# Android业务组件化开发实践

组件化并不是新话题，其实很早很早以前我们开始为项目解耦的时候就讨论过的。但那时候我们说的是功能组件化。比如很多公司都常见的，网络请求模块、登录注册模块单独拿出来，交给一个团队开发，而在用的时候只需要接入对应模块的功能就可以了。

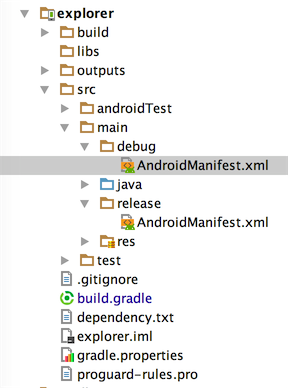
今天我们来讨论一下业务组件化，拿出手机，打开淘宝或者大众点评来看看，里面的美食电影酒店外卖就是一个一个的业务。如果我们在一个项目里面去写的时候，总会出现或多或少的代码耦合，最典型的有时为了赶上线时间而先复制粘贴一段类似的代码过来，结果这段代码引用的资源可能是另一个模块独立的资源或代码。但是如果将一个项目作为独立的工程来运行，就完全可以避免这种情况了。但是这并不是业务组件化最大的优势，我认为最大的优势是它大大缩减了工程结构直接降低了编译时间。

## 代码实现

注意，组件化不是插件化，插件化是在［运行时］，而组件化是在［编译时］。换句话说，插件化是基于多 APK 的，而组件化本质上还是只有一个 APK。

代码实现上核心思路要紧记一句话：开发时是 application，发版时是 library。  
来看一段 gradle 代码：

if (isDebug.toBoolean()) {  
 apply plugin: 'com.android.application'  
} else {  
 apply plugin: 'com.android.library'  
}

非常好理解，我们在开发的时候，module 如果是一个库，会使用com.android.library插件，如果是一个应用，则使用com.android.application插件，我们通过一个开关来控制这个状态的切换。  
然后因为我们需要在 library 和 application 之间切换，manifest文件也需要提供两套。  


#### **举个栗子？**

你可以根据这个项目一起看：[https://github.com/kymjs/Modularity](https://github.com/kymjs/Modularity" \t "http://kymjs.com/code/2016/10/18/_blank)

假设有一个项目，这个项目包含一个叫 explorer 的文件浏览器的模块和一个叫 memory-box 的笔记的模块。因为这两个功能相对独立，我们将这两个功能拆分成两个 module，再加上原本项目的 app module，总共三个。  
在 explorer 的根目录建立一个作为开关的 properties 文件（写一个全局变量也可以，怎么简单怎么来），方便用来改变当前是开发状态还是发版状态(debug & release)。 从gradle中读取这个文件中的值，来切换不同状态所需要调用的配置。顺便一提，当你修改了 properties 文件中的值时，必须要重新 sync 一下。 详细配置过程可以看看这篇文章：[http://www.zjutkz.net/](http://www.zjutkz.net/2016/10/07/%E5%85%B3%E4%BA%8EAndroid%E4%B8%9A%E5%8A%A1%E7%BB%84%E4%BB%B6%E5%8C%96%E7%9A%84%E4%B8%80%E4%BA%9B%E6%80%9D%E8%80%83/" \t "http://kymjs.com/code/2016/10/18/_blank)

## 遇到的问题

阿布他们的项目大量的用了 databinding 和 dagger，然而我们项目并没有用这些，用了这两个库的可以看看他是怎么爬坑的：[魔都三帅](http://mp.weixin.qq.com/s?__biz=MzIwOTQ1MjAwMg==&mid=2247483776&idx=1&sn=df3160bf0e88c3cbb2a651ef38a1de8d&chksm=9772ef0da005661b7e01c63decdb413662990d3b2e385cef9f1ed86eac4843d000681522d3fd&mpshare=1&scene=1&srcid=1017hRjlGpVuFGVxkdr5tYXp" \l "rd" \t "http://kymjs.com/code/2016/10/18/_blank)

当你采用了组件化开发的时候，一定会遇到这几个问题，这几个问题除了第三个都只能规避，没有好的处理办法：

1、module 中 Application 调用的问题  
2、跨 module 的 Activity 或 Fragment 跳转问题  
3、AAR 或 library project 重复依赖   
4、资源名冲突

#### **解决 Application 冲突**

由于 module 在开发过程中是以 application 的形式存在的，如果这个 module 调用了类似((XXXApplication)getApplication()).xxx()这种代码的话，最终 release 项目时一定会发生类转换异常。因为在 debug 状态下的 module 是一个 application，而在 release 状态下它只是一个 lib。所以也就是在 debug 和 release 时获取到的 Application 不是同一个类的对象。  
这个问题还好，我们只要在 application 里面尽量不要写方法实现，不要做强转操作就好。  
如果确实要区分，业务模块在 debug 状态和 release 状态有不同的行为，可以通过扩展 BuildConfig 这个类，在代码中通过 boolean 值来执行不同的逻辑。只需要在 gradle 中加入(具体代码用法可查看[【line:48】](https://github.com/kymjs/Modularity/blob/master/Modular/explorer/src/main/java/com/kymjs/app/explorer/MainActivity.java" \t "http://kymjs.com/code/2016/10/18/_blank))：

if (isDebug.toBoolean()) {

buildConfigField 'boolean', 'ISAPP', 'true'

} else {

buildConfigField 'boolean', 'ISAPP', 'false'

}

有些人喜欢将 application 单例，写一个静态的对象，然后在代码里面需要context的时候用这个全局单例。这样的情况我送大家一个工具类(其实是从冯老师代码里偷来的)：[Common](https://github.com/kymjs/Common/blob/master/Common/common/src/main/java/com/kymjs/common/App.java" \t "http://kymjs.com/code/2016/10/18/_blank)

public class *App* {

public static final Application INSTANCE;

static {

Application app = null;

try {

app = (Application) Class.forName("android.app.AppGlobals").getMethod("getInitialApplication").invoke(null);

if (app == null)

throw new IllegalStateException("Static initialization of Applications must be on main thread.");

} catch (final Exception e) {

LogUtils.e("Failed to get current application from AppGlobals." + e.getMessage());

try {

app = (Application) Class.forName("android.app.ActivityThread").getMethod("currentApplication").invoke(null);

} catch (final Exception ex) {

LogUtils.e("Failed to get current application from ActivityThread." + e.getMessage());

}

} finally {

INSTANCE = app;

}

}

}

#### **跨 module 跳转**

如果单独是 Activity 跳转，常见的做法是：隐式启动 Activity、或者定义 scheme 跳转。  
但是如果界面是一个 Fragment 就比较麻烦了，我推荐的是直接通过类名跳转。

首先创建一个所有界面类名的列表

public class *RList* {

public static final String ACTIVITY\_MEMORYBOX\_MAIN = "com.kymjs.app.memory.module.main.MainActivity";

public static final String FRAGMENT\_MEMORYBOX\_MAIN = "com.kymjs.app.memory.module.list.MainFragment";

}

在获取 Fragment 的时候就可以根据列表中的类名来读取指定的 Fragment 了。

public class FragmentRouter {

public static Fragment getFragment(String name) {

Fragment fragment;

try {

Class fragmentClass = Class.forName(name);

fragment = (Fragment) fragmentClass.newInstance();

} catch (Exception e) {

throw new RuntimeException(e);

}

return fragment;

}

}

同理，Activity 其实也可以用这种方法来跳转：

public static void startActivityForName(Context context, String name) {

try {

Class clazz = Class.forName(name);

startActivity(context, clazz);

} catch (ClassNotFoundException e) {

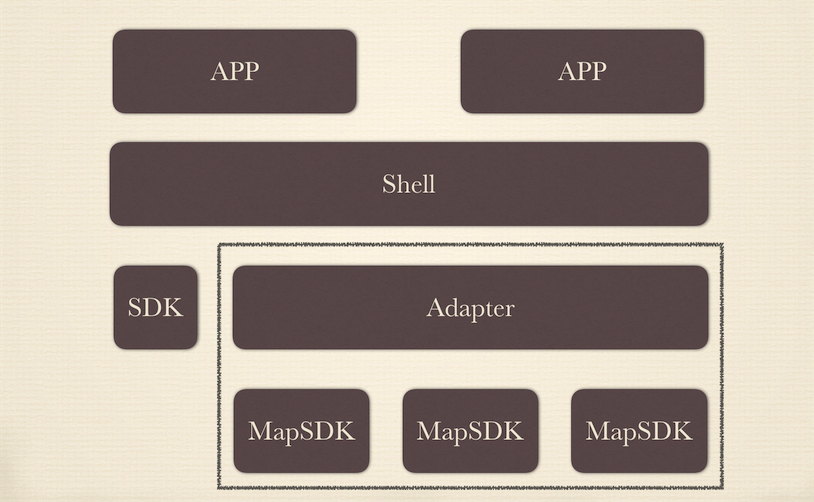
throw new RuntimeException(e);

}

}

最后，对于这个RList类，我们还可以通过 Gradle 脚本来生成，就像 R 文件一样，这样子开发就要方便很多了。

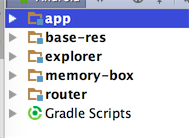
#### **重复依赖**

重复依赖问题其实在开发中经常会遇到，比如你 compile 了一个A，然后在这个库里面又 compile 了一个B，然后你的工程中又 compile 了一个同样的B，就依赖了两次。  
默认情况下，如果是 aar 依赖，gradle 会自动帮我们找出新版本的库而抛弃旧版本的重复依赖。但是如果你使用的是 project 依赖，gradle 并不会去去重，最后打包就会出现代码中有重复的类了。  
一种是 将 compile 改为 provided，只在最终的项目中 compile 对应的代码，但是这种办法只能用于没有资源的纯代码工程或者jar包；  
还可以使用这种方案：  
  
可以将所有的依赖写在 shell 层的 module，这个 shell 并不做事情，他只用来将所有的依赖统一成一个入口交给上层的 app 去引入，而项目所有的依赖都可以写在 shell module 里面。

#### **资源名冲突**

因为分了多个 module，在合并工程的时候总会出现资源引用冲突，比如两个 module 定义了同一个资源名。  
这个问题也不是新问题了，做 SDK 基本都会遇到，可以通过设置 resourcePrefix 来避免。设置了这个值后，你所有的资源名必须以指定的字符串做前缀，否则会报错。  
但是 resourcePrefix 这个值只能限定 xml 里面的资源，并不能限定图片资源，所有图片资源仍然需要你手动去修改资源名。

## 项目结构



****app**** 是最终工程的目录  
****explorer**** 和 ****memory-box**** 是两个功能模块，他们在开发阶段是以独立的 application，在 release 时才会作为 library 引入工程。  
****router**** 有两个功能，一个是作为路由，用于提供界面跳转功能。另一个功能是前面讲的 shell ，作为依赖集合，让各业务 module 接入。 ****base-res**** 是一些通用的代码，即每个业务模块都会接入的部分，它会在 router 中被引入。

最终代码可以查看：[https://github.com/kymjs/Modularity](https://github.com/kymjs/Modularity" \t "http://kymjs.com/code/2016/10/18/_blank)