***开源项目的对比***

这篇博文主要就是针对平常使用到的框架做一个整理和分析其优劣。

为了从整体上进行把握，先来看看一个完整的APP整体架构

# 1.APP的整体架构

从较高的层次将，一个APP的整体架构可以分为两层，即应用层和基础框架层。

应用层专注于行业领域的实现，例如金融、支付、地图导航、社交等，它直接面向用户，是用户对产品的第一层感知。

基础框架层专注于技术领域的实现，提供APP公有的特性，避免重复制造轮子，它是用户对产品的第二层感知，例如性能、稳定性等。

一个理想的APP架构，应该拥有如下特点：

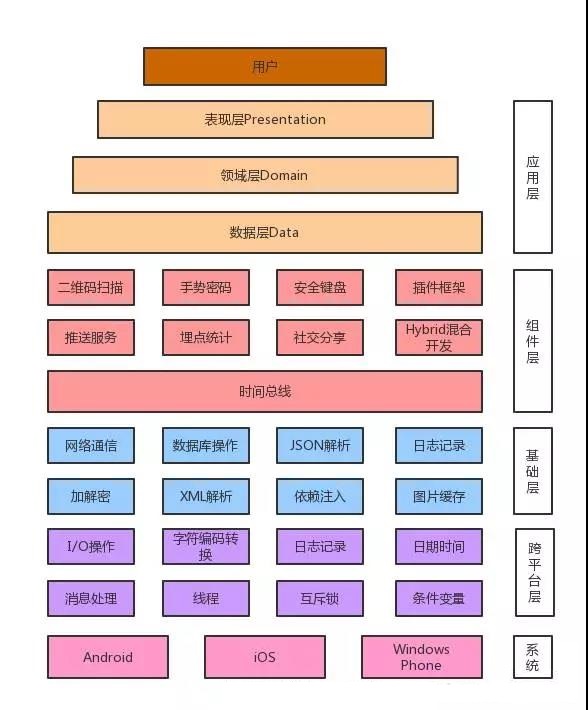
1、支持跨平台开发

2、具有清晰的层次划分，同一层模块间充分解耦，模块内部符合面向对象设计六大原则

在功能、性能、稳定性等方面达到综合最优

基于以上设计原则，我们可以看出APP架构图，最上层是应用层，应用层以下都属于基础框架层，基础框架层包括：组件层、基础层和跨平台层。

我们要讨论的重点是基础层，下面开始一步一步地阐述如何基于开源函数库搭建属于自己的一个基础技术堆栈。



# 2. 技术选型的考量点

首先要明确的是，我们选择开源函数库或者第三方SDK、一般需要综合考虑一下几个方面

特性：提供的特性是否满足项目的需求

可用性，是否提供了简洁便利的API，方便开发者集成使用。

性能：性能不能太差，否则项目后面性能优化会过不去，可能回出现需要替换函数库的情况。

文档：文档应该比较齐全，且可读性高。

技术支持：遇到问题或者发现BUG，是否能够及时得到官方的技术支持是很重要的

大小：引入函数库会增加APK的大小，需要慎重抉择

方法数：如果函数库方法数太多，积累起来会导致你的APP遇到64K问题，应该尽量避免

# 3. 日志记录能力

日志记录无论在服务端开发还是移动端开发，都是一个基础且重要的能力，开发人员在代码调试以及错误定位过程中，大多说都要依赖日志信息，一个简洁灵活的日志记录模块是相当重要的。

## 1、Logger

Logger 是基于系统Log类基础上进行的封装，但新增了如下超赞的特性。

<https://github.com/orhanobut/logger>

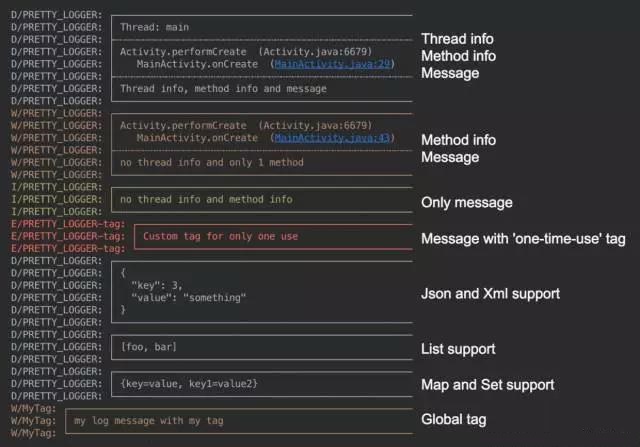
在Logcat中完美的格式化输出，再也不用担心和手机其他APP或者系统的日志信息相混淆了

包含线程、类、方法信息，可以清楚地看到日志记录的调用堆栈

支持跳转到源码处

支持格式化输出JSON、XML格式信息

Logcat截图



当然Logger也不是完备的，它虽然支持格式化输出JSON、XML，但并不支持诸如List、Set、Map和数组等常见Java集合类的格式化输出。

如何解决呢？可以看下LogUtils这个开源库，它实现了Logger缺失的上述特性。

<https://github.com/pengwei1024/LogUtils>

再者，Logger只支持输出日志到Logcat，但项目开发中往往还存在将日志保存到磁盘上的需求，如何将两者结合起来呢？这次就用到了timber 。

<https://github.com/JakeWharton/timber>

## 2、Hugo

<http://www.jianshu.com/p/1e65aa223b59>

Hugo是JakeWharton大神开发的一个通过注解触发的Debug日志库。它是一个非常好的AOP框架，在Debug模式下，Hugo利用aspectj库来进行切面编程，插入日志代码。通过分析Hugo的代码能够对gradle以及aspectj有一个非常好的了解。

<http://blog.csdn.net/xxxzhi/article/details/53048476>

## 3、Timber

timber是JakeWharton开源的一个日志记录库，它的特点是可扩展的框架，开发者可以方便快捷的集成不同类型的日志记录方式，例如，打印日志到Logcat、打印日志到文件、打印日志到网络等，timber通过一行代码就可以同时调用多种方式。

timber的思想很简单，就是维护一个森林对象，它由不同类型的日志树组合而成，例如，Logcat记录树、文件记录树、网络记录树等，森林对象提供对外的接口进行日志打印。每种类型的树都可以通过种植操作把自己添加到森林对象中，或者通过移除操作从森林对象中删除，从而实现该类型日志记录的开启和关闭。

最终我们的日志记录模块将由timber+Logger+LogUtils组成，当然轮子找到了，轮子的兼容合并就得靠我们自己实现了，同时我们还得增加打印到文件的日志树和打印到网络的日志树实现。

# 4.网络解析能力

## 4.1、Json解析

移动互联网产品与服务器端通信的数据格式，如果没有特殊需求的话，一般都使用JSON格式。Android系统也原生的提供了JSON解析的API，但是它的速度非常慢，而且没有提供简洁方便的接口来提高开发者的效率和降低出错的可能。所以我们就开始找第三方开源库来实现JSON解析，比较优秀的包括如下几种。

### 4.1.1 gson

gosn是Google出品的JSON解析函数库，可以将JSON字符串反序列化对应的Java对象，或者反过来将Java对象序列化为对应的JSON字符串，免去了开发者手动通过JSONObject和JSONArray将JSON字段逐个进行解析的烦恼，也减少了出错的可能性，增强了代码的质量。使用gson解析时，对应的Java实体类无需使用注解进行标记，支持任意复杂Java对象包括没有源代码的对象。

### 4.1.2 jackson

jcakson是Java语言的一个流行的JSON函数库，在Android开发中使用时，主要包含三部分。

jackson-core：JSON流处理核心库

jackson-databind：数据绑定函数库，实现Java对象和JSON字符串流的相互转换。

jackson-annotations：databind使用的注解函数库

由于jackson是针对Java语言通用的JSON函数库，并没有为Android优化定制过，因此函数保重包含很多非必要的API，相比其他的JSON函数库，用于Android平台会更显著的增大最终生成的APK的体积。

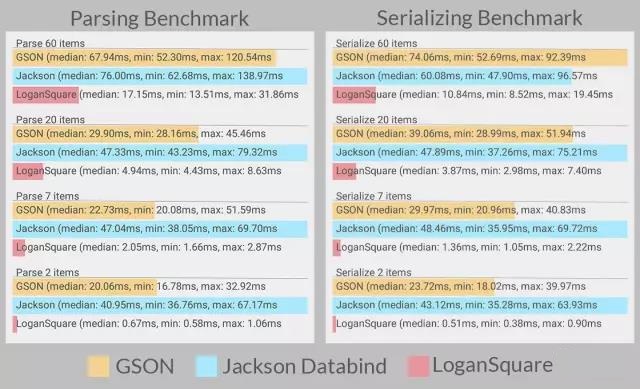
### 4.1.3 Fastjson

Fastjson是阿里巴巴出品的一个Java语言编写的高性能且功能完善的JSON函数库。它采用一种“假定有序快速匹配”的算法，把JSON Parse的性能提升到极致，号称是目前Java语言中最快的JSON库。Fastjson接口简单易用，已经被广泛使用在缓存序列化、协议交互、Web输出、Android客户端等多种应用场景。

由于是Java语言通用的，因此，以前在Android上使用时，Fastjson不可避免的引入了很多对于Android而言冗余的功能，从而增加了包大小，很多人使用的就是标准版的fastjson，但事实上，fastjson还存在一个专门为Android定制的版本---fastjson.android （https://github.com/alibaba/fastjson/wiki/Android%E7%89%88%E6%9C%AC）。和标准版本相比，Android版本去掉了一些Android虚拟机dalvik不支持的功能，使得jar更小。

### 4.1.4 LoganSquare

LoganSquare是近两年崛起的快速解析和序列化JSON的Android函数库，其底层基于jackson的streaming API，使用APT(Android Annotation Tool)实现编译时注解，从而提高JSON解析和序列化的性能。官网上可以看到LoganSquare和gson、jackson databind的性能对比。



从性能方面看，LoganSquare是完胜gson和jackson的。如果和fastjson相比较，两者应该是不相上下的。

再来看下jar包的大小

gson：232KB

jackson：259+47+1229 = 1.5M

Fastjson：417KB

Fastjson.android：256KB

LoganSquare：48+259 = 307KB

从性能和包大小综合考虑，最终我们会选择Fastjson.android作为基础技术堆栈中的JSON解析和序列化库。

## 4.2、Html解析

### 4.2.1 HtmlPase

### 4.2.2 Jsoup

# 5. 数据库操作能力

无论是iOS还是Android，底层数据库都是基于开源的SQLite实现，然后在系统层封装成用于应用层的API。虽然直接使用系统的数据库API性能很高，但是这些API接口并不是很方便开发者使用，一不小心就会引入Bug，而且代码的视觉效果也不佳。为了解决这个问题，对象关系映射（ORM）框架出现了，比较好的有ActiveAndroid，ormlite和greenDAO。

## 5.1 ActiveAndroid

ActiveAndroid是一种Active Record风格的ORM框架，Active Record（活动目录）是Yii,Rails等框架中对ORM实现的典型命名方式。它极大的简化数据库的使用，使用面向对象的方式管理数据库，告别手写SQL的历史。每一个数据库表都可以被映射为一个类，开发者只需使用类似save()或者delete()这样的函数即可。

不过ActiveAndroid已经基本上处于维护阶段了，最新的一个Release版本是在2012年发布的。

## 5.2 ormlite

ormlite是Java平台的一个ORM框架，支持JDBC连接、Spring和Android平台。在Android中使用时，它包含两部分。

ormlite-core：核心模块，无论在哪个平台使用，都必须基于这个核心库，是实现ORM映射的关键模块。

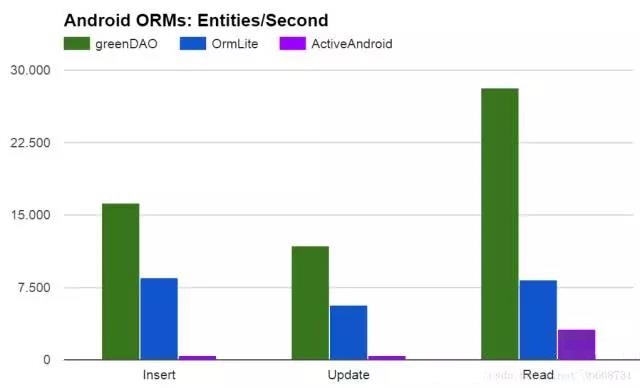
ormlite-android：基于ormlite-core封装的针对Android平台的适配器模块，Android开发中主要跟这个模块打交道。

与ActiveAndroid类似，ormlite也已经不是一个活跃的开源库，最近一次Release版本是在2013年发布的。

## 5.3 greenDAO

greenDAO是一个轻量级且快速的ORM框架，专门为Android高度优化和定制，它能够支持每秒数千条记录的CRUD操作。

官网上给出一张性能对比图



纵轴表示每秒执行的操作数。而且greenDAO处在高度活跃中，最新Release版本是在2017年3月份发布的

## 5.4 Realm

Realm是一个全新的移动数据库引擎，它既不是基于iOS平台的Core Data，也不是基于SQLite，它拥有自己的数据库存储引擎，并实现了高效快速的数据库构建操作，相比Core Data和SQLite，Realm操作要快很多，跟ORM框架相比就更不用说了。

Realm的好处如下：

跨平台：Android和iOS已经是事实上的两大移动互联网操作系统，绝大多数应用都会支持这两个平台。使用Realm，Android和iOS开发者无需考虑内部数据的架构，调用Realm提供的API即可轻松完成数据的交换。

用法简单：相比Core Data和SQLite所需的入门知识，Realm可以极大降低开发者的学习成本，快速实现数据库存储功能。

可视化操作：Realm为开发者提供了一个轻量级的数据库可视化操作工具，开发者可以轻松查看数据库中的内容，并实现简单地插入和删除等操作。

我们看下上述四种数据库包大小。

activeandroid：40KB

greendao：100KB

ormlite-android：57KB

realm-android：4.2M

可以看出，前三个还是正常范围，但Realm的大小一般项目可能无法接受。这是因为不同CPU架构平台的 .so 文件增加了整个包的大小，由于arm平台的so在其他平台上面能够以兼容模式运行的，虽然会损失性能，但是可以极大地减少函数库占用的空间。

因此，可以选择只保留armeabi-v7a和x86两个平台的 .so 文件，直接删除无用的 .so 文件，或者通过工程的build.gradle文件中增加 ndk abi 过滤，语句如下：

android {

...

defaultConfig {

...

ndk {

abiFilters "armeabi-v7a", "x86"

}

}

}

因此，综合性能考虑，包大小以及开源库的可持续发展等因素，我们最终选择greenDAO。

## 5.5 Sugar

## 5.6 SQLBrite

# 6. 网络通信能力

现在的APP几乎都需要从服务器获取数据，不可避免的需要具备网络通信的能力，否则就是一个死界面。

## 6.1 android-async-http

Android最经典的网络异步通信函数库，它对Apache的HttpClient API的封装使得开发者可以简洁优雅地实现网络请求和响应，并且同时支持同步和异步请求。主要特性如下：

支持异步HTTP请求，并在匿名回调函数中处理响应

在子线程中发起HTTP请求

内部采用线程池来处理并发请求

通过RequestParams类实现GET/POST参数构造

无需第三方库支持即可实现Multipart文件上传

库的大小只有60KB

支持多种移动网络环境下自动智能的请求重试机制

HTTP响应中实现自动的gzip解码，实现快速请求响应

内置多种形式的响应解析，有原生的字节流、String、JSON对象，甚至可以将response写入到文件中。

可选的永久cookie保存，内部实现使用的是Android的SharedPreferences。

但是在6.0之后，系统对开发者隐藏了HttpClient函数库，这显著增大了使用android-async-http的代价。 如果铁了心想继续使用HttpClient，官方推荐的做法是在编译期引入org.apache.http.legacy 这个库，库目录在Android SDK目录下的platforms\android-23\optional中找到，它的作用是确保在编译时不会出现找不到HttpClient相关API的错误，在应用运行时可以不依赖这个库，因为6.0以上的Android系统还没有真正移除HttpClient的代码，只不过API设置为对开发者不可见。

我们查看android-async-http源码发现，需要使用下面这个函数库来替换之前的Apache的HttpClient。

dependencies {

compile 'cz.msebera.android:httpclient:4.3.6'

}

这样显著的增加了APP的包的大小，如果想继续使用android-async-http，那么你的APP需要额外增加1.1MB左右的大小。

## 6.2 OkHttp

OkHttp是一个高效的HTTP客户端，具有如下特性。

支持HTTP/2和SPDY，对同一台主机的所有请求共享同一个socket。

当SPDY不可用时，使用连接池减少请求的延迟。

透明的GZIP压缩减少下载数据大小

缓存响应避免重复的网络请求

OkHttp在网络性能很差的情况下能够很好地工作，它能够避免常见的网络连接问题。如果你的HTTP服务有多个IP地址，OkHttp在第一次连接失败是，会尝试其他可选的地址。这对于IPv4+IPv6以及托管在冗余数据中心的服务来说是必要的。

OkHttp使用现代的TLS特性（SNI,ALPN）初始化HTTP连接，当握手失败时，会降低使用TSL1.0初始化连接。

OkHttp依赖于okio，okio作为java.io和java.nio的补充，是square公司开发的一个函数库。okio使得开发者可以更好地访问、存储和处理数据。一开始是作为OkHttp的一个组件存在的，当然我们也可以单独使用它。

使用Okhttp需要引入Jar包，包的大小为：326+66 = 392KB

## 6.3 Volley

Volley是Google在2013年发布的用于Android平台的网络通信库，能使网络通信更快、更简单、更健壮。官网配出一张弓箭发射图来说明Volley特别使用于数据量小等通信频繁的场景。

具体的将，Volley是为了简化网络任务而设计的，用于帮助开发者处理请求、加载、缓存、多线程、同步等任务。

Volley设计了一个灵活的网络栈适配器，在Android2.2及之前的版本中，Volley底层使用Apache HttpClient，在Android2.3及以上版本中，它使用HttpURLConnection来发起网络请求，而且开发者也很容易将网络栈切换成使用OkHttp。

Volley 官方源码托管在Google Source上面，使用时只能直接以Jar包形式引入，如果想在Gradle中使用compile在线引入，可以考虑使用mcxiaoke在Github上面的Volley Mirror，然后再build.gradle中使用如下语句即可。

compile 'com.mcxiaoke.volley:library:1.0.19'

## 6.4 Retrofit

确切的说，Retrofit并不是一个完整的网络请求函数库，而是将REST API转换成Java接口的一个开源函数库，它要求服务器API接口遵循REST规范。基于注解使得代码变得很简洁，Retrofit默认情况下使用GSON作为JSON解析器，使用OkHttp实现网络请求，三者通常配合使用，当然我们也可以将这两者换成其他的函数库。

通过以上分析，HttpURLConnection、Apache HttpClient 和OkHttp封装了底层的网络请求，而android-async-http，Volley和Retrofit是基于前面三者的基础上二次开发而成。

最后看下函数库的大小

android-async-http：106KB+1.1MB = 1.2MB

OkHttp：326KB+66KB = 392KB

Volley：94KB

Retrofit：122KB+211KB = 333KB

## 6.5 HttpURLConnection

# 7. 图片缓存和显示能力

图片缓存函数库有很多非常优秀的，开发人员可以根据需求进行选择。传统的图片缓存方案中设置有两级缓存，分别是内存缓存和磁盘缓存。在Facebook推出的Fresco中，它增加了一级缓存，也就是Native缓存，这极大地降低了使用Fresco的APP出现OOM的概率。

## 7.1 BitmapFun

BitmapFun函数库是Android官方教程中的一个图片加载和缓存实例，对于简单的图片加载需求来说，使用BitmapFun就够了，在早期用的多，现在渐渐退出了实际项目开发的舞台。

## 7.2 Picasso

Picasso是著名的square公司众多开源项目中的一个，它除了实现图片的下载和二级缓存功能，还解决了常见的一些问题。

在adapter中正常的处理ImageView回收和下载的取消

使用尽量小的内存实现复杂的图像变换

在Picasso中，我们使用一行代码即可实现图片下载并渲染到ImageView中。

Picasso.with(context).load(url).into(imageView);

## 7.3 Glide

Glide是Google推荐的用于Android平台上的图片加载和缓存函数库。这个库被广泛应用在Google的开源项目中，Glide和Picasso有90%的相似度，只是在细节上还是存在不少区别。

Glide为包含图片的滚动列表做了尽可能流畅的优化。除了静态图片，Glide也支持GIF格式图片的显示。Glide提供了灵活的API可以让开发者方便地替换下载图片所用的网络函数库，默认情况下，它使用HttpUrlConnection作为网络请求模块，开发者也可以根据自己项目的实际需求灵活使用Google的Volley或者Square的OkHttp等函数库进行替换。

Glide的使用也可以使用一行代码来完成，语句如下

Glide.with(context).load(url).into(imageView);

## 7.4 Fresco

Fresco是Facebook开源的功能强大的图片加载和缓存函数库，相比其他图片缓存库，Fresco最显著的特点是具有三级缓存：两级内存缓存和一级磁盘缓存。主要特性如下：

渐进式地加载JPEG图片

显示GIF和WebP动画

可扩展，可自定义图片加载和显示

在Android 4.X及以下的系统上，将图片放在Android内存一个特殊的区域，从而使得应用运行更流畅，同时极大减低出现OutOfMemoryError的错误。

## 7.5 Android-Universal-Image-Loader

Android-Universal-Image-Loader简称UIL，是Android平台老牌的图片下载和缓存函数库，功能强大灵活且高度可自定义，它提供一系列配置选项，并能很好地控制图片加载和缓存的过程。使用者甚多，现在项目仍在使用。UIL也支持二级缓存，特性如下：

同步或异步的多线程图片加载

高度可自定义：线程池、下载器、解码器、内存和磁盘缓存、图片显示选项等。

每张图片的显示支持多种自定义选项：默认存根图片、解码选项、Bitmap处理和显示等。

图片可缓存在内存或者磁盘（设备的文件系统或者SD卡）上。

可实时监听图片加载流程，包括下载进度。

最后看下几个库的包大小

BitmapFun：71KB

Picasso：120KB

Glide：475KB

Fresco：47KB+93KB+93KB+10KB+3MB+62KB+8KB+111KB = 3.4MB

Android-Universal-Image-Loader：162KB

图片函数库的选择需要根据APP的具体情况而定，对于严重依赖图片缓存的APP,例如壁纸类，图片社交类APP来说，可以选择最专业的Fresco。对于一般的APP，选择Fresco会显得比较重，毕竟Fresco 3.4MB的体量摆在这。

根据APP对图片显示和缓存的需求从低到高我们可以对以上函数库做一个排序

BitmapFun < Picasso < Android-Universal-Image-Loader

< Glide < Fresco

值得一提的是，如果你的APP计划使用React Native进行部分模块功能的开发的话，那么在基础函数库选择方面需要考虑和React Native的依赖库的复用，这样可以减少引入React Native 所增加的APP的大小，可以复用的函数库有：OkHttp，Fresco,jackson-core.

## 7.6 Picasso-transformations

## 7.7 Glide-transformations

## 7.8 Android-Gpuimage

# 8.缓存

## 8.1 DiskLruCahe（Lru磁盘缓存）

## 8.2 三级缓存

# 9.依赖注入

## 9.1 ButterKnife

## 9.2 Dagger2

## 9.3 AndroidAnotations

## 9.4 RoboGuice

# 10.图表

## 10.1 WilliamChart

## 10.2 HelloCharts

## 10.3 MPAndroidChart

# 11.后台处理

## 11.1 tape

## 11.2 Android Priority Job Queue

# 12.响应式编程

## 12.1 RxJava

## 1.2.2 RxJavaJoins

## 12.3 RxAndroid

## 12.4 RxBinding

## 12.5 Agera

# 13.测试框架

## 13.1 Mockito

## 13.2 Robotium

## 13.3 Robolectric

# 14.调试框架

## 14.1 Stetho

# 15.性能优化

## 15.1 内存泄露检测（LaekCanary）

## 15.2 崩溃报告（ACRA）

# 16.事件总线

## 16.1 EventBus

## 16.2 Otto

参考书籍：《Android高级进阶》顾浩鑫