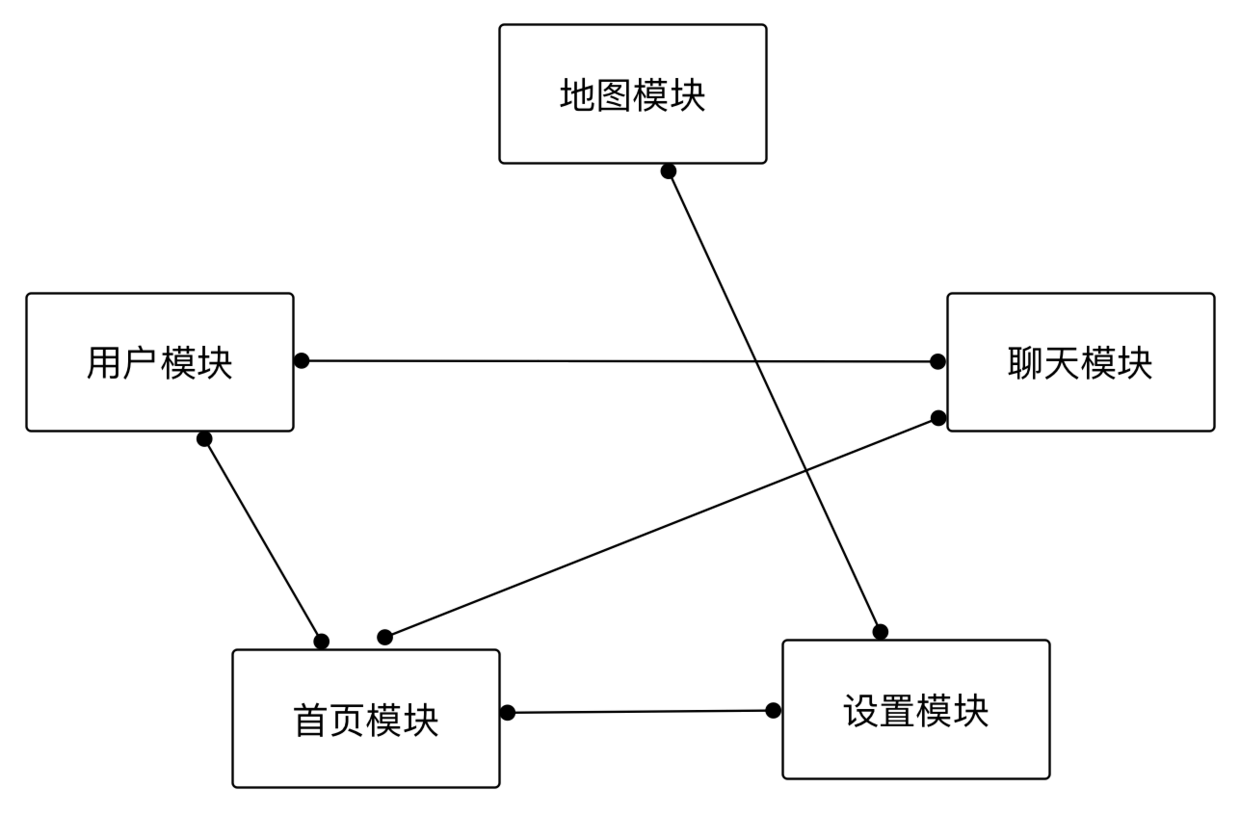
# IOS 组件化开发

## 组件化的由来

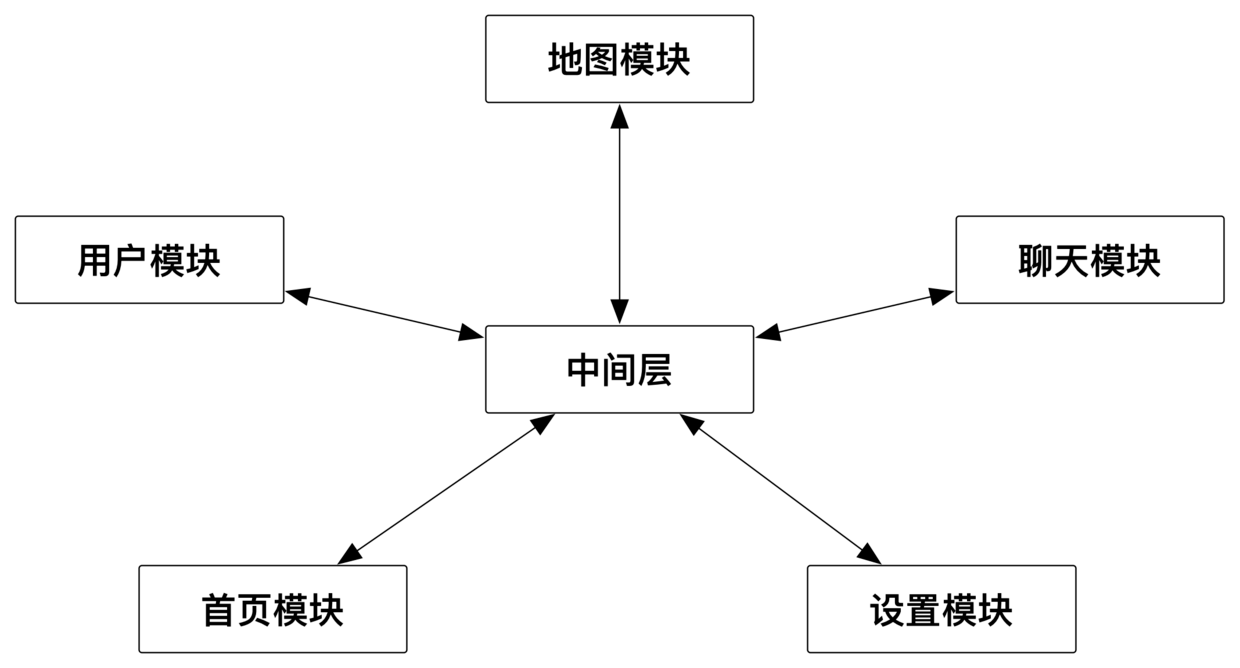
随着移动互联网的不断发展，移动端app代码量和业务越来越多，像淘宝，蘑菇街这样的大项目，现有的单工程 + MVC架构越显吃力。淘宝在13年开启了插件化架构的重构，后来在14年迎来了手机淘宝有史以来最大的重构，将其彻底重构为组件化架构。

## 蘑菇街的组件化架构

在一个项目越来越大，开发人员越来越多的情况下，项目会遇到很多问题。业务模块间划分不清晰，模块之间耦合度很大，非常难维护。所有模块代码都编写在一个项目中，测试某个模块或功能，需要编译运行整个项目。



为了解决上面的问题，可以考虑加一个中间层来协调模块间的调用，所有的模块间的调用都会经过中间层中转。**(**注意看两张图的箭头方向**)**



添加中间层

但是发现增加这个中间层后，耦合还是存在的。中间层对被调用模块存在耦合，其他模块也需要耦合中间层才能发起调用。这样还是存在之前的相互耦合的问题，而且本质上比之前更麻烦了。大体结构所以应该做的是，只让其他模块对中间层产生耦合关系，中间层不对其他模块发生耦合。

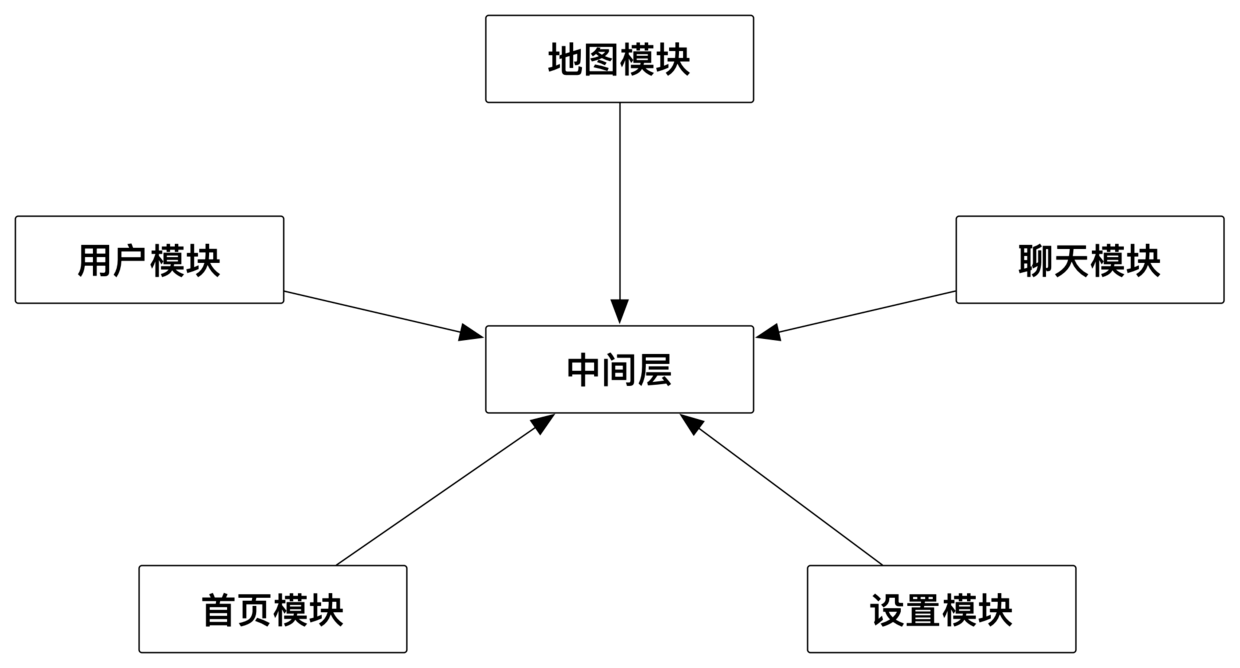
对于这个问题，可以采用组件化的架构，将每个模块作为一个组件。并且建立一个主项目，这个主项目负责集成所有组件。这样带来的好处是很多的：

1.业务划分更佳清晰，新人接手更佳容易，可以按组件分配开发任务。

2.项目可维护性更强，提高开发效率。

3.更好排查问题，某个组件出现问题，直接对组件进行处理。

4.开发测试过程中，可以只编译自己那部分代码，不需要编译整个项目代码。



进行组件化开发后，可以把每个组件当做一个独立的**app**，每个组件甚至可以采取不同的架构，例如分别使用MVVM、MVC、MVCS等架构。

# 部署流程

## 技术点

1. Linux系统
2. Gitlab等开源版本管理系统
3. Gitlab-runner / Gitlab-CI 持续集成系统
4. CocoaPods（IOS方案）
5. CocoaPods-Packager（IOS方案）：私有仓库打包工具
6. Maven（Android方案）
7. 定义中间层路由方案：通过路由进行组件间消息转发（个人推荐MGJRouter思想，具体技术方案，需要个业务工程师商定）

业务组件

XXRouter

主业务

组件1

组件2

基础组件

资源组件

网络组件

缓存组件

组件化方案路由协议

8.内存管理

IOS系统对内存的控制是极其严格的，系统对每个APP占用内存情况严格把控。如果app内存过高，APP会随时触发didReceiveMemoryWarning方法。如果工程师处理不当，程序立即闪退。目前我想到两种方案实现组件间内存管理：Protocol和XXRouter方式。

Protocol协议：有内存警告时,各组件间通过协议抛出内存警告，其他哪个组件需要处理该警告，就实现该协议。

XXRouter：类似第6点的 消息转发路由。

## 组件化开发APP流程图（IOS）

代码托管工具

CocoaPods

组件1

主业务

组件2

组件…

# IOS代码规范

## Main方法、AppDelegate，info.plist、Podfile等

此类文件请保持组件特性，不要提交到代码托管仓库。关于原本需要在AppDelegate内完成的初始化等操作，需要通过中间件路由实现（技术方面需要根据业务详细划分）

## 类、方法名命名规范（避免命名冲突）

如果项目使用Objective-C开发，要严格遵守语言命名规范（本文档建议使用小驼峰命名法），并且遵循各组件间命名规范，防止冲突。

如果组件内部使用了swift，请使用命名空间区

## 数据持久化方案

IOS常用的持久化方案有五种：

1. plist文件（属性列表）、
2. preference）（偏好设置）、
3. NSKeyedArchiver（归档）、
4. SQLite、
5. CoreData

组件化前，需要了解各组件内部使用的持久化方案。尽量协商使用同一套持久化方案；不同的持久化方案需要规避的问题不同，业务工程师需要细化讨论。

## Objective-C Runtime 的使用

各组件内部一定谨慎使用Runtime，不允许使用Runtime修改系统生命周期、公开接口、中间层路由等函数，如果需要动态添加属性，谨慎使用runtime关联函数，确保key值唯一（详细技术细节，个人认为需要业务工程师在一起根据各自组件的特点集中讨论）。

## Category的使用

对于Category的使用也要保持高警惕，不要随意覆盖系统生命周期方法，如果需要对UIView等常用组件添加方法，要严格遵循各组件间命名规范（详见 四、命名规范）

## 第三方库引用规范（规避三方开源项目冲突问题）

需要检查各个组件对第三方库引用情况，尽量确保不存在相同的第三方库；如存在强引用，业务复杂，不宜解耦的组件，可以各自组件内部分别引用相同三方库，但是必须保证三方库版本一致

# 商定协议

## 登陆机制

通过XXRouter路由实现（具体需要根据业务协商路由URL，参数等）

## 更新机制

1.如果是组件内部支持热更新，可以自行升级。

2.如果更新了组件的主业务逻辑需要重新编译代码的，需要通知主业务方更新。

## 消息推送机制（IOS）

如果组件内部需要实现APNs机制，需要跟主业务方沟通，将Apple userToken等参数传递到主业务组件的Delegate里集中处理（个人建议将APNs模块放在基础组件里面公用）。

# 人员配置

基础组件部分：需要随着各组件的业务持续更新，优化

各组件、主业务组件：按照各自需求配备人员

需要跟 AGQJ 商议的协议：横竖屏协议