前言   
在移动开发横行，应用日渐饱和，开发周期，迭代周期要求越来越快的时代下，经常看见有群里小伙伴抱怨问题：

* Gradle编译一个项目需要10分钟、20分钟…..
* 这什么JB玩意儿，什么都忘一个类里放，怎么改呀（扣脑壳）….

这里给出两个方案：

* + 加大电脑内存，提升AndroidStudio的使用内存
  + 合理规划项目结构，组件化、模块化开发项目

第一个解决方案，貌似不太靠谱，总不能每次都换电脑吧？都已经硬件顶配还怎么换啊。。。下面我们谈谈第二点，使用组件化，模块化解决。

**什么是组件化**

大致来说，组件可以分为两大类，一类是**application**组件，一类是libs组件的module.

application组件是一个可运行的app.   
libs可以作为application的依赖，但是自身不可作为程序运行的存在。

**一般开发的缺点，为什么组件化**

* 实施者对基础模块，基础组件，中间层，上层业务的规划不合理
* 业务，组件，资源放在一起，任何一个改动都会消耗甚大，运行整个application
* 因为所有东西都一堆，项目不易维护，耦合度超高
* 不利于单独模块化测试

**组件化好处**

组件化描述完毕后，看组件化后有什么好处呢。

* 每个Module可以单独调试开发，节省编译时间
* 单独个模块开发可共享工具类，网络库等
* 对测试来说可以对单个模块进行快速测试
* 公司业务繁重可以不断复用模块，节省开发时间
* 对个人来说，可以积累个人的开发工具

**组件化弊端**

* 组件与组件之间存在业务联系
* 组件调用application
* 资源命名重复
* 引用的库版本不对应，使用冲突

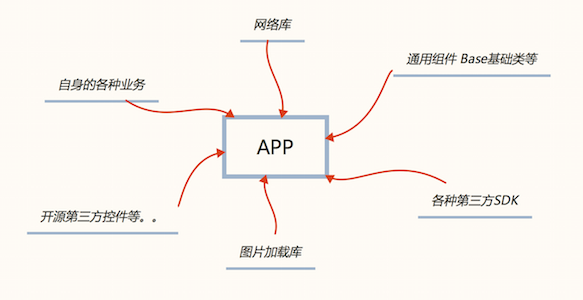
第一点：存在业务联系的可以归纳为同一个组件

第二点：在基础架构层建BaseApplication,统一使用

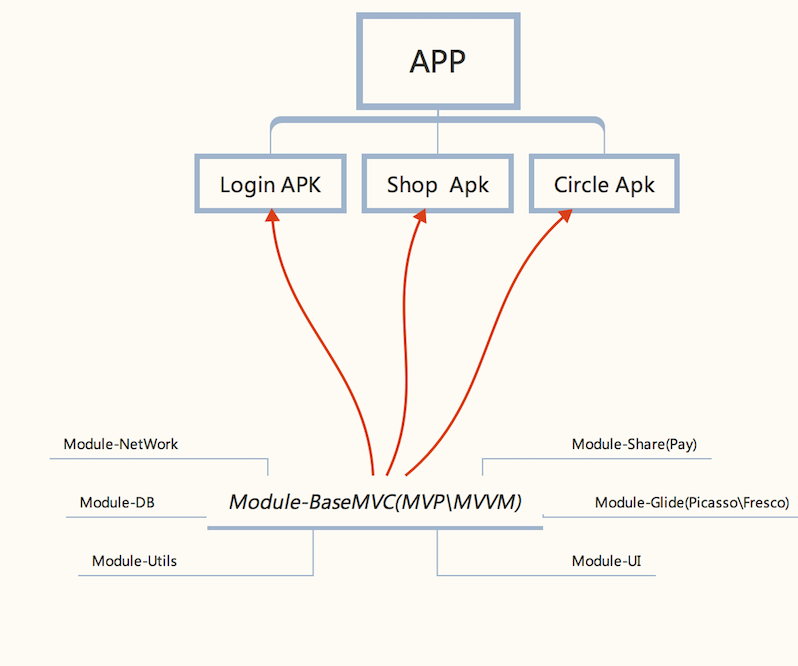
第三点：命名按照moduel的开头命名

第四点： [参考此文](http://blog.csdn.net/asddavid/article/details/53322689)

下面看两张图：   
**①：没有组件化、模块化的的项目**



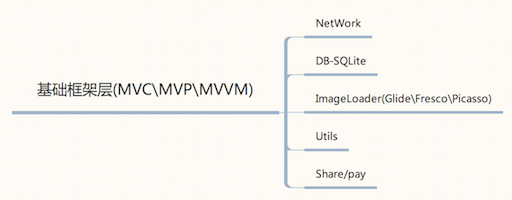
不用多说，什么都在application一个组件内部。

**②：具有一定架构并组件化、模块化**   


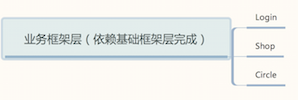
这张图也许有的兄弟看起来很模糊，马上我们将其拆分从最底部慢慢升上来看。

* SDK层   
  

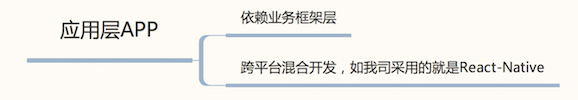
SDK层主要为Android的SDK以及我们需要使用的第三方SDK（地图、定位、直播等）

* 基础架构层   
  

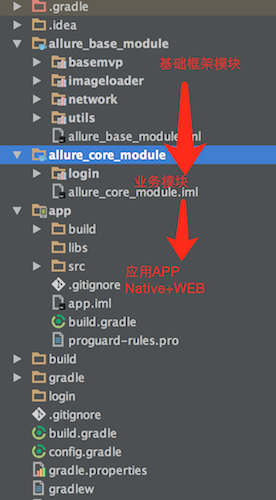
首先我们得选取一个整体架构模式，比如MVC。 其他模块全部依赖于Base的基础框架层完成，NetWork，SQLite、图片加载库，支付等组件模块以便给予业务层使用。

* 业务框架层   
  

此处简单列举三个例子，有Login、Shop、Circle三大组件模块，   
此3个业务模块需要什么即依赖基础框架层的Module完成。DEBUG期间可以单独作为application使用，当要正式打包时候将作为libs使用。

* 应用层   
     
  应用层即为application，依赖于业务框架（上诉的Login、Shop、Circle）完成的Native部分，如果有部分业务跨平台了，如HyBird，React-Native等，混合开发将其和Native部分综合即可完成一个App.

**项目组件化图例**



我们完成基础框架模块的newwork等模块后将其加入basemvp，看看如何在basemvp依赖：

apply plugin: 'com.android.library'

android {

compileSdkVersion rootProject.ext.android.compileSdkVersion

buildToolsVersion rootProject.ext.android.buildToolsVersion

defaultConfig {

minSdkVersion rootProject.ext.android.minSdkVersion

targetSdkVersion rootProject.ext.android.targetSdkVersion

versionCode rootProject.ext.android.versionCode

versionName rootProject.ext.android.versionName

}

buildTypes {

release {

minifyEnabled false

proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

}

}

}

dependencies {

compile fileTree(dir: 'libs', include: ['\*.jar'])

compile rootProject.ext.dependencies.appcompatV7

compile rootProject.ext.dependencies.design

compile project(':allure\_base\_module:imageloader')

compile project(':allure\_base\_module:network')

compile project(':allure\_base\_module:utils')

}

均依赖于basemvp的基础框架。

**业务模块**

apply plugin: 'com.android.library'

android {

compileSdkVersion rootProject.ext.android.compileSdkVersion

buildToolsVersion rootProject.ext.android.buildToolsVersion

defaultConfig {

minSdkVersion rootProject.ext.android.minSdkVersion

targetSdkVersion rootProject.ext.android.targetSdkVersion

versionCode rootProject.ext.android.versionCode

versionName rootProject.ext.android.versionName

}

buildTypes {

release {

minifyEnabled false

proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

}

}

}

dependencies {

compile fileTree(dir: 'libs', include: ['\*.jar'])

compile rootProject.ext.dependencies.appcompatV7

compile rootProject.ext.dependencies.design

compile project(':allure\_base\_module:basemvp')

}

依赖于基础框架模块完成业务逻辑的书写。

**最终APP**

apply plugin: 'com.android.application'

android {

compileSdkVersion rootProject.ext.android.compileSdkVersion

buildToolsVersion rootProject.ext.android.buildToolsVersion

defaultConfig {

minSdkVersion rootProject.ext.android.minSdkVersion

targetSdkVersion rootProject.ext.android.targetSdkVersion

versionCode rootProject.ext.android.versionCode

versionName rootProject.ext.android.versionName

}

buildTypes {

release {

minifyEnabled false

proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

}

}

}

dependencies {

compile fileTree(dir: 'libs', include: ['\*.jar'])

compile rootProject.ext.dependencies.appcompatV7

compile rootProject.ext.dependencies.design

compile project(':allure\_core\_module:login')//登陆

//其他模块

}

仅仅需要依赖 业务模块即可，至于你有多少业务模块取决于你的业务。

**总结**

看完整篇内容，相信大家对组件化有了一定的认识，并知道组件化的好处，为什使用它。   
个人认为，组件化的开发难度并不大，真正的难度在于理解当前公司的业务需求，并在其基础上能很好的解耦提高灵活度，所以具体是否组件化还是得看公司的业务发展。

若公司业务发展单一，是否组件化意义并不大，反而会加大自身开发成本，当业务已经成熟在回头来优化组件化也未尝不可。

[示例查看](http://download.csdn.net/detail/asddavid/9700178)