Android 开发规范与应用(一)

开发规范

摘要

• 使用 Gradle 和它推荐的工程结构
• 把密码和敏感数据放在gradle.properties
• 不要自己写 HTTP 客户端,使用Volley或OkHttp库
• 使用Jackson库解析JSON数据
• 避免使用Guava同时使用一些类库来避免65k method limit（一个Android程序中最多能执行65536个方法）
• 使用 Fragments来呈现UI视图
• 使用 Activities 只是为了管理 Fragments
• Layout 布局是 XMLs代码，组织好它们
• 在layoutout XMLs布局时，使用styles文件来避免使用重复的属性
• 使用多个style文件来避免单一的一个大style文件
• 保持你的colors.xml 简短DRY(不要重复自己)，只是定义调色板
• 总是使用dimens.xml DRY(不要重复自己)，定义通用常数
• 不要做一个深层次的ViewGroup
• 在使用WebViews时避免在客户端做处理，当心内存泄露
• 使用Robolectric单元测试，Robotium 做UI测试
• 使用Genymotion 作为你的模拟器
• 总是使用ProGuard 和 DexGuard混淆来项目

Android SDK

将你的Android SDK放在你的home目录或其他应用程序无关的位置。
当安装有些包含SDK的IDE的时候，可能会将SDK放在IDE同一目录下，当你需要升级（或重新安装）IDE或更换的IDE时，会非常麻烦。
此外，若果你的IDE是在普通用户，不是在root下运行，还要避免吧SDK放到一下需要sudo权限的系统级别目录下。

构建系统

你的默认编译环境应该是Gradle.
Ant 有很多限制，也很冗余。使用Gradle，完成以下工作很方便：
• 构建APP不同版本的变种
• 制作简单类似脚本的任务
• 管理和下载依赖
• 自定义秘钥
• 更多

同时，Android Gradle插件作为新标准的构建系统正在被Google积极的开发。

工程结构

有两种流行的结构：老的Ant & Eclipse ADT 工程结构，和新的Gradle & Android Studio 工程结构，
你应该选择新的工程结构，如果你的工程还在使用老的结构，考虑放弃吧，将工程移植到新的结构。

- **老的结构**

```
old-structure
├ assets
├ libs
├ res
├ src
|  └ com/futurice/project
├ AndroidManifest.xml
├ build.gradle
├ project.properties
└ proguard-rules.pro
```

- **新的结构**

```
new-structure
├ library-foobar
├ app
|  ├ libs
|  ├ src
|  |  ├ androidTest
|  |  |  └ java
|  |  |     └ com/futurice/project
|  |  └ main
|  |     ├ java
|  |     |  └ com/futurice/project
|  |     ├ res
|  |     └ AndroidManifest.xml
|  ├ build.gradle
|  └ proguard-rules.pro
├ build.gradle
└ settings.gradle
```

主要的区别在于，新的结构明确的分开了’source sets’ (main,androidTest)，Gradle的一个理念。
你可以做到，例如，添加源组‘paid’和‘free’在src中，这将成为您的应用程序的付费和免费的两种模式的源代码。

你的项目引用第三方项目库时（例如，library-foobar），拥有一个顶级包名app从第三方库项目区分你的应用程序是非常有用的。
然后settings.gradle不断引用这些库项目，其中app/build.gradle可以引用。

Gradle 配置

- **常用结构** 参考Google’s guide on Gradle for Android

- **小任务** 除了(shell, Python, Perl, etc)这些脚本语言，你也可以使用Gradle 制作任务。更多信息请参考Gradle's documentation。

- **密码** 在做版本release时你app的 build.gradle你需要定义 signingConfigs.此时你应该避免以下内容：

不要做这个 . 这会出现在版本控制中。

```
signingConfigs {
    release {
        storeFile file("myapp.keystore")
        storePassword "password123"
        keyAlias "thekey"
        keyPassword "password789"
    }
}
```

而是，建立一个不加入版本控制系统的gradle.properties文件。

```
signingConfigs {
    release {
        try {
            storeFile file("myapp.keystore")
            storePassword KEYSTORE_PASSWORD
            keyAlias "thekey"
            keyPassword KEY_PASSWORD
        }
        catch (ex) {
            throw new InvalidUserDataException("You should define
KEYSTORE_PASSWORD and KEY_PASSWORD in gradle.properties.")
        }
    }
}
```

使用 Maven 依赖方案代替使用导入jar包方案 如果在你的项目中你明确使用率jar文件，那么它们可能成为永久的版本，如2.1.1.下载jar包更新他们是很繁琐的，这个问题Maven很好的解决了，这在Android Gradle构建中也是推荐的方法。你可以指定版本的一个范围，如2.1.+,然后Maven会自动升级到制定的最新版本，例如：

```
dependencies {
    compile 'com.netflix.rxjava:rxjava-core:0.19.+'
    compile 'com.netflix.rxjava:rxjava-android:0.19.+'
    compile 'com.fasterxml.jackson.core:jackson-databind:2.4.+'
    compile 'com.fasterxml.jackson.core:jackson-core:2.4.+'
    compile 'com.fasterxml.jackson.core:jackson-annotations:2.4.+'
    compile 'com.squareup.okhttp:okhttp:2.0.+'
    compile 'com.squareup.okhttp:okhttp-urlconnection:2.0.+'
}
```

IDEs and text editors

- **IDE集成开发环境和文本编辑器**
  无论使用什么编辑器，一定要构建一个良好的工程结构 编辑器每个人都有自己的选择，让你的编辑器根据工程结构和构建系统运作，那是你自己的责任。

当下首推Android Studio,因为他是由谷歌开发，最接近Gradle，默认使用最新的工程结构，已经到beta阶段
（目前已经有release 1.0了），它就是为Android开发定制的。

你也可以使用Eclipse ADT ，但是你需要对它进行配置，因为它使用了旧的工程结构和Ant作为构建系统。你甚至可以使用纯文版编辑器如Vim，Sublime Text，或者Emacs。如果那样的话，你需要使用Gardle和adb命令行。如果使用Eclipse集成Gradle
不适合你，你只是使用命令行构建工程，或迁移到Android Studio中来吧。

无论你使用何种开发工具，只要确保Gradle和新的项目结构保持官方的方式构建应用程序，避免你的编辑器配置文件加入到版本控制。例如，避免加入Ant build.xml文件。
特别如果你改变Ant的配置，不要忘记保持build.gradle是最新和起作用的。同时，善待其他开发者，不要强制改变他们的开发工具和偏好。

类库

- **Jackson** 是一个将java对象转换成JSON与JSON转化java类的类库。Gson是解决这个问题的流行方案，然而我们发现Jackson更高效,因为它支持替代的方法处理JSON:流、内存树模型,和传统JSON-POJO数据绑定。不过，请记住，Jsonkson库比起GSON更大，所以根据你的情况选择，你可能选择GSON来避免APP 65k个方法限制。其它选择: Json-smart and Boon JSON

- **网络请求**，缓存，图片 执行请求后端服务器，有几种交互的解决方案，你应该考虑实现你自己的网络客户端。使用 Volley
  或Retrofit。Volley 同时提供图片缓存类。若果你选择使用Retrofit,那么考虑使用 Picasso
  来加载图片和缓存，同时使用OkHttp作为高效的网络请求。Retrofit，Picasso和 OkHttp都是有同一家公司开发（注：
  是由Square 公司开发），所以它们能很好的在一起运行。OkHttp 同样可以和Volley 在一起使用 Volley.

- **RxJava** 是函数式反应性的一个类库，换句话说，能处理异步的事件。
  这是一个强大的和有前途的模式，同时也可能会造成混淆，因为它是如此的不同。
  我们建议在使用这个库架构整个应用程序之前要谨慎考虑。
  有一些项目是使用RxJava完成的，如果你需要帮助可以跟这些人取得联系：
  Timo Tuominen, Olli Salonen, Andre Medeiros, Mark Voit, Antti Lammi, Vera Izrailit, Juha Ristolainen.
  我们也写了一些博客：
  [1], [2],
  [3],
  [4].

如若你之前有使用过Rx的经历，开始从API响应应用它。
另外，从简单的UI事件处理开始运用，如单击事件或在搜索栏输入事件。
若对你的Rx技术有信心，同时想要将它应用到你的整体架构中，那么请在复杂的部分写好Javadocs文档。
请记住其他不熟悉RxJava的开发人员，可能会非常难理解整个项目。
尽你的的全力帮助他们理解你的代码和Rx。

Retrolambda 是一个在Android和预JDK8平台上的使用Lambda表达式语法的Java类库。
它有助于保持你代码的紧凑性和可读性，特别当你使用如RxJava函数风格编程时。
使用它时先安装JDK8，在Android Studio工程结构对话框中把它设置成为SDK路径，同时设置JAVA8_HOME和JAVA7_HOME环境变量，
然后在工程根目录下配置 build.gradle：

```
dependencies {
    classpath 'me.tatarka:gradle-retrolambda:2.4.+'
}
```

同时在每个module 的build.gradle中添加

```
apply plugin: 'retrolambda'

android {
    compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}

retrolambda {
    jdk System.getenv("JAVA8_HOME")
    oldJdk System.getenv("JAVA7_HOME")
    javaVersion JavaVersion.VERSION_1_7
}
```

Android Studio 提供Java8 lambdas表带是代码提示支持。如果你对lambdas不熟悉，只需参照以下开始学习吧：

- **任何只包含一个接口的方法都是”lambda friendly”同时代码可以被折叠成更紧凑的语法**
- **如果对参数或类似有疑问，就写一个普通的匿名内部类，然后让Android Status为你生成一个lambda。**

当心dex方法数限制，同时避免使用过多的类库 Android apps，当打包成一个dex文件时，有一个65535个应用方法强硬限制[1] [2] [3]。
当你突破65k限制之后你会看到一个致命错误。因此，使用一个正常范围的类库文件，同时使用dex-method-counts
工具来决定哪些类库可以再65k限制之下使用，特别的避免使用Guava类库，因为它包含超过13k个方法。

Activities and Fragments
Fragments应该作为你实现UI界面默认选择。你可以重复使用Fragments用户接口来组合成你的应用。我们强烈推荐使用Fragments而不是activity来呈现UI界面，理由如下：

- **提供多窗格布局解决方案** Fragments 的引入主要将手机应用延伸到平板电脑，所以在平板电脑上你可能有A、B两个窗格，但是在手机应用上A、B可能分别充满整个屏幕。如果你的应用在最初就使用了fragments，那么以后将你的应用适配到其他不同尺寸屏幕就会非常简单。

- **屏幕间数据通信** 从一个Activity发送复杂数据(例如Java对象)到另外一个Activity，Android的API并没有提供合适的方法。不过使用Fragment，你可以使用一个activity实例作为这个activity子fragments的通信通道。即使这样比Activity与Activity间的通信好，你也想考虑使用Event Bus架构，使用如Otto 或者 greenrobot EventBus作为更简洁的实现。如果你希望避免添加另外一个类库，RxJava同样可以实现一个Event Bus。

- **Fragments 一般通用的不只有UI** 你可以有一个没有界面的fragment作为Activity提供后台工作。
  进一步你可以使用这个特性来创建一个fragment 包含改变其它fragment的逻辑
  而不是把这个逻辑放在activity中。

- **甚至ActionBar 都可以使用内部fragment来管理** 你可以选择使用一个没有UI界面的fragment来专门管理ActionBar,或者你可以选择使用在每个Fragment中
  添加它自己的action 来作为父Activity的ActionBar.参考.

很不幸，我们不建议广泛的使用嵌套的fragments，因为
有时会引起matryoshka bugs。我们只有当它有意义(例如，在水平滑动的ViewPager在像屏幕一样fragment中)或者他的确是一个明智的选择的时候才广泛的使用fragment。

在一个架构级别，你的APP应该有一个顶级的activity来包含绝大部分业务相关的fragment。
你也可能还有一些辅助的activity ，这些辅助的activity与主activity
通信很简单限制在这两种方法
Intent.setData() 或 Intent.setAction()或类似的方法。

Java 包结构
Android 应用程序在架构上大致是Java中的Model-View-Controller结构。
在Android 中 Fragment和Activity通常上是控制器类
(http://www.informit.com/articles/article.aspx?p=2126865).
换句话说，他们是用户接口的部分，同样也是Views视图的部分。

正是因为如此，才很难严格的将fragments (或者 activities) 严格的划分成 控制器controlloers还是视图 views。
最还是将它们放在自己单独的 fragments 包中。只要你遵循之前提到的建议，Activities 则可以放在顶级目录下。
若果你规划有2到3个以上的activity，那么还是同样新建一个activities包吧。

然而，这种架构可以看做是另一种形式的MVC，
包含要被解析API响应的JSON数据，来填充的POJO的models包中。
和一个views包来包含你的自定义视图、通知、导航视图，widgets等等。
适配器Adapter是在数据和视图之间。然而他们通常需要通过getView()方法来导出一些视图，
所以你可以将adapters包放在views包里面。

一些控制器角色的类是应用程序级别的，同时是接近系统的。
这些类放在managers包下面。
一些繁杂的数据处理类，比如说”DateUtils”,放在utils包下面。
与后端交互负责网络处理类，放在network包下面。

总而言之，以最接近用户而不是最接近后端去安排他们。

```
com.futurice.project
├ network
├ models
├ managers
├ utils
├ fragments
└ views
   ├ adapters
   ├ actionbar
   ├ widgets
   └ notifications
```