23.模块化&组件化

23.1.什么是模块化

原本一个 App模块 承载了所有的功能，而模块化就是拆分成多个模块放在不同的Module里面，每个功能的代码都在自己所属的 module 中添加

通常还会有一个通用基础模块module_common，提供BaseActivity/BaseFragment、图片加载、网络请求等基础能力，然后每个业务模块都会依赖这个基础模块。 业务模块之间有有依赖

但多个模块中肯定会有页面跳转、数据传递、方法调用 等情况，所以必然存在以上这种依赖关系，即模块间有着高耦合度。 高耦合度 加上 代码量大，就极易出现上面提到的那些问题了，严重影响了团队的开发效率及质量。

23.2.什么是组件化

组件化，去除模块间的耦合，使得每个业务模块可以独立当做App存在，对于其他模块没有直接的依赖关系。 此时业务模块就成为了业务组件。

23.3.组件化优点和方案

加快编译速度：每个业务功能都是一个单独的工程，可独立编译运行，拆分后代码量较少，编译自然变快。

提高协作效率：解耦 使得组件之间 彼此互不打扰，组件内部代码相关性极高。 团队中每个人有自己的责任组件，不会影响其他组件；降低团队成员熟悉项目的成本，只需熟悉责任组件即可；对测试来说，只需重点测试改动的组件，而不是全盘回归测试。

组件化方案

https://juejin.cn/post/6844904147641171981

宿主app 在组件化中，app可以认为是一个入口，一个宿主空壳，负责生成app和加载初始化操作。

业务层 每个模块代表了一个业务，模块之间相互隔离解耦，方便维护和复用。

公共层

既然是base，顾名思义，这里面包含了公共的类库。如Basexxx 、 toast,logutil，glide工具类，资源文件等

网络层

提供网络加载

三方库层

提供网络加载

组件化开发的问题点

https://juejin.cn/post/6881116198889586701#heading-26

https://www.jianshu.com/p/8b6e6a50e21e

https://juejin.cn/post/6844904147641171981#heading-5

23.4.组件独立调试

1.1gradle.properties中定义一个常量值 isModule 1.2 apply plugin根据boolean值判断 if (isModule.toBoolean()){ apply plugin: 'com.android.application' }else { apply plugin: 'com.android.library' } 1.3 配置 applicationId 和mainfest

23.5.组件间通信

跳转

ARouter 1.@Route 2.ARouter.getInstance().build("/xx/xx").navigation()

为彼此提供服务

既然首页组件可以访问购物车组件接口了，那就需要依赖购物车组件啊，这俩组件还是耦合

1.首先在commonlib模块里创建一个暴露方法的接口，并定义接口签名，同时继承 Iprovide 1.首先在commonlib模块里创建一个暴露方法的接口，并定义接口签名，同时继承 Iprovider 接口

2.然后在home模块中继承commonlib里定义的接口，并实现签名方法。

3.Arouter的 @Router注解调用

r 接口

2.然后在home模块中继承commonlib里定义的接口，并实现签名方法。

3.Arouter的 @Router注解调用

23.6.Aplication动态加载

组件有时候也需要获取应用的Application，也需要在应用启动时进行初始化。这就涉及到组件的生命周期管理问题。

假设我们有组件ModuleA、ModuleB、ModuleC，这3个组件内分别有ModuleAAppLike、ModuleBAppLike、ModuleCAppLike，那么我们在壳工程集成时，怎么去组装他们呢。最简单的办法是，在壳工程的Application.onCreate()方法里执行

问题1： 组件初始化的先后顺序，上层业务组件是依赖下层业务组件的， 那么我们在加载组件时，必然要先加载下层组件，否则加载上层组件时可能会出现问题。

问题2： 新增加一个组件，去修改壳工程代码，不利于代码维护。

解决

1.定义一个注解来标识实现了BaseAppLike的类。

2.通过APT技术，在组件编译时扫描和处理前面定义的注解，生成一个BaseAppLike的代理类

3.组件集成后在应用的Application.onCreate()方法里，调用组件生命周期管理类的初始化方法。

4.组件生命周期管理类的内部，扫描到所有的BaseAppLikeProxy类名之后，通过反射进行类实例化。

难点

需要了解APT技术，怎么在编译时动态生成java代码；

1创建一个注解类 2定义baseapplication接口,设置优先级 3继承AbstractProcessor process中 ，生成代理类，并写入到文件里（ StringBuilder 。 append ）

应用在运行时，怎么能扫描到某个包名下有多少个class，以及他们的名称呢；

如果有十多个组件里都有实现IAppLike接口的类，最终我们也会生成10多个代理类，这些代理类都是在同一个包下面运行时读取手机里的dex文件，从中读取出所有的class文件名，根据我们前面定义的代理类包名，来判断是不是我们的目标类，这样扫描一遍之后，就得到了固定包名下面所有类的类名了通常一个安装包里，加上第三方库，class文件可能数以千计、数以万计，这让人有点杀鸡用牛刀的感觉。每次应用冷启动时，都要读取一次dex文件并扫描全部class，这个性能损耗是很大的，我们可以做点优化，在扫描成功后将结果缓存下来，下次进来时直接读取缓存文件

在应用编译成apk时，就已经全量扫描过一次所有的class，并提取出所有实现了IAppLike接口的代理类呢，这样在应用运行时，效率就大大提升了。答案是肯定的，这就是gradle插件、动态插入java字节码技术。

采用gradle插件技术，在应用打包编译时，动态插入字节码来实现

https://www.jianshu.com/p/3ec8e9574aaf

1Gradle Transform在打包前去扫描所有的class文件

Gradle Transform技术，简单来说就是能够让开发者在项目构建阶段即由class到dex转换期间修改class文件

nputs就是所有扫描到的class文件或者是jar包，一共2种类型 遍历查找所有的jar包

2通过ASM动态修改字节码

init方法里找到 AppLifeCycleManager里的addClassFile()方法，我们在这个方法里插入字节码 通过反射创建的实例

23.7.ARouter原理

https://juejin.cn/post/6885932290615509000#heading-1

https://juejin.cn/post/6844903648690962446#heading-0

1.ARouter 路由表生成原理

@Route注解，会在编译时期通过apt生成一些存储path和activityClass映射关系的类文件

APT是Annotation Processing Tool的简称,即注解处理工具。它是在编译期对代码中指定的注解进行解析，然后做一些其他处理（如通过javapoet生成新的Java文件）

第一步：定义注解处理器，用来在编译期扫描加入@Route注解的类，然后做处理。 这也是apt最核心的一步，新建RouterProcessor 继承自 AbstractProcessor,然后实现process方法。在项目编译期会执行RouterProcessor的process()方法，我们便可以在这个方法里处理Route注解了

第二步，在process()方法里开始生成EaseRouter_Route_moduleName类文件和EaseRouter_Group_moduleName文件。这里在process()里生成文件用javapoet生成java文件，就会用 JavaPoet 生成 Group、Provider 和 Root 路由文件，路由表就是由这些文件组成的，内容loadInto方法通过传入一个特定类型的map就能把分组信息放入map里为，一个map(其实是两个map，一个保存group列表，一个保存group下的路由地址和activityClass关系)保存了路由地址和ActivityClass的映射关系，然后通过map.get("router address") 拿到AncivityClass，通过startActivity()调用就好了

2.ARouter 路由表加载原理

https://github.com/Xiasm/EasyRouter/wiki/%E6%A1%86%E6%9E%B6%E7%9A%84%E5%88%9D%E5%A7%8B%E5%8C%96

app进程启动的时候会拿到这些类文件，把保存这些映射关系的数据读到内存里(保存在map里)到这些类文件便可以得到所有的routerAddress---activityClass映射关系 去扫描apk中所有的dex，遍历找到所有包名为packageName的类名，然后将类名再保存到classNames集合里

1. 读取apk中所有的dex文件 2.然后判断类的包名是否为 “com.alibaba.android.arouter.routes”，获取到注解处理器生成的类名时，就会把这些类名保存 SharedPreferences 中，下次就根据 App 版本判断，如果不是新版本，就从本地中加载类名，否则就用 ClassUtils 读取类名。 3.就会根据类名的后缀判断类是 IRouteRoot 、IInterceptorGroup 还是 IProviderGroup ，然后根据不同的类把类文件的内容加载到索引中。获取到映射关系

3.ARouter 跳转原理

路由跳转的时候，通过build()方法传入要到达页面的路由地址，ARouter会通过它自己存储的路由表找到路由地址对应的Activity.class(activity.class = map.get(path))，然后new Intent()

1.在build的时候，传入要跳转的路由地址，build()方法会返回一个Postcard对象，我们称之为跳卡。然后调用Postcard的navigation()方法完成跳转

2.ARouter会通过它自己存储的路由表找到路由地址对应的Activity.class(activity.class = map.get(path))，然后new Intent()