# 组件化在项目中的使用姿势

https://mp.weixin.qq.com/s?__biz=MzA5MzI3NjE2MA==&mid=2650243065&idx=1&sn=013d54ab819fc239bb2bc0b9183e80ae&chksm=88638e96bf140780dad45ee03eb65406a03d11c545969899aef5cd74189a6e402f976841bcac&scene=38#wechat_redirect

组件化在项目中的使用姿势

原创： smileCH 郭霖 5月22日

## 今日科技快讯

5月18日下午，百度宣布，集团总裁兼COO陆奇因个人和家庭原因，从7月起不再担任上述职务。消息一出，百度股价应声下跌，一夜之间市值蒸发接近100亿美元。北京时间5月18日，百度盘前大跌超过6.5%，盘中更是跌幅超过10%，随后有小幅回调。19日凌晨收盘时，百度收盘价为253.01美元，下跌9.54%，与18日凌晨相比，市值蒸发93亿美元。

## 作者简介

本篇来自 **smileCH** 的投稿，分享了组件化在项目中的使用姿势，一起来看看！希望大家喜欢。

**smileCH** 的博客地址：

https://www.jianshu.com/u/22e84d1967f4

## 开始

老规矩还是先来张图，找不到录制gif的工具了，就先来张静态图吧

为什么要引入组件化呢？相信大家在开发中一定经历了这几个阶段，起初项目业务还比较单一时，我们采用MVC进行开发，当然了，写的很快也很开心，不过随着产品不停的加需求时，慢慢的就感觉MVC过于臃肿了，进而改用MVP模式或者MVVM进行开发，慢慢的..慢慢的...我们发现利用MVP框架也已经招架不住产品的洪荒之力了，单一工程下的代码耦合性越来越高，编译速度越来越慢。就拿我们公司的项目来说吧，我们是做旅游方面的，产品涉及到机票酒店和火车票等业务，那么组件化就很适合了，很显然可以将单一工程拆分成机票组件、火车票组件、酒店组件等，同事之间只需要各自负责各自的组件即可，单元测试也会变得比较方便，当然了，我们可以将这些公共组件、业务组件上传到自己公司搭建的仓库上，比如我负责机票模块，那么我只需要把机票的down下来，那么编译项目的速度可想而知。在将项目改造成组件化的过程中出现了很多问题，也参考了网上很多前辈的博客，非常感谢前辈们。

说了这么多废话，还请各位看官放下手中的刀，本文将详细讲解组件化的使用，demo中使用到的地址是鸿洋老铁的wanandroid接口地址，在此表示感谢。

先来看下选择集成化模式和组件化模式下运行项目的截图

*集成模式.png*

*组件化模式.png*

然后再来看下组件化之后的项目结构：

*组件化之后的项目结构.png*

接下来针对上面的结构我们来简要解释下，以及一些注意事项等。

### app壳工程

首先app这个组件作为项目的壳，里面没有任何代码，主要做两件事，一是我们可以在这个壳工程中的build.gradle中去配置一些签名信息以及非组件化时依赖其他业务等组件的判断

*依赖配置.png*

二是我们还可以在这个壳工程中定义我们的自定义application去进行一些第三方的初始化等，提及到application我们下文会详细说明下。

## common_module公共组件

common_module公共组件，主要放一些通用的工具类等，尽量不要在此组件中涉及到业务相关的内容，比如我们的一些base基类、封装的http请求框架、一些util工具类以及styles.xml中统一设置主题配置等都可以放到这个公共组件中，当然了项目中需要引用到的一些第三方依赖肯定也是要放到这个公共组件的build.gradle中的，这样做的好处就是不需要每一个业务组件都去引用这些依赖，只需要依赖我们的common_module就行了，同时，对于一些AndroidManifest.xml文件中设置的权限等也都放到common组件中统一处理，这样就不用在每一个组件的配置文件中都去设置网络、读写文件等等权限。

```
dependencies {
    api fileTree(dir: 'libs', include: ['*.jar'])
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

    api rootProject.ext.dependencies.appcompat_v7
    api rootProject.ext.dependencies.design
    api rootProject.ext.dependencies.retrofit
    api rootProject.ext.dependencies.retrofit_converter_gson
    api rootProject.ext.dependencies.retrofit_adapter_rxjava2
    api rootProject.ext.dependencies.gson
    api rootProject.ext.dependencies.rxandroid
    api rootProject.ext.dependencies.rxjava
    api rootProject.ext.dependencies.glide
    api rootProject.ext.dependencies.constraint_layout
    api rootProject.ext.dependencies.eventbus
    api rootProject.ext.dependencies.okhttp3_logging
    api rootProject.ext.dependencies.SmartRefreshLayout
    api rootProject.ext.dependencies.banner
    api rootProject.ext.dependencies.arouter_api
}
```

为了方便项目管理，我们习惯性的喜欢新建一个config.gradle文件并将这些依赖配置到此文件中，便于管理。这里需要注意的是采用的时api的方式，因为implementation指令是不对外公开的，也就是说其他业务组件依赖common_module后仍然无法引用到retrofit、gson等。前面说过将封装的网络请求框架放到common_module组件中，这里我用Retrofit2+RxJava2封装的一个网络请求框架（感兴趣的可以自己去查看代码，欢迎拍砖指正），那么我们是不是需要面对这样的代码

```
retrofit = new Retrofit.Builder()
                .baseUrl(Constant.BASE_URL)
                .client(client)
                .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
                .addConverterFactory(GsonConverterFactory.create())
                .build();
        server = retrofit.create(RetrofitServer.class);
```

那么，问题就来了，假如我们请求wanandroid中的banner图的接口

```
@GET("banner/json")
Observable<BaseResponseBean<List<BannerBean>>> getBannerImgs();
```

很显然，我们就需要在common_module中新建这个BannerBean实体类，这就违背了公共组件中不涉及业务的原则，那么该如何解决呢？很简单我们可以通过泛型去解决，动态的将api传过来，详情可以查看RxRetrofitManager类

```
public <T> T getApiService(Class<T> apiServer){
        return retrofit.create(apiServer);
}
```

紧接着我们在使用的时候动态的把这个api类传给我们的请求框架，下面贴一下article_module组件中model包下的ArticleListModel类中的调用请求的部分代码片段

```
RxRetrofitManager.getInstance()
                .setTag("articleBanner")
                .getApiService(ArticleApi.class)
                .getBannerImgs()
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .........省略n行代码
```

这样问题就很容易的被解决了，感兴趣的可以自行去查看代码。

接下来再来介绍下common_module组件中一个比较重要的BaseApplication类，对于application大家都是比较熟悉的这里就不再啰嗦了，我们在代码中有时候可能用到application的context，那么我们的article_module、detail_module等组件在组件化模式下是可以单独运行的，不再是一个library而是一个正常可以跑起来的工程，那么我们一般也会自定义我们自己的application，

代码中可能也会使用到application的context，那么问题就来了，在组件化模式下跑项目都没问题，最终在集成模式下这些组件是都要被合到app壳工程里的，最终项目也只会有一个application，那么我们组件中定义的application肯定是没法使用的，所以我们就需要创建一个BaseApplication，每个业务组件以及app壳工程中的application都去继承这个BaseApplication，那么不管是集成模式下还是组件化模式下，最终用的都是这个BaseApplication的全局context，这样问题相应也就解决了。common_module公共组件主要需要注意的点都讲完了，接下来就要说说article_module、detail_module以及main_module等具体的业务组件。

## 具体的业务组件(article_module、detail_module)

前面我们说了，组件化模式下我们的业务组件可以单独看作是一个正常的应用，可以正常的编译运行的，我们都知道一个正常的应用工程和library的一个很明显的区别就是build.gradle文件，library的build.gradle是这样的

```
apply plugin: 'com.android.library'
```

而我们可以正常运行的工程的build.gradle是这样的

```
apply plugin: 'com.android.application'
```

当然了，能正常运行并不是说build.gradle这样设置就行了，这里只是针对build.gradle文件进行说明，所以这些业务组件就要满足以下两点，一是在组件化模式下他们就是可以正常运行的工程，二是在集成模式下他们就会被作为library合并到app壳工程中，所以我们需要设置一个开关用来达到不同模式要求，我们需要在总的工程根目录的gradle.properties中设置这个开关

```
# Project-wide Gradle settings.

# IDE (e.g. Android Studio) users:
# Gradle settings configured through the IDE *will override*
# any settings specified in this file.

# For more details on how to configure your build environment visit
# http://www.gradle.org/docs/current/userguide/build_environment.html

# Specifies the JVM arguments used for the daemon process.
# The setting is particularly useful for tweaking memory settings.
org.gradle.jvmargs=-Xmx1536m

# When configured, Gradle will run in incubating parallel mode.
# This option should only be used with decoupled projects. More details, visit
# http://www.gradle.org/docs/current/userguide/multi_project_builds.html#sec:decoupled_projects
# org.gradle.parallel=true

# false表示是集成化开发模式，true表示是组件化开发模式
isModule = false
```

接下来，再来看下每一个业务组件的build.gradle中是如何使用这个开关处理不同的模式的，需要注意下，gradle.properties中的值都是String类型的，这里需要转换下

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

这样我们就完成了第一步，纳尼？？这才是第一步，老铁，你没看错.....

还是先把业务组件的大体结构展示下吧，不然说起来...会说不清的

*业务组件结构目录.png*

我们看到java包下我们新建了一个debug包，里面有我们自定义的application以及一个activity，前者的用处我们在上文中已经啰嗦过了，是为了保证组件化模式下使用application的context是BaseApplication的context，后者的作用也很明显，比如我们这个article_module组件主要是展示banner以及文章列表数据，但是做这些操作需要登录之后拿到用户的id等信息作为接口的参数才能正常展示数据，然而登录模块并不在此组件中，那么怎么办？我们就可以在这个activity中去模拟登录，获取到id等信息后再跳转到这个文章列表页面，当然了，在最终进行代码集成合并打包的时候，这些debug包是需要删除掉的。在build.gradle中可以这样处理

```
sourceSets {
        main {
            if (isModule.toBoolean()) {
                manifest.srcFile 'src/main/module/AndroidManifest.xml'
```

```
            } else {
                manifest.srcFile 'src/main/AndroidManifest.xml'
                //集成开发模式下则需要排除debug文件夹中的所有Java文件
                java {
                    exclude 'debug/**'
                }
            }
        }
    }
```

由上图我们看到main包下新建了个module文件，里面放的是AndroidManifest.xml文件，因为在组件化模式下，项目组件是可以直接运行的，所以就需要像正常清单文件那样，有自己的启动activity等

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.smile.ch.article">

    <application
        android:name="debug.ArticleApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/article_name"
        android:supportsRtl="true"
        android:theme="@style/AppBaseTheme">
        <activity android:name=".ArticleListActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

然而在集成模式下，这些组件中的清单配置文件中的代码是要被合并到app壳工程的清单配置文件中的，所以在集成模式下我们引用的是此路径下的清单配置文件（manifest.srcFile 'src/main/AndroidManifest.xml'），主题我们都使用common组件中统一配置的主题，这样整个app主题就保持一致了

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.smile.ch.article">

    <application
        android:theme="@style/AppBaseTheme"
        >
        <activity android:name=".ArticleListActivity"
            android:screenOrientation="portrait"/>
    </application>

</manifest>
```

最后贴下完整的业务组件的build.gradle文件

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


android {
    compileSdkVersion rootProject.ext.android.compileSdkVersion
    defaultConfig {
        minSdkVersion rootProject.ext.android.minSdkVersion
        targetSdkVersion rootProject.ext.android.targetSdkVersion
        versionCode rootProject.ext.android.versionCode
        versionName rootProject.ext.android.versionName

        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
```

```
        //ARouter
        javaCompileOptions {
            annotationProcessorOptions {
                arguments = [moduleName: project.getName()]
            }
        }

    }

    sourceSets {
        main {
            if (isModule.toBoolean()) {
                manifest.srcFile 'src/main/module/AndroidManifest.xml'
            } else {
                manifest.srcFile 'src/main/AndroidManifest.xml'
                //集成开发模式下则需要排除debug文件夹中的所有Java文件
                java {
                    exclude 'debug/**'
                }
            }
        }
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }

}

dependencies {
    implementation fileTree(include: ['*.jar'], dir: 'libs')
    annotationProcessor rootProject.ext.dependencies.arouter_compiler
    implementation project(':common_module')
}
```

到这里业务组件需要注意的点基本上都说完了，detail_module组件基本一样，这里就不再赘述了，main_module需要单独拿出来说一下

## main_module

其实，main_module组件跟其他业务组件一样，一般建议把启动页、登录页、注册页等页面放到此组件中，因为我们的app壳工程中是不放任何业务代码的，最终在集成模式下，所有的业务组件是都要合并到这个app壳工程中的。

## 资源重名导致的一些问题

当我们潇潇洒洒的写完代码，在集成开发模式下运行程序时可能会出现类似下图所示的错误

*布局文件名冲突.png*

出现这个问题的原因是我在app壳工程的layout文件夹下有一个activity_main.xml文件，同时在main_module组件的布局文件中也有这么一个activity_main.xml文件，这样就出现了上图所示的错误，那么如何避免呢？一般情况下，在不同的组件下的布局文件、资源图片等的命名我们可以根据当前所在组件的关键字作为前缀，就拿article_module和detail_module两个组件举个例子吧，我们可以这样来命名：activity_article_main.xml、activity_detail_main.xml等，具体如何做可以根据自己的习惯，最终保证这些资源不重名即可。

## activity之间的跳转以及传值

比如article_module组件中有一个文章列表数据的activity，点击item需要跳转到detail_module组件中的展示详情的一个activity页面中，很显然我们不能再用之前intent的方式跳转了，因为引用不到其他组件中的activity类了，那么问题来了，我们该如何处理呢？我们可以通过路由机制来实现，这里我采用的是阿里开源的ARouter路由，至于如何使用不是本文的讲解范围，有兴趣的可以自行查看文档，当然了并不是说我们在组件化模式下单独运行article_module组件之后点击item列表就可以通过此机制跳转到detail_module组件里的详情activity，而是通过这种方式在最终的集成模式下可以跨组件进行跳转，说的有点绕哈，见谅...在demo中我是把ARouter的初始化放在了app壳工程的application中去初始化的，组件与组件之间activity跳转

是通过此路由机制实现的，所以，单独运行其中的某一个组件的化点击跳转会报错，是因为我们单独运行组件时然而没在组件中进行初始化，所以我们可以在每一个组件的debug包下的application中进行初始化，反正最后合并的时候debug下的java文件是会被删除掉的。

组件间跳转的问题解决了，如何解决不同组件间的通信呢？

### 不同组件间的通信

其实方式有很多种，我们可以采用EventBus等进行通信。

关于组件化的使用介绍就到这里了，其实这个demo还是比较简单的，我把文章banner以及列表作为一个article_module组件，点击banner或者列表item到详情页，我把详情页单独抽出来作为detail_module组件，为什么要这样抽呢？为了达到演示组件化的效果，毕竟这只是一个demo帮助大家理解的。对于公司的老项目改成这种组件化模式问题还是挺多的，就比如我们公司的项目，机票、酒店等都用到了一些通用的activity页面，总不能把这些页面放到common组件中吧，因为前面说过common组件中不应该涉及到业务相关的内容，所以我目前的做法是再新建一个功能组件用于放置这些通用的activity页面等，以及把之前的代码抽到不同的组件中时还需要修改switch语句等等，所以整个改造下来还是很痛苦的。在此期间查阅了很多前辈写的文章，再次向前辈们表示感谢。

## 总结

能坚持听我啰嗦到这的都是真老铁啊，文章有点长，文字描述比较多，目的主要是为了把这个点尽量的说明白，因为这些也是我自己在实践中踩的坑，如果对于文章中有什么不理解的，可以下载代码看一下然后再跑一遍就明白了，或者提出来我们可以一起讨论，有不合理的地方还请大家指出。最后，没有更好的架构，只有更适合的架构。以此来共勉。

github地址如下所示：

https://github.com/smileCH/ComponentProject