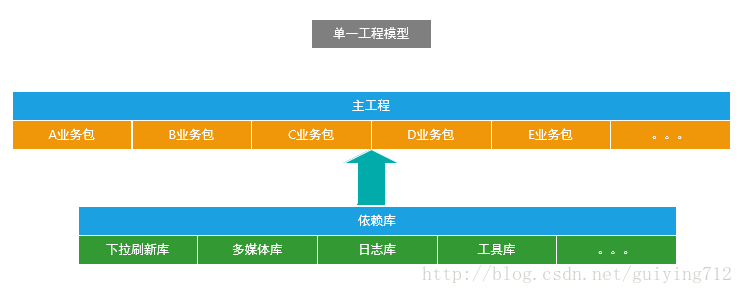
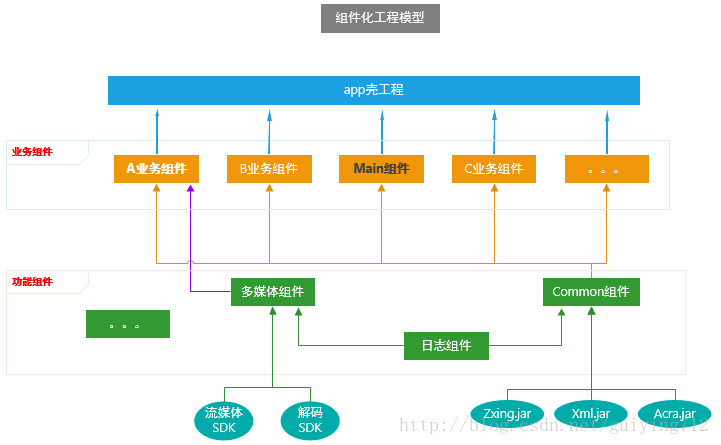
# 安卓工程组件化开发方案

## 一、为什么要项目组件化

随着APP版本不断的迭代，新功能的不断增加，业务也会变的越来越复杂，APP业务模块的数量有可能还会继续增加，而且每个模块的代码也变的越来越多，这样发展下去单一工程下的APP架构势必会影响开发效率，增加项目的维护成本，每个工程师都要熟悉如此之多的代码，将很难进行多人协作开发，而且Android项目在编译代码的时候电脑会非常卡，又因为单一工程下代码耦合严重，每修改一处代码后都要重新编译打包测试，导致非常耗时，最重要的是这样的代码想要做单元测试根本无从下手，所以必须要有更灵活的架构代替过去单一的工程架构。





组件化开发简单来说：开发的时候moudle作为application开发, 正式发布的时候将moudle作为依赖模块，集成到app壳工程中去。

## 二、组件化开发实施流程

### 1、开发时各Moudle的属性切换问题

Android Studio中的Module主要有两种属性，分别为：

application属性，可以独立运行的Android程序，也就是我们的APP；

apply plugin: ‘com.android.application’

library属性，不可以独立运行，一般是Android程序依赖的库文件；

apply plugin: ‘com.android.library’

Module的属性是在每个组件的 build.gradle 文件中配置的，当我们在组件模式开发时，业务组件应处于application属性，这时的业务组件就是一个 Android App，可以独立开发和调试；而当我们转换到集成模式开发时，业务组件应该处于 library 属性，这样才能被我们的“app壳工程”所依赖，组成一个具有完整功能的APP。但是我们如何让组件在这两种模式之间自动转换呢？ 看下面步骤：

（1）我们在gradle.properties 文件中添加静态常量 isModule=false/true 用于标志开发模式的切换（集成开发模式（moudle作为library） 和 组件开发模式（moudle作为app）的切换）

（2）然后我们在各个业务组件的build.gradle中读取 isModule，但是 gradle.properties 还有一个重要属性： gradle.properties 中的数据类型都是String类型，使用其他数据类型需要自行转换；也就是说我们读到 isModule 是个String类型的值，而我们需要的是Boolean值，代码如下：

if (isModule.toBoolean()) {

apply plugin: 'com.android.application'

} else {

apply plugin: 'com.android.library'

}

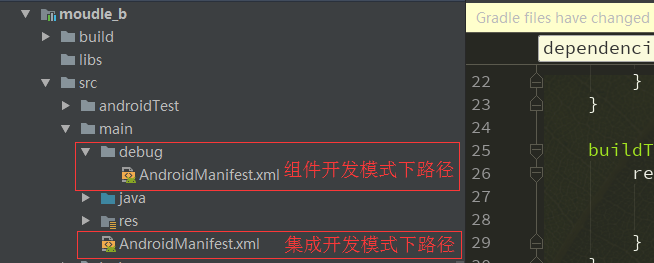
这样我们第一个问题，moudle属性切换的问题就解决了。

### 2、组件之间AndroidManifest合并问题

在 AndroidStudio 中每一个组件都会有对应的 AndroidManifest.xml，用于声明需要的权限、Application、Activity、Service、Broadcast等，当项目处于组件模式时，业务组件的 AndroidManifest.xml 应该具有一个 Android APP 所具有的的所有属性，尤其是声明 Application 和要 launch的Activity，但是当项目处于集成模式的时候，每一个业务组件的 AndroidManifest.xml 都要合并到“app壳工程”中，要是每一个业务组件都有自己的 Application 和 launch的Activity，那么合并的时候肯定会冲突。怎样解决此问题呢？（两套文件）看下面操作：

针对不同开发模式，使用不同的AndroidManifest.xml 文件：我们需要在build.gradle 中指定不同开发模式下的 AndroidManifest.xml 的位置，为组件开发模式下的业务组件再创建一个 AndroidManifest.xml，然后根据isModule指定AndroidManifest.xml的文件路径，让业务组件在集成模式和组件模式下使用不同的AndroidManifest.xml。

这样我们第二个问题，表单冲突的问题（各组件AndroidManifest.xml合并的问题）就可以解决了。目录结构及配置代码如下两图。





### 集成模式下，组件之间Application冲突问题及解决办法

我们在组件化开发的时候，可能为了数据的问题每一个组件都会自定义一个Application类，如果我们在自己的组件中开发时需要获取 全局的Context，一般都会直接获取 application 对象，但是当所有组件要打包合并在一起的时候就会出现问题，因为最后程序只有一个 Application，我们组件中自己定义的 Application 肯定是没法使用的，因此我们需要想办法再任何一个业务组件中都能获取到全局的 Context，而且这个 Context 不管是在组件开发模式还是在集成开发模式都是生效的。按照以上需求我们有以下解决思路：

创建一个Connmon组件，这个组件中封装了项目中需要的各种基础功能、各组件共用的三方库、各种Base类等等，然后每一个业务组件都依赖这个Common组件，在这个Common组件中有一个BaseApplication类继承Application,这个BaseApplication就是整个项目的基础Application对象，其他各务组件开发过程中需要自定义Application的时候，只需要继承Common组件中的这个BaseApplication类就可以了。

这样通过继承BaseApplication的方式，我们就解决了集成模式下各组件之间Application冲突的问题。

### 组件之间的调用和通信

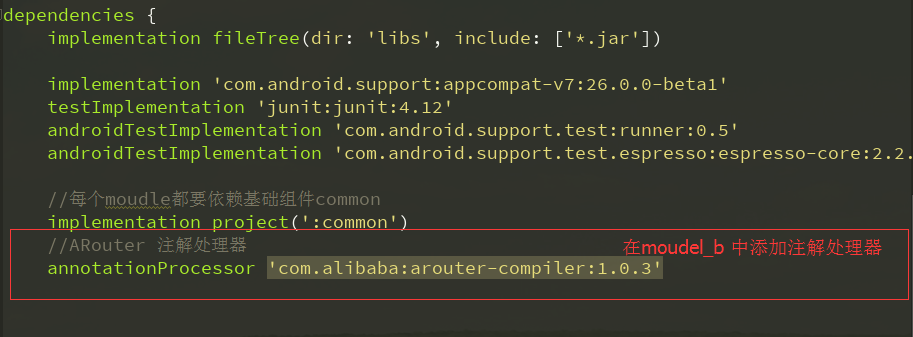
在组件化开发的时候，组件之间是没有依赖关系，我们不能在使用显示调用来跳转页面了，因为我们组件化的目的之一就是解决模块间的强依赖问题，假如现在要从A业务组件跳转到业务B组件，并且要携带参数跳转，这时候怎么办呢？而且组件这么多怎么管理也是个问题，这时候就需要引入“路由”的概念了，由本文开始的组件化模型下的业务关系图可知路由就是起到一个转发的作用。

这里介绍开源库“ARouter”，阿里巴巴开发的一个非常好用的Android路由框架。具体使用方法如下步骤：

1. 在Common组件中添加ARouter依赖（因为所有模块都依赖Common模块，所以我们将这个所有组件共用的三方库添加到Common组件中）。

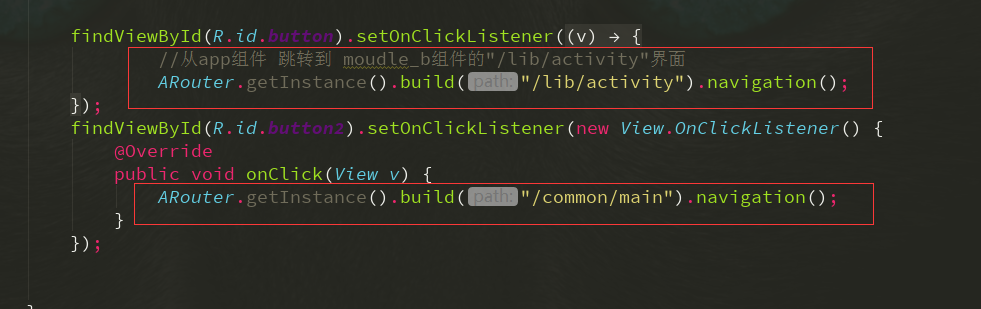


1. 在每一个业务组件的 build.gradle 都引入ARouter 的 Annotation处理器，我们将会在声明组件和Url的时候使用，annotationProcessor是Android官方提供的Annotation处理器插件，代码如下：



到这里ARouter配置就算完成了。

1. Arouter使用示例：



### 5、组件之间资源名冲突

因为我们拆分出了很多业务组件和功能组件，在把这些组件合并到“app壳工程”时候就有可能会出现资源名冲突问题，例如A组件和B组件都有一张叫做“ic\_back”的图标，这时候在集成模式下打包APP就会编译出错，解决这个问题最简单的办法就是在项目中约定资源文件命名规约，比如强制使每个资源文件的名称以组件名开始，这个可以根据实际情况和开发人员制定规则。