# 组件化开发注意事项

**一、组件化要考虑以下8个方面问题：**

1. **代码解耦**。如何将一个庞大的工程分成有机的整体？
2. **组件单独运行**。因为每个组件都是高度内聚的，是一个完整的整体，如何让其单独运行和调试？
3. **组件间通信**。由于每个组件具体实现细节都互相不了解，但每个组件都需要给其他调用方提供服务，那么主项目与组件、组件与组件之间如何通信就变成关键？
4. **UI 跳转**。UI 跳转指的是特殊的数据传递，跟组件间通信区别有什么不同？
5. **组件生命周期**。这里的生命周期指的是组件在应用中存在的时间，组件是否可以做到按需、动态使用、因此就会涉及到组件加载、卸载等管理问题。
6. **集成调试**。在开发阶段如何做到按需编译组件？一次调试中可能有一两个组件参与集成，这样编译时间就会大大降低，提高开发效率。
7. **组件资源合并冲突**。集成模式下各组件的资源命名冲突如何解决？
8. **代码隔离**。组件之间的交互如果还是直接引用的话，那么组件之间根本没有做到解耦，如何从根本上避免组件之间的直接引用，也就是如何从根本上杜绝耦合的产生？

**二、对应以上8个问题具体解决方案**

1. **代码解耦问题**

模块拆分，模块分为三种类型，一种是功能组件模块，封装一些公共的方法服务等，作为依赖库对外提供，一种是资源组件模块，为各业务组件提供共用资源，最后一种是业务组件模块，专门处理业务逻辑等功能，这些业务组件模块最终负责组装APP。

1. **组件单独运行问题**

通过 **Gradle** 脚本配置方式，进行不同环境切换。比如只需要把 Apply plugin: 'com.android.library' 切换成Apply plugin: 'com.android.application' 就可以，同时还需要在 AndroidManifest 清单文件上进行设置，因为一个单独调试需要有一个入口的 Activity。比如设置一个变量 isModule，标记当前是否需要单独调试，根据isModule 的取值，使用不同的 gradle 插件和 AndroidManifest 清单文件，甚至可以添加 Application 等 Java 文件，以便可以做一下初始化的操作。

1. **组件间通信问题**

通过接口+实现的结构进行组件间的通信。每个组件声明去common组件提供的服务 provider API，这些 provider 都是一些接口，组件负责将这些 provider 实现并注册到一个统一的路由 Router 中去，如果要使用某个组件的功能，只需要向Router 请求这个 Provider 的实现，具体的实现细节我们全然不关心，只要能返回我们需要的结果就可以了。在组件化架构设计图中 Common 组件就包含了路由服务组件，里面包括了每个组件的路由入口和跳转。

1. **UI 跳转问题**

可以说 UI 跳转也是组件间通信的一种，但是属于比较特殊的数据传递。不过一般 UI 跳转基本都会单独处理，一般通过短链的方式来跳转到具体的 Activity。每个组件可以注册自己所能处理的短链的 Scheme 和 Host，并定义传输数据的格式，然后注册到统一的 UIRouter 中，UIRouter 通过 Scheme 和 Host 的匹配关系负责分发路由。但目前比较主流的做法是通过在每个 Activity 上添加注解，然后通过 APT 形成具体的逻辑代码。目前方式是引用阿里的 [ARouter](https://github.com/alibaba/ARouter) 框架，通过注解方式进行页面跳转。

1. **组件生命周期问题**

在架构图中的核心管理base组件会定义一个组件生命周期接口，通过在每个组件设置一个配置文件application,把这个配置文件全路径放到壳app的application中统一实例化并绑定application的生命周期。

1. **集成调试问题**

每个组件单独调试通过并不意味着集成在一起没有问题，因此在开发后期我们需要把几个组件机集成到一个 APP 里面去验证。由于经过前面几个步骤保证了组件之间的隔离，所以可以任意选择几个组件参与集成，这种按需索取的加载机制可以保证在集成调试中有很大的灵活性，并且可以加大的加快编译速度。需要注意的一点是，每个组件开发完成之后，需要把 isModule 设置为 true并同步，这样主项目就可以通过参数配置统一进行编译。

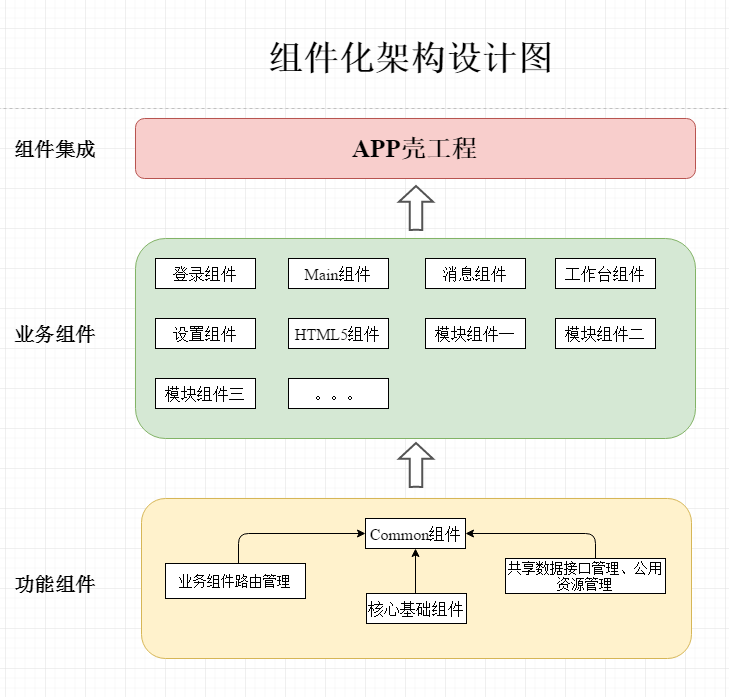
1. **组件合并资源冲突问题。**

组件资源资源命名需要遵循一定组件命名规范，避免出现冲突。比如登陆组件，那么名称：module-login(组件名称)，并且在gradle约束资源开头命名resourcePrefix “login\_(组件名称)”，需要注意的是：**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手动去修改资源名**

1. **代码隔离问题**

主项目只依赖 Common 的依赖库，业务组件通过路由服务依赖库按需进行查找，用反射方式进行组件加载，然后在主工程中调用组件服务，组件与组件之间调用则是通过接口+实现进行通信。

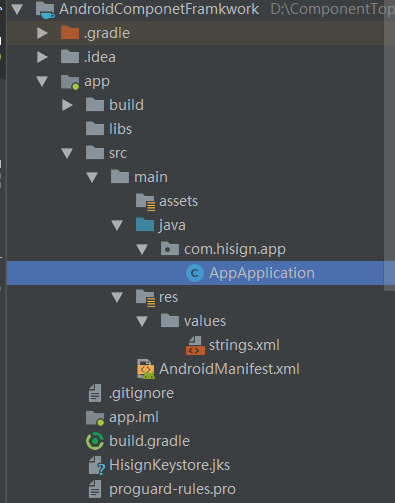
### 三、架构设计图

****

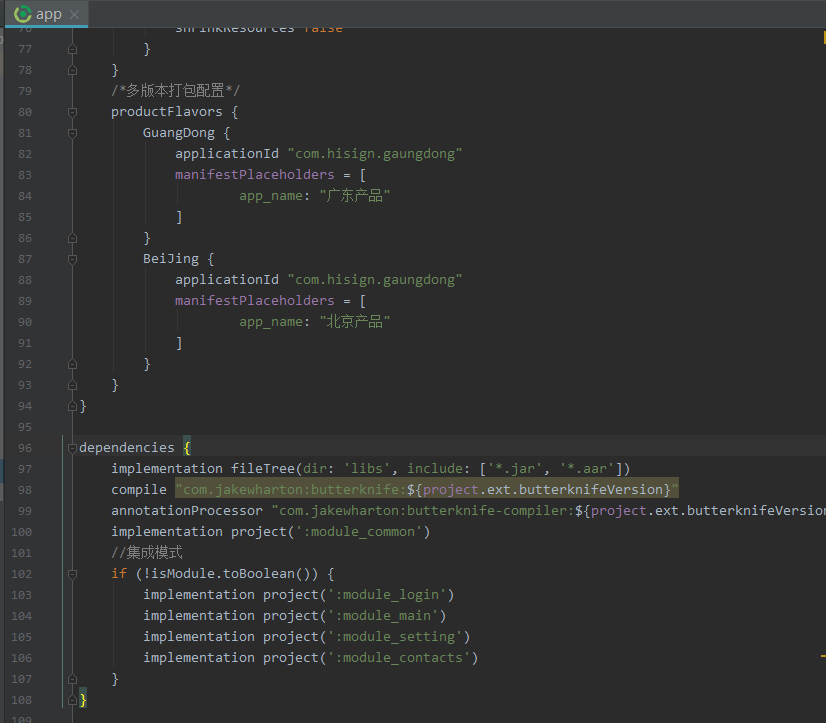
### 四、具体实践步骤

**1.App空壳工程**

**1）创建一个APP空壳工程，如图：**

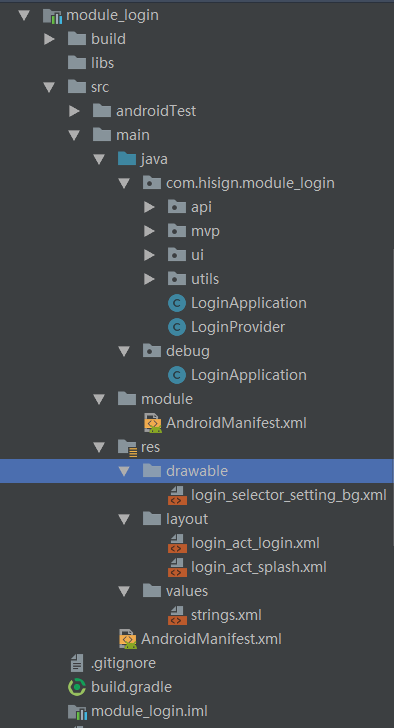


然后在 APP 工程添加依赖具体业务组件 Module。比如：



**2）具体业务组件Module**

需要遵循一定组件命名规范，为何需要规范呢，因为需要通过组件命名规范来约束和保证组件的统一性和一致性，避免出现冲突。比如登陆组件，那么名称：module-login(组件名称)，并且在gradle约束资源开头命名resourcePrefix “login\_(组件名称)”。 **注意**：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手动去修改资源名。



**2.业务组件配置文件**

**1）业务组件gradle配置如下**

if (isModule.toBoolean()) {  
 apply plugin: 'com.android.application'  
} else {  
 apply plugin: 'com.android.library'  
}  
apply plugin: 'com.jakewharton.butterknife'  
  
android {  
 compileSdkVersion project.ext.compileSdkVersion  
 defaultConfig {  
 minSdkVersion project.ext.minSdkVersion  
 targetSdkVersion project.ext.targetSdkVersion  
 versionCode 1  
 versionName "1.0"  
 resourcePrefix "login\_"  
 javaCompileOptions {  
 annotationProcessorOptions {  
 arguments = [moduleName: project.getName()]  
 includeCompileClasspath = true  
 }  
 }  
 }  
   
 compileOptions {  
 sourceCompatibility JavaVersion.*VERSION\_1\_8* targetCompatibility JavaVersion.*VERSION\_1\_8* }  
   
 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(dir: 'libs', include: ['\*.jar'])  
 annotationProcessor "com.alibaba:arouter-compiler:${project.ext.arouterCompilerVersion}"  
 compile "com.jakewharton:butterknife:${project.ext.butterknifeVersion}"  
 annotationProcessor "com.jakewharton:butterknife-compiler:${project.ext.butterknifeVersion}"  
 implementation project(':module\_common')  
}

**这里需要有几点说明一下：**

1．通过 isModule.toBoolean() 方法来进行组件间集成模式和组件模式的切换，包括模块是属于Application 还是 Library，由于集成了 ARouter，所以需要对 ARouter 配置文件进行处理。

2．如果组件模式下， 则需要重新设置 AndroidManifest.xml 文件，里面配置新的Application路径。比如Login组件单独运行 AndroidManifest 清单文件

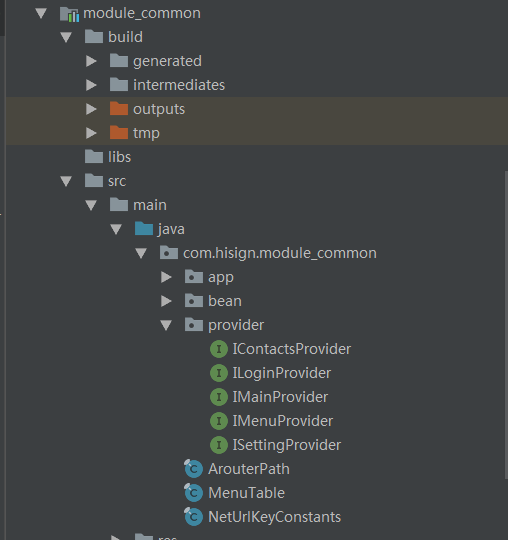
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 package="com.hisign.module\_login">  
  
 <application  
 android:name="debug.LoginApplication"  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/login\_str\_app\_name"  
 android:supportsRtl="true"  
 tools:replace="android:label,android:theme"  
 android:theme="@style/BaseAppTheme">  
 <activity  
 android:name="com.hisign.module\_login.ui.activity.SplashActivity"  
 android:screenOrientation="portrait">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN"/>  
  
 <category android:name="android.intent.category.LAUNCHER"/>  
 </intent-filter>  
 </activity>  
 <!--登录-->  
 <activity android:name="com.hisign.module\_login.ui.activity.LoginActivity"/>  
 </application>  
  
</manifest>

**3．实现组件全局应用配置类**，这个类的目的是在组件加载时初始化一些组件自身的资源，如下：

public class LoginApplication implements IModuleLifeCycle {  
  
 @Override  
 public void onCreate() {  
 //初始化文件夹  
 InitUtil.*initFiles*();  
 //初始化网络访问链接  
 SPUtils.*setSharedStringData*(NetUrlKeyConstants.*MAIN\_URL\_KEY*, "http://211.157.146.9:58080/ykc/api/");  
 }  
  
 @Override  
 public void onTerminate() {  
  
 }  
  
 @Override  
 public void onTrimMemory(int level) {  
  
 }  
  
 @Override  
 public void onLowMemory() {  
  
 }  
}

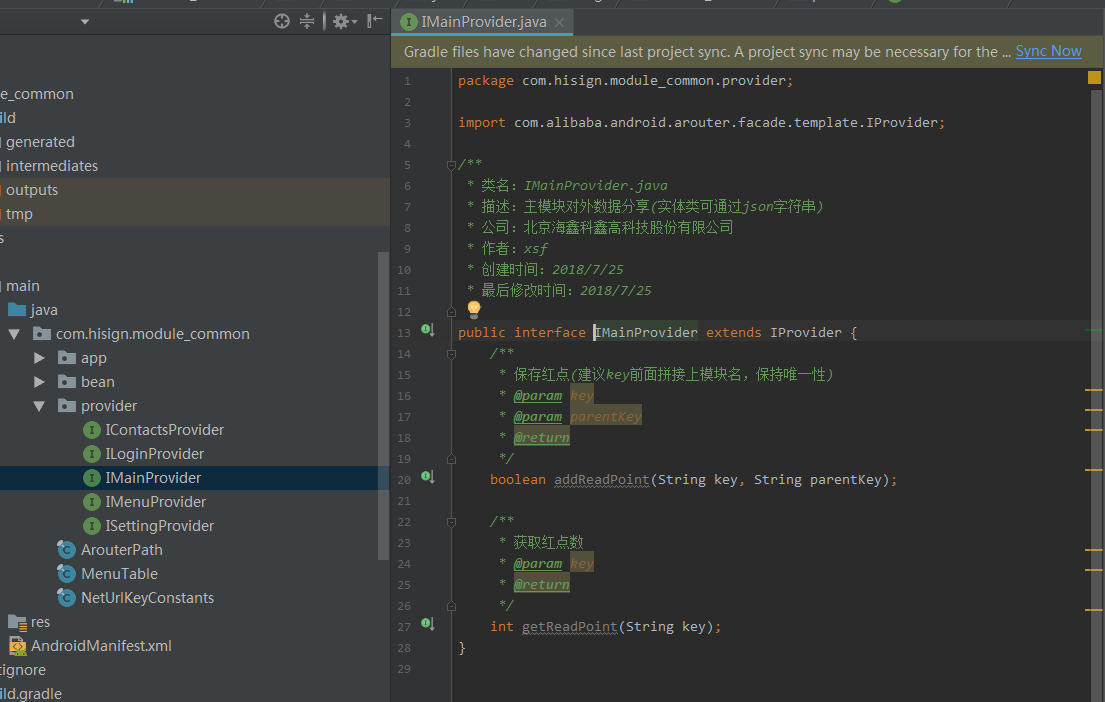
**三、路由服务**

**1．定义公共组件路由Path和入口**，通过路由服务组件查找，如图：

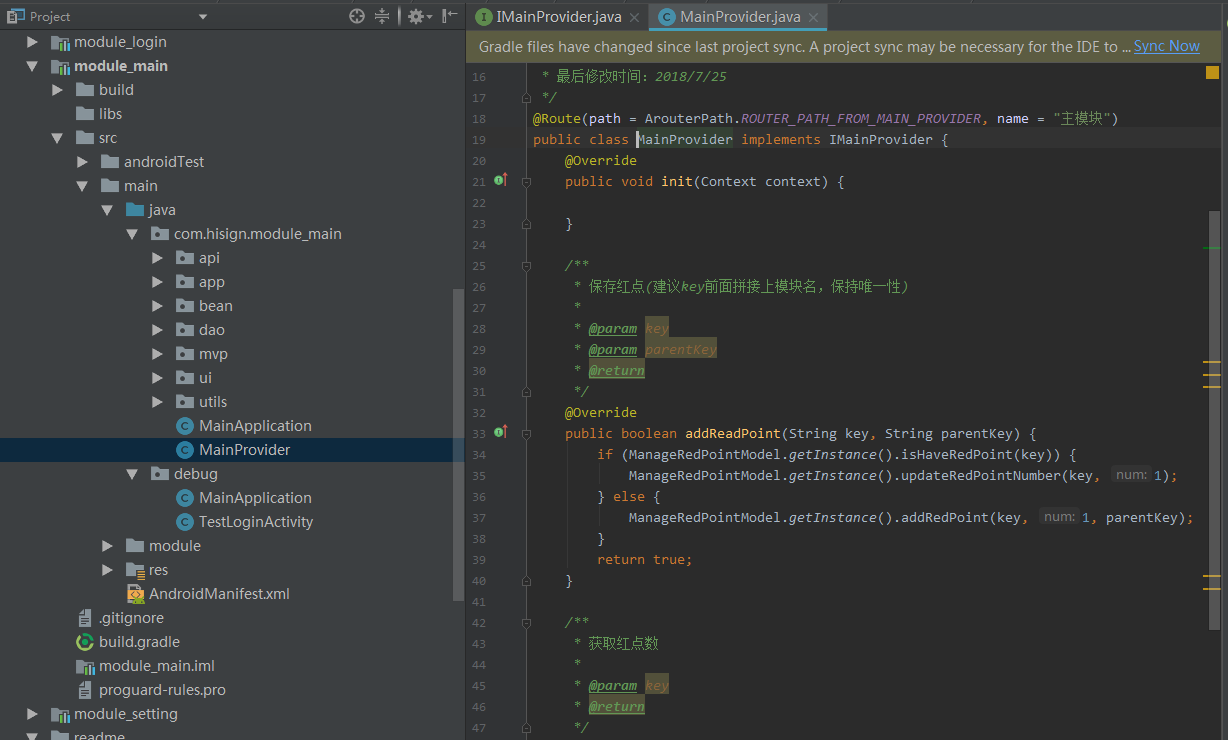


**2．组件路由实现**

每个组件对外提供什么能力，首先需要在路由服务组件创建一个接口文件，如下是登陆组件接口声明和实现。



具体实现：



**路由使用api**

// 构建标准的路由请求  
ARouter.getInstance().build("/home/main").navigation();  
  
// 构建标准的路由请求，并指定分组  
ARouter.getInstance().build("/home/main", "ap").navigation();  
  
// 构建标准的路由请求，通过Uri直接解析  
Uri uri;  
ARouter.getInstance().build(uri).navigation();  
  
// 构建标准的路由请求，startActivityForResult  
// navigation的第一个参数必须是Activity，第二个参数则是RequestCode  
ARouter.getInstance().build("/home/main", "ap").navigation(this, 5);  
  
// 直接传递Bundle  
Bundle params = new Bundle();  
ARouter.getInstance()  
 .build("/home/main")  
 .with(params)  
 .navigation();  
  
// 指定Flag  
ARouter.getInstance()  
 .build("/home/main")  
 .withFlags();  
 .navigation();  
  
// 获取Fragment  
Fragment fragment = (Fragment) ARouter.getInstance().build("/test/fragment").navigation();  
  
// 对象传递  
ARouter.getInstance()  
 .withObject("key", new TestObj("Jack", "Rose"))  
 .navigation();  
  
// 觉得接口不够多，可以直接拿出Bundle赋值  
ARouter.getInstance()  
 .build("/home/main")  
 .getExtra();  
  
// 转场动画(常规方式)  
ARouter.getInstance()  
 .build("/test/activity2")  
 .withTransition(R.anim.slide\_in\_bottom, R.anim.slide\_out\_bottom)  
 .navigation(this);  
  
// 转场动画(API16+)  
ActivityOptionsCompat compat = ActivityOptionsCompat.  
 makeScaleUpAnimation(v, v.getWidth() / 2, v.getHeight() / 2, 0, 0);  
  
// ps. makeSceneTransitionAnimation 使用共享元素的时候，需要在navigation方法中传入当前Activity  
  
ARouter.getInstance()  
 .build("/test/activity2")  
 .withOptionsCompat(compat)  
 .navigation();  
  
// 使用绿色通道(跳过所有的拦截器)  
ARouter.getInstance().build("/home/main").greenChannel().navigation();  
  
// 使用自己的日志工具打印日志  
ARouter.setLogger();

肖世锋

2018年7月30日