Aliyun Android SDK

阿里云安卓 SDK 设计开发说明书

Version 1.0.0

2012-10-01

作者姓名	联系方式
黄舒志	shuzhi21@sina.com
罗睿辞	luoruici@gmail.com
杨珺	yangjvn@126.com

文档修改记录					
版本	日期	相关章节	修改者	动作	备注
1.0.0	2012-10-6	所有	黄舒志	创建	新建文档
1.0.1	2012-10-9	第 3 章	杨珺	修改	

目录

1.引言

1.1 编写目的

本文档用于描述基于 OSS API 规范接口实现的安卓 SDK 开发包（我们称之为 Aliyunsdk，下文同）的架构和实现细节。是使用 Aliyunsdk 进行安卓上开发的

指南，详细介绍了 Aliyunsdk 的基本概念、实现细节、提供的服务、可用的 API，以及对用户使用方法的举例介绍、最后介绍开发中碰到的问题、对阿里云 OSS API 的一些建议以及后续工作展望；使得有兴趣于该开源 SDK 的开发者能更好的使用和改进它。

本文档的阅读对象包括：使用该安卓 sdk 的开发人员，希望修改或者扩充该 sdk 的开发人员。

1.2 名词解释

基本概念我们基本沿用 OSS API 的概念，使得开发者可以直观的与 OSS API 的功能和接口联系起来。在此基础上，我们抽象出几个和一般用户使用习惯相吻合的概念，使得用户可以方便直观的使用一些我们封装好的功能。

基础的名词解释如下。

命名空间：由阿里云 OSS API 所规定，允许一个用户最多建 10 个命名空间，用户上传文件等操作需要指定某个命名空间，在指定的命名空间里进行操作。

文件：与 Java 的 File 类相似，文件包括普通文件和文件夹。阿里云 OSS API 中基本与之相应的概念是 Object，称之为文件是为了让用户更直观的理解和使用

文件分组：可以对多个文件进行分组，将它们归为一类。不过文件间实质上还是互相独立的个体，如果分组中的某个文件发生改变，那这个分组也就失效了。

以上是一些基本概念的介绍，为了支撑起用户对这些基本概念的操作，我们使用了更多的模型和操作，将在下文设计部分进行详细介绍。

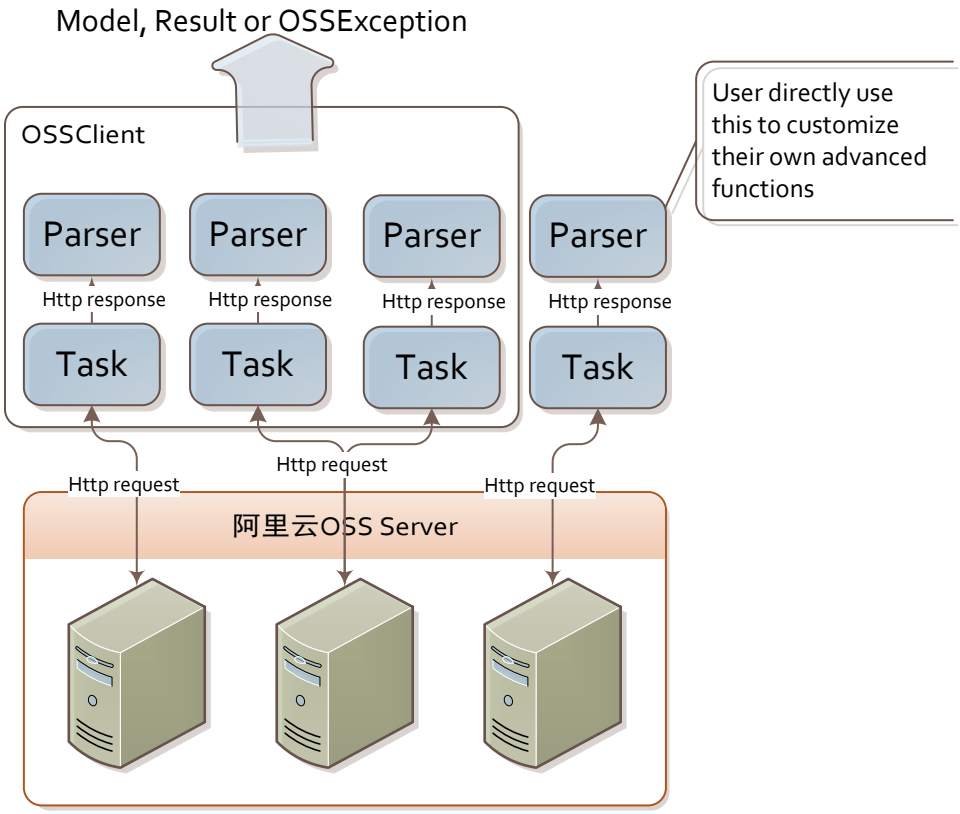
2.总体设计

Aliyunsdk 的设计主要考虑以下几个方面：

1. 与阿里云 OSS API 可以有直观的映射。这样有几个好处：如果阿里云 OSS API 进行改进或者升级，aliyunsdk 不至于更新的时候付出很大的代价；aliyunsdk 目的是让 Android 上的开发者更好的使用阿里云 OSS 的服务，用户实质还是在使用阿里云的服务，因此与阿里云 OSS API 直观映射能够让有需求的用户更好的去扩展开发出自己的自定义功能。
2. 提供的概念和功能更符合用户的使用习惯。这一点貌似与第一条考虑有些矛盾，因为阿里云所提出的 object、marker、prefix 等概念貌似对于普通用户来说不那么直观。但其实如果注意到这其实是两种需求的场景，问题也就能得到解决了。

3. 使用方便。SDK 就是为了使用户使用起来更方便，因此这是一个基本的要求。这就要求概念清晰，提供的 API 简单有效。
4. 易于阅读和扩充。开源软件的代码需要方便别人进行修改和扩展，因此要求架构要清晰合理易于扩充。

最终，Aliyunsdk 的整体结构如下图：



整个结构主要分成三层对 OSS API 进行封装，它们是 Task, Parser 和 OSSClient，再加上模型的定义。这个设计与上文提到的总体考虑是相吻合的。

整个过程基本分为如下步骤：调用 task 发起 http 请求，获取 HttpResponse –> 由 xmlparser 解析 HttpResponse，获得相应的返回结果或者抛出异常 OSSException。外围在由 OSSClient 去封装一些常用的功能，比如创建文件夹、创建文件等。

其中 Task 对阿里云 OSS API 进行了封装，OSS API 提供的每个功能基本上都作为一个 Task，在里面发起 Http 请求，获得 HttpResponse。如果阿里云 OSS API 发生了改动，只须改动对应的 task 即可。

Parser 负责解析 HttpResponse。将对 HttpResponse 的解析单独分离出来，这一步骤是联系底层 http 请求和软件模型的关键点。

OSSClient 支撑起用户的基本操作习惯，对阿里云 OSS 的基本使用就可以方便的通过 OSSClient 来实现。

而且用户也可以绕过 OSSClient，直接使用 Task 和 Parser，可以去开发一些 OSS API 所支持的自定义功能。

具体的实现参见下文“详细设计”部分。

3.详细设计

3.1 模型

模型包(com.aliyun.android.oss.model)下定义了 aliyunsdk 用到的一些数据结构，其中包括用户需要直接接触到的一些基本概念以及为了支撑起功能的一些辅助模型。

3.1.1 基本概念

1. Bucket：命名空间
2. OSSObject：文件（文件夹）
   - 描述

     与 Java 的 File 类定义类似，既可以表示文件夹、也可以表示文件，通过 isDirectory()来进行区分。其中文件夹采用不含数据、命名以”/”结束的 Object 进行表示。
   - 解释

     OSS API 原本只有 Object 的概念，并没有文件和文件夹的概念。API 文档里大致介绍了两种方式可以引入文件夹的概念，一个是使用 prefix，object 的 prefix 可以直接视作是文件夹；另一个是创建一个空的 object 来表示文件夹。出于更好的支持空文件夹等考虑，我们采用的是后一种方法。
3. ObjectGroup：文件分组
   - 描述

     可以对多个文件进行分组，将它们归为一类。不过文件间实质上还是互相独立的个体，如果分组中的某个文件发生改变，那这个分组也就失效了。
4. MetaData：请求的元数据
   - 描述

     包括 OSS 定义的一些 Http 请求的元数据以及用户自定义的元数据。其子类目前包括表示用户信息元数据的 UserMetaData、表示文件分组元数据的 ObjectGroupMetaData 和表示文件自定义的元数据，其中后者可以加入以 x-oss-meta-为前缀的元数据，用户可以用此来对文件格式或者内容进行标注。
5. Part：文件块

- 描述

  为了支持分块上传等功能，将文件分成多个文件块。每个文件块包含 etag, partNumber, uploadId, data 等属性。

6. AccessLevel：访问权限
   - 描述

     从 OSS API 文档来看，目前只应用于 Bucket 的可见性上。有 Private, Public-read, public-read-write 三种权限。
7. ObjectsQuery：文件信息查询器
   - 描述

     用于查询 bucket 下或者某个目录下的子文件和子文件夹信息，以分页器的形式辅助展示。具体信息见分页器的介绍。
8. PageMarker：分页器页码
   - 描述

     分页器的文件页码标识，实质上是 OSS API 请求时加上的 Marker 信息，表示从哪个 marker 之后开始罗列。有向前和向后的 Marker 链接，可以索引到分页器的前一页和后一页。
9. ListPartsQuery：文件块查看器
   - 描述

     与 ObjectsQuery 类似。

3.1.2 辅助模型

1. Range：区间范围
2. GetBucketXmlObject：Get Bucket 请求返回的结果
3. DeleteMultipleObjectResult：Delete Multiple Object 请求返回的结果
4. HttpResponseError：阿里云服务器返回的错误信息
5. ListPartXmlObject: List Part 请求返回的结果
6. ListMultipartUploadXmlObject：List MultipartUpload 请求返回的结果

以上介绍了大部分主要的 model，有的没有介绍的地方可能是目前没有用到或者相对比较不重要的，如果想知道更详细的信息，可以自行阅读代码或者本 SDK 的 API 文档。

3.2 Task

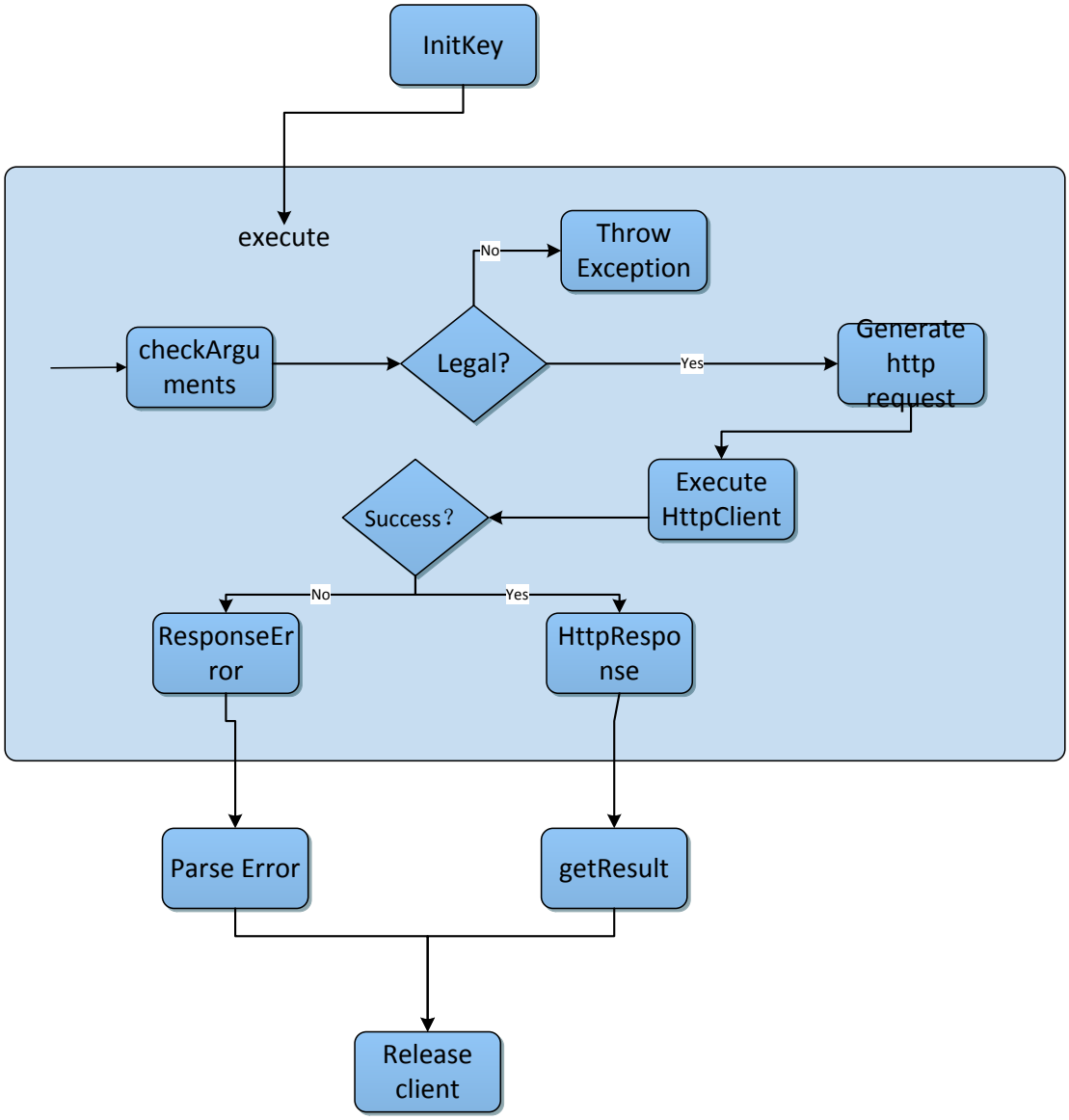
一般来说一个 Task 对应了一个 OSS API，比如 Delete Object 的功能对应到的就是 DeleteObjectTask。每个 Task 里构造了一个 Http 请求，并发起这个请求，获得来自服务器的 HttpResponse。

所有各式各样的 task 均继承自抽象基类 Task。Task 定义了一些基本方法，其子类一般只是对这些方法有些小的改动或者增添，这些基本方法如下：

1. initKey：设置 accessKey 和 accessId
2. checkArguments：检查参数的合法性，如果参数不正确，则抛出 IllegalArgumentException，每次执行前都进行一遍检查
3. execute：构造并发起 http 请求，返回 HttpResponse
4. generateHttpRequest：构造 http 请求
5. getResponseError：获取阿里云服务器返回的错误信息
6. releaseHttpClient：释放 httpClient，每次执行后都需要释放，否则系统资源将很快被用光。

此外大部分 Task 的子类还实现了 getResult 方法，用于返回每个 task 执行后的结果，比如 bool 值用来判断操作是否成功，或者是返回由 xml 解析出来的模型。

一般 task 的运作流程如下图所示：



即首先设置 AccessId 和 AccessKey，有了访问权限后执行 execute 方法，在该方法里面首先检查参数是否正确，如果没有问题的话构造 http 请求并发送请求，最后返回由 parser 去处理返回的 httpResponse（可能是服务器返回的错误信息）。

3.3 HttpResponse 解析

3.3.1 模型解析

BucketListXmlParser 解析 GetServiceTask 操作返回的 xml

CopyObjectXmlParser 解析 CopyOjbect 操作返回的 xml

GetBucketAclXmlParser 解析 GetBucketAcl 操作返回的 xml

GetBucketObjectsXmlParser 解析 GetBucketObjects 操作返回的 xml

GetObjectGroupIndexXmlParser 解析 GetObjectGroupIndex 操作返回的 xml
ListMultipartUploadXmlParser 解析 ListMultipartUpload 操作返回的 xml
ListPartXmlParser 解析 ListPartXml 操作返回的 xml
MultipartUploadCompleteXmlParser 解析 MultipartUploadComplete 操作返回的 xml
MultipartUploadInitXmlParser 解析 MultipartUploadInit 操作返回的 xml
PostObjectGroupXmlParser 解析 PostObjectGroup 操作返回的 xml
以上均继承自 AbstractXmlParser

3.3.2 错误解析

HttpResponseErrorParser 解析由阿里云服务器返回的错误信息 xml

3.4 命名空间、文件、文件夹操作

在模型部分已经介绍过命名空间、文件、文件夹和文件分组的概念和表示了，这部分的功能均在 OSSClient 里有直接的 API 可以调用。

3.4.1 命名空间

对命名空间支持以下操作：

- 获取用户 Bucket 列表
- 创建 bucket
- 删除 bucket
- 获取 bucket 可见性
- 更改 bucket 可见性
- 查看 bucket 内容

这部分的功能都比较简单，一般一个功能用一个 task 就能实现。其中查看 bucket 内容和浏览文件夹下的内容的实现方法是基本一致的，其实也可以把命名空间看做是一个特殊的文件夹。

3.4.2 文件和文件夹操作

对文件和文件夹支持如下基本操作：

- 创建、删除文件夹
- 文件夹查看，获取文件夹下的子文件和子文件夹，非递归。文件夹下的内容采用指定 prefix 和 delimiter=/来实现。

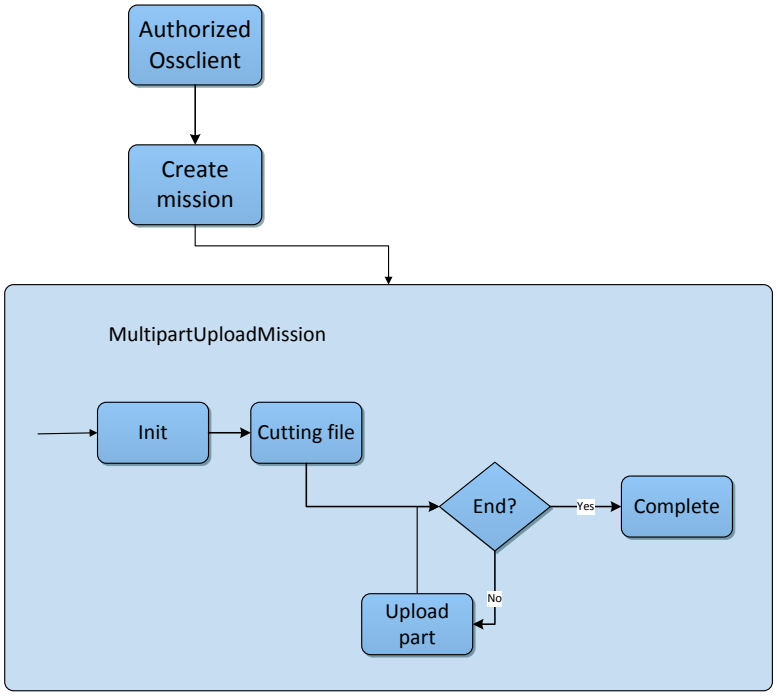
- 删除文件夹下子文件和子文件夹，递归的删除文件夹下的内容。
- 复制文件夹
- 移动文件夹
- 上传、下载、获取文件（保存在内存中）
- 删除文件
- 复制文件
- 移动文件
- 获取文件摘要信息

3.4.3 文件分组

可以若干个文件合并成为一个文件分组（Object Group），注意这些文件必须在同一个 Bucket 下。

3.5 分块上传

对于大文件，支持对文件的分块上传。分块上传的设计是由 OSSClient 去生成一个分块上传任务 MultipartUploadMission，接着由这个对象来负责此次分块上传的全过程，具体流程如下图：



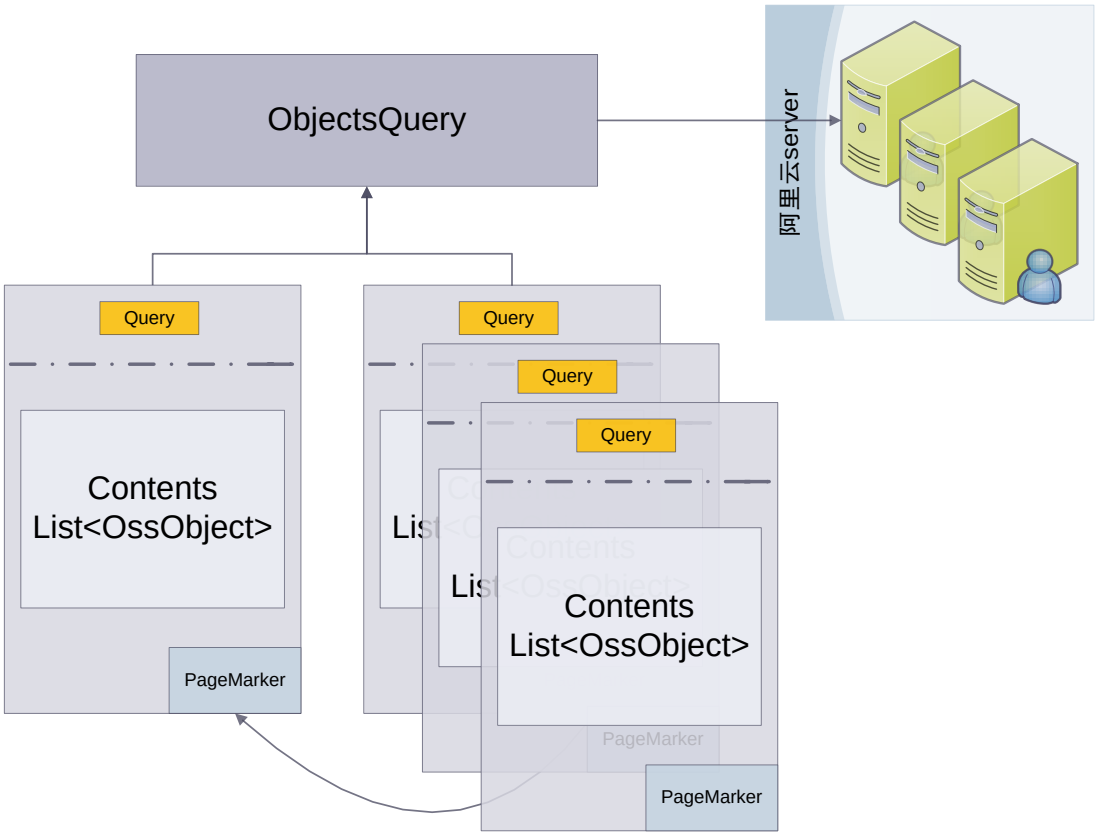
其中创建 MultipartUploadMission 后，首先进行初始化，调用 MultipartUploadInitTask，从服务器那获取一个 uploadId，通过这个 uploadId 去标识一个分块上传任务。接着会对文件进行裁切，根据传入的每个文件块的大小将

文件分成多份，以 List<Part>的方式保存。然后对每个文件块进行上传，全部上传后调用 MultipartUploadCompleteTask 向服务器确认上传完毕，确认时需要附上每一块的摘要信息，确认成功后才真正完成一个大文件的上传，否则服务器那不会建立相应的 object。

3.6 分页读取

某个 bucket 或者文件夹下可能有很多的子文件和子文件夹，当用户执行一次查看 bucket 或者某个文件夹下的内容时，可能一次无法完全返回（比如超过了 OSS API 规定的上限 1000 或者是用户自己定义了 max-keys），这个时候就需要分页显示。分页的方法在文档里有说明可以使用 marker 来实现。

我们设计了分页器的模式以方便用户分页查看，具体设计如下图：



其中由 OSSClient 去生成一个 ObjectsQuery 对象，在该对象中记录了访问的资源和每次访问的返回数等信息，ObjectsQuery 对象生成一个 Pagination 分页器对象，并调用 GetBucketTask 发起 Http 请求填充该 Pagination 对象的 Contents。

每个 Pagination 对象如上图所示均有一个页码标记 PageMarker，每个 PageMarker 记录了当前请求的 Marker 值，并且通过链接指向了其前驱和后继页

码，这样便可实现前翻和后翻。每次前翻和后翻会通过 query(ObjectsQuery)去重新发送请求，而不是将请求过的缓存在内存中，以免造成大量的内存消耗。

3.7 压缩和加密

支持对文件进行压缩和加密，使用到的主要类如下：

- PutZipEncObjectTask

  放置两个开关 isZipped 和 isEncrypted，控制是否压缩及是否加密，如果加密的话还需要注入 byte[] key。

- CompressUtil

  压缩的工具类。支持使用 zip 方法进行压缩。提供了以下接口：

  - byte[] zipBytes(byte[] source)
  - byte[] unzipBytes(byte[] source)

- CipherUtil:

  加密的工具类。支持的加密方式有 DES, AES, 三重 DES 和 IDEA 等对称加密方式。因为非对称加密开销大，一般用于加密密钥等内容较少的重要信息，因此用于加密文件并不实用，而且用户目前来看也不会有这方面的需求，因此暂时先不加入非对称加密的功能。提供了以下接口：

  - Key generateKey(CipherAlgorithm algorithm)
  - Key toKey(byte[] key, CipherAlgorithm algorithm)
  - byte[] encrypt(byte[] data, byte[] key, CipherAlgorithm algorithm)
  - byte[] decrypt(byte[] data, byte[] key, CipherAlgorithm algorithm)

如果一个文件同时想压缩和加密的话，流程如下：

压缩 → 加密 → 上传文件 → 下载文件 → 解密 → 解压缩

因为加密会使得文件熵增，失去可以压缩的特性，所以应该先压缩然后进行加密。同样逆过程就得先解密然后解压缩。

同时在 OSSClient 里也封装了相应的 API 可供用户更方便的使用压缩加密功能。具体在用户文档里有详细的介绍。

3.8 异常

根据 OSS API 文档中对错误响应的描述，我们将服务器返回的错误信息封装到[OSSException](ossexception)对象中。

所有的错误消息体都包括一下几个元素：

- Code: OSS 返回给用户的错误码（附表中有错误码详细信息）
- Message: OSS 给出的详细错误信息
- RequestID: 用于唯一标识该次请求的 UUID；当你无法解决问题时，可以凭这个 RequestId 来请求 OSS 工程师的帮助
- HostId: 用于标识访问的 OSS 集群（目前统一为 oss.aliyun.com)

对于 OSS 服务器返回的错误信息，我们通过设置这些域的值来抛出异常。为了统一处理方便，我们对于其他真正的系统异常(Exception)，我们同样也作为 OSSException 抛出，但是为了区分，对于这类异常我们将其 Code 字段设置为常量 `OSSException.NON_OSS_ERROR`。

在使用我们提供的 Task, OSSClient 的过程中，需要统一 catch OSSException，所以的错误信息都封装到这个异常中，而不再需要捕获其他异常了。

4.使用示例和注意事项

建议使用 OSSClient 作为入口，OSSClient 里提供的 API 支持了用户经常会使用到的一些常规操作；如果需要自定义更高级的功能的话，可以自行使用 Task 去封装方法。

使用时注意安卓应用程序的权限配置，比如联网、文件读写权限。

具体的使用方法和注意事项参见用户文档。

5.总结

5.1 实现中遇到的问题

1. 感觉 Object Group 可能实际用处不太大？文件分组可以用文件夹，而且 Group 里只要有一个 Object 改变这个 Group 就失效了。并且下载一个 Group 的时候是把该组里面的所有 Object 的数据都拼成一个大的文件，那么这个文件貌似也没什么用了，比如有可执行文件，有文本文件等，下载下来的内容是他们全部拼在一起，那这个拼在一起的大文件貌似也没有太大的价值？

2. 失效的信息会在服务器上保留多久？比如 init 一个 Multipart Upload 事件，但是不 complete 也不 abort，那么这个 key 会保留多久？如果一直保留着，在 List Multipart Uploads 执行后返回的结果里就会被这样的 key 充斥着。同样失效的 Object Group 会保存多久。