import android.app.Activity;
import android.os.Bundle;
import android.util.Log;


import com.aliyun.mbaas.oss.OSSClient;
import com.aliyun.mbaas.oss.callback.GetBytesCallback;
import com.aliyun.mbaas.oss.model.AccessControlList;
import com.aliyun.mbaas.oss.model.OSSException;
import com.aliyun.mbaas.oss.model.TokenGenerator;
import com.aliyun.mbaas.oss.storage.OSSBucket;
import com.aliyun.mbaas.oss.storage.OSSData;
import com.aliyun.mbaas.oss.util.OSSToolKit;


public class MainActivity extends Activity {


    static final String accessKey = "Your accessKey"; // 测试代码没有考虑AK/SK的安全性
    static final String screctKey = "Your secretKey";


    static {
        OSSClient.setGlobalDefaultTokenGenerator(new TokenGenerator() {
            @Override
            public String generateToken(String httpMethod, String md5, String type, String date,
                    String ossHeaders, String resource) {


                String content = httpMethod + "\n" + md5 + "\n" + type + "\n" + date + "\n" + ossHeade
rs
                        + resource;


                return OSSToolKit.generateToken(accessKey, screctKey, content);
```

```
            }
        });
        OSSClient.setGlobalDefaultHostId("oss-cn-hangzhou.aliyuncs.com");
        OSSClient.setGlobalDefaultACL(AccessControlList.PRIVATE);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        OSSClient.setApplicationContext(getApplicationContext()); // 传入应用程序context

        OSSBucket sampleBucket = new OSSBucket("oss-example");
        // sampleBucket.setBucketACL(AccessControlList.PRIVATE); // 如果这个Bucket跟全局默认的访问权限不一致，就需要单独设置
        // sampleBucket.setBucketHostId("oss-cn-hangzhou.aliyuncs.com"); // 如果这个Bucket跟全局默认的数据中心不一致，就需要单独设置
        // sampleBucket.setBucketTokenGen(new TokenGenerator() {...}); // 如果这个Bucket跟全局默认的加签方法不一致，就需要单独设置

        OSSData downloadData = new OSSData(sampleBucket, "sample-data");
        Log.d("OSS Android SDK", "begin download data");
        downloadData.getInBackground(new GetBytesCallback() {

            @Override
            public void onSuccess(String objectKey, byte[] data) {
                // TODO Auto-generated method stub
                Log.d("OSS Android SDK", "complete download data");
            }

            @Override
            public void onProgress(String objectKey, int byteCount, int totalSize) {
```

```
                // TODO Auto-generated method stub
                Log.d("OSS Android SDK", "downloading");
            }

            @Override
            public void onFailure(String objectKey, OSSException e) {
                // TODO Auto-generated method stub
                Log.d("OSS Android SDK", e.toString());
            }
        });
    }
}
```
