# **[Android业务组件化之子模块SubModule的拆分以及它们之间的路由Router实现](http://www.cnblogs.com/whoislcj/p/5860138.html)**

### **前言：**

     前面分析了APP的现状以及业务组件化的一些探讨（[Android业务组件化之现状分析与探讨](http://www.cnblogs.com/whoislcj/p/5853393.html)），以及通信的桥梁Schema的使用（[Android业务组件化之URL Schema使用](http://www.cnblogs.com/whoislcj/p/5825333.html)），今天重点来聊下子模块SubModule的拆分以及它们之间的路由Router实现。本篇涉及的相关知识比较多，阅读本篇之间需要大致了解一下Java的注解（[Java学习之注解Annotation实现原理](http://www.cnblogs.com/whoislcj/p/5671622.html)）、Java的动态代理机制（[Java设计模式之代理模式（Proxy）](http://www.cnblogs.com/whoislcj/p/5693980.html)）等。业务组件化是一个循序渐进的过程，一开始很难就能拿出终极解决方案，还是一步步来走的比较踏实实在。

   业务组件化相关文章地址：

* [Android业务组件化之现状分析与探讨](http://www.cnblogs.com/whoislcj/p/5853393.html)
* [Android业务组件化之URL Scheme使用](http://www.cnblogs.com/whoislcj/p/5825333.html)
* [Android业务组件化之子模块SubModule的拆分以及它们之间的路由Router实现](http://www.cnblogs.com/whoislcj/p/5860138.html)

### **我们首先搞清楚什么是业务组件？**

    搞清楚这个对我们来说至关重要，否则很难拆分业务与依赖库，也很难搞清楚哪些类应该放在业务子模块里面哪些类应该放在依赖库里面。

#### **1.）和业务无关**

     完全和业务没有一点关系，比如项目中常用的各种Utils工具类，一些公共自定义控件例如显示圆角图片的ImageView等

#### **2.）弱业务**

    为什么称之为弱业务，原因就是这些不是完整的业务，但是又和APP业务相关，比如我们的网络请求，数据库操作等。

#### **3.）业务**

   这个就是我们针对要拆分的业务组件，一个完整的独立的业务线才能称之为业务，比如我们APP的登录注册业务等。

### **业务组件的拆分粒度？**

    业务组件的拆分将是整个整改的重点，关于拆分的粒度也将成为讨论的焦点，到底是粗一点好还是细一点好？粗一点更容易拆分，细一点更容易解耦灵活度高，这个根据实际情况来定，由于我们项目整改的过程中不能影响到新需求的开发，开始还是以粗一点的粒度进行拆分，先把大致几个业务拆分出来，后期慢慢再做细。

### **子模块SubModule拆分：**

#### **1.）子模块没有拆分之间**

  页面跳转

Intent intent = new Intent(this, XXX.class);

startActivity(intent);

 数据传递

  直接页面startActivityForResult返回获取, 间接页面通过存储或者变量 ，或者借助开源框架EventBus等传递

没有拆分的这种开发方式，其实使用起来简单方便，但是这种显示调用没有任务文档，每一个跳转页面都要和相关开发人员沟通，沟通成本比较高。

#### **2.）拆分子模块之后**

 拆分子模块之后，任何模块之间的跳转都要采用路由Router中转，需要在相关Activity中配置如下信息

[IMG_256](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

<activity

android:name=".GoodsDetailActivity"

android:theme="@style/AppTheme">

<intent-filter>

<data

android:host="goods"

android:path="/goodsDetail"

android:port="8888"

android:scheme="xl"/>

<category android:name="android.intent.category.DEFAULT"/>

<action android:name="android.intent.action.VIEW"/>

<category android:name="android.intent.category.BROWSABLE"/>

</intent-filter>

</activity>

[IMG_257](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

页面跳转

[IMG_258](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

/\*\*

\* 通过uri跳转指定页面

\*

\* @param url

\*/

private void openRouterUri(String url) {

PackageManager packageManager = mContext.getPackageManager();

Intent intent = new Intent(Intent.ACTION\_VIEW, Uri.parse(url));

intent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

List<ResolveInfo> activities = packageManager.queryIntentActivities(intent, 0);

boolean isValid = !activities.isEmpty();

if (isValid) {

mContext.startActivity(intent);

}

}

[IMG_259](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

数据的传递同样可以直接页面startActivityForResult返回获取, 间接页面通过存储或者变量 ，或者借助开源框架EventBus等传递，但是这些弱业务公共数据统一放在依赖库里。模块之间不存在代码引用。通过这种方式android，iOS与H5客户端可以通过一些简单的url来实现跳转了，通维护一份文档来约束各个页面的参数。

#### **3.）路由Router简单实现**

   1.定义两个注解参数，一个标示URI注解，一个标示参数

RouterUri.java

[IMG_260](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

@Documented

@Target(METHOD)

@Retention(RUNTIME)public @interface RouterUri {

String routerUri() default "";

}

[IMG_261](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

RouterParam.java

[IMG_262](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

@Documented

@Target(PARAMETER)

@Retention(RUNTIME)public @interface RouterParam {

String value() default "";

}

[IMG_263](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

 2.定义一个URI协议接口

IRouterUri.java

[IMG_264](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

public interface IRouterUri {

@RouterUri(routerUri = "xl://goods:8888/goodsDetail")//请求Url地址

String jumpToGoodsDetail(@RouterParam("goodsId") String goodsId, @RouterParam("des") String des);//参数商品Id 商品描述

}

[IMG_265](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

3.定义一个单例，内部通过动态代理机制实现跳转

[IMG_266](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

public class XLRouter {

private final static String TAG = XLRouter.class.getSimpleName();

private static XLRouter mInstance;

private IRouterUri mRouterUri;

private Context mContext;

/\*\*

\* 获取单例引用

\*

\* @return 单例

\*/

public static XLRouter getInstance(Context context) {

XLRouter inst = mInstance;

if (inst == null) {

synchronized (XLRouter.class) {

inst = mInstance;

if (inst == null) {

inst = new XLRouter(context.getApplicationContext());

mInstance = inst;

}

}

}

return inst;

}

/\*\*

\* 构造函数

\*

\* @param mContext application

\*/

private XLRouter(Context mContext) {

this.mContext = mContext;

mRouterUri = create(IRouterUri.class);

}

/\*\*

\* 返回Api

\*/

public IRouterUri routerUri() {

return mRouterUri;

}

public IRouterUri create(Class<?> aClass) {

return (IRouterUri) Proxy.newProxyInstance(aClass.getClassLoader(), new Class<?>[]{aClass},

new InvocationHandler() {

@Override

public Object invoke(Object proxy, Method method, Object... args) throws Throwable {

StringBuilder stringBuilder = new StringBuilder();

RouterUri reqUrl = method.getAnnotation(RouterUri.class);

Log.e(TAG, "IReqApi---reqUrl->" + reqUrl.routerUri());

stringBuilder.append(reqUrl.routerUri());

//Type[] parameterTypes = method.getGenericParameterTypes();//获取注解参数类型

Annotation[][] parameterAnnotationsArray = method.getParameterAnnotations();//拿到参数注解

//Annotation[] annotation = method.getDeclaredAnnotations();

int pos = 0;

for (int i = 0; i < parameterAnnotationsArray.length; i++) {

Annotation[] annotations = parameterAnnotationsArray[i];

if (annotations != null && annotations.length != 0) {

if (pos == 0) {

stringBuilder.append("?");

} else {

stringBuilder.append("&");

}

pos++;

RouterParam reqParam = (RouterParam) annotations[0];

stringBuilder.append(reqParam.value());

stringBuilder.append("=");

stringBuilder.append(args[i]);

Log.e(TAG, "reqParam---reqParam->" + reqParam.value() + "=" + args[i]);

}

}

//下面就可以执行相应的跳转操作 openRouterUri(stringBuilder.toString());

return null;

}

});

}

/\*\*

\* 通过uri跳转指定页面

\*

\* @param url

\*/

private void openRouterUri(String url) {

PackageManager packageManager = mContext.getPackageManager();

Intent intent = new Intent(Intent.ACTION\_VIEW, Uri.parse(url));

intent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

List<ResolveInfo> activities = packageManager.queryIntentActivities(intent, 0);

boolean isValid = !activities.isEmpty();

if (isValid) {

mContext.startActivity(intent);

}

}

}

[IMG_267](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

4.调用方式

XLRouter.getInstance(context).routerUri().jumpToGoodsDetail("1000110002","goods des");

### **总结：**

   要实现真正的业务组件化任重而道远，我们这里实现第一步拆分子模块，让各个模块的代码各自维护，先解耦他们之间的依赖关系，在app壳工程通过compile project(':umeng\_social\_sdk\_library')这种方式选择性接入哪个子模块，开发过程中开发哪个模块就引入哪个模块，测试环节也是如此，测试哪个模块打包引入哪个模块，最终测试需要引入全部相关模块，测试上线。