# **Android组件化开发实践**

Android项目中代码量达到一定程度，编译将是一件非常痛苦的事情，短则一两分钟，长则达到五六分钟。Android studio推出instant run由于各种缺陷一般情况下是被关闭的……  
组件化开发可以有效降低代码模块的耦合度，使代码架构更加清晰，同时模块化的编译可以有效减少编译时间，当然总的编译时间是不会减少的，只是App模块化之后开发某个模块时，只需要编译特定模块，可以快速编译调试。

## 原理

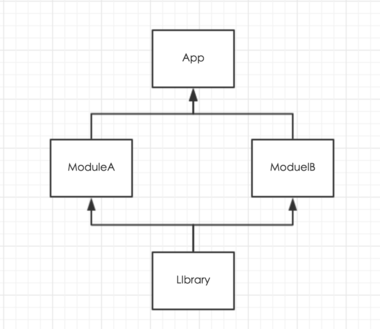
组件化和插件化有些同学有些迷惑，简单来说组件化是在编译期分模块，插件化是在运行期。一般插件化用于动态修复bug或者动态更新模块，相对来说黑科技更多一些。

正常一个App中可以有多个module，但是一般只会有一个module是设置为application的，其他均设置为library，组件化开发就是要每个module都可以运行起来，因此在开发期间每个module均设置为application，发布时再进行合并。

## 实践

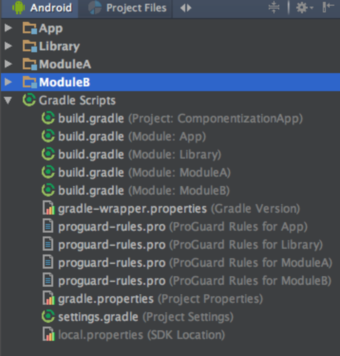
本文主要介绍一下项目组件化开发过程碰到的问题和解决办法，这里以  
[ModularizationApp](https://github.com/wutongke/ModularizationApp" \t "_blank)项目为例。[ModularizationApp](https://github.com/wutongke/ModularizationApp" \t "_blank)是一个组件化的app：

#### 架构



Paste\_Image.png

* 其中App是主application，ModuleA和ModuleB是两个业务模块，Library是基础模块，包含**所有模块需要的依赖库**，以及一些工具类：如网络访问、时间工具等。代码结构如图：



Paste\_Image.png

#### Module作为application开发

ModuleA和ModuleB是相对独立的业务模块，可以分别进行开发，编译时只编译自身，具体实现时在**gradle.properties**中设置变量，如：IsBuildMudle=false  
在模块的的build.gradle中：

if (IsBuildMudle.toBoolean()) {

apply plugin: 'com.android.application'

} else {

apply plugin: 'com.android.library'

}

在主模块的build.gradle中设置：

if (!IsBuildMudle.toBoolean()) {

compile project(':ModuleA')

compile project(':ModuleB')

} else {

compile project(':Library')

}

这样每个module就可以独立安装使用了，注意在修改IsBuildMudle的值时，一定要sysn gradle。  
当module单独运行和作为module运行时，其activity在manifest中设置也会不同，这里可以根据IsBuildMudle设置不同的manifest：

sourceSets {

main {

if (IsBuildMudle.toBoolean()) {

manifest.srcFile 'src/main/debug/AndroidManifest.xml'

} else {

manifest.srcFile 'src/main/release/AndroidManifest.xml'

}

}

}

分别在不同的目录下创建manifest文件。一定要注意两个**manifest的同步**问题，否则出现了莫名其名的bug，还找不到原因……

#### 资源冲突问题

当分别开发模块时，容易出资源重复命名的问题，可以在build.gradle中设置

resourcePrefix "module1\_"

通过给模块设置不同的资源前缀，可以避免重复命名。

#### Activity跳转问题

拆分业务代码时，自然会涉及到跨module的Activity跳转，当单独编译时，自然是不能获取到其他模块的引用的。有几种方式可以实现跨模块的唤起Activity：

**隐式启动**  
通过设置intent-filter实现，这需要在manifest中插入大量代码，同时也降低了安全性（其他app就可以通过这种方式随意启动）。  
**通过类名跳转**  
[Android业务组件化开发实践](http://kymjs.com/code/2016/10/18/01" \t "_blank)提出了一种通过类名跳转的方式，使用脚本生成Rlist类，比较方便快捷，但感觉不方便activity间传递数据。  
**Scheme跳转**  
Scheme的方式是建立映射表，集中处理Activity，这种方式可以传递一定的数据。Activity传递大量数据时可以通过EventBus来进行传递（其实即使通过intent显示启动，也不要把大量数据放置在intent中，intent对数据大小有限制）。  
在进行本次实践时找到github上的一个url Router，同时支持http和程序内Activity跳转，而且通过注解的方式进行，使用非常方便，于是引入到了项目中。项目地址[ActivityRouter](https://github.com/mzule/ActivityRouter" \t "_blank)。[ActivityRouter](https://github.com/mzule/ActivityRouter" \t "_blank)的readme中已经有比较详细的wiki，但是还有一些需要注意的：

依赖问题：

[ActivityRouter](https://github.com/mzule/ActivityRouter" \t "_blank)使用了apt方式，因此每个使用的module中均需要设置

apt 'com.github.mzule.activityrouter:compiler:1.1.5'

注意是每个module,在Library module中设置

classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'

即可。

多module问题

[ActivityRouter](https://github.com/mzule/ActivityRouter" \t "_blank)通过注解在编译时生成mapping，如果多个module设置依赖，就会生成多个java文件，导致源文件重复，编译出错，[ActivityRouter](https://github.com/mzule/ActivityRouter" \t "_blank)目前提供了解决方法，但是还没有正式发布版本，可以设置：

compile 'com.github.mzule.activityrouter:activityrouter:1.1.9'

apt 'com.github.mzule.activityrouter:compiler:1.1.6'

使用。使用在application中注解：

@Modules({"app", "moduleA", "moduleB"})

public class ModularizationApplication extends Application {

}

每个module中创建空java类注解：

@Module("moduleA")public class ModuleA {

}

具体可以clone [ModularizationApp](https://github.com/wutongke/ModularizationApp" \t "_blank)查看代码。

## 编译运行

当在gradle.properties中设置IsBuildMudle=true时，可以独立运行每个module，独立运行调试，当设置IsBuildMudle=false，可以编译运行整个project，注意IsBuildMudle变量设置改变时，要对**gradle进行sysn**。

运行过程中有什么问题可以评论或者在github中提issue。

参考：  
[http://kymjs.com/code/2016/10/18/01](http://kymjs.com/code/2016/10/18/01" \t "_blank)  
[https://github.com/mzule/ActivityRouter](https://github.com/mzule/ActivityRouter" \t "_blank)  
[https://github.com/liangzhitao/ComponentizationApp](https://github.com/liangzhitao/ComponentizationApp" \t "_blank)