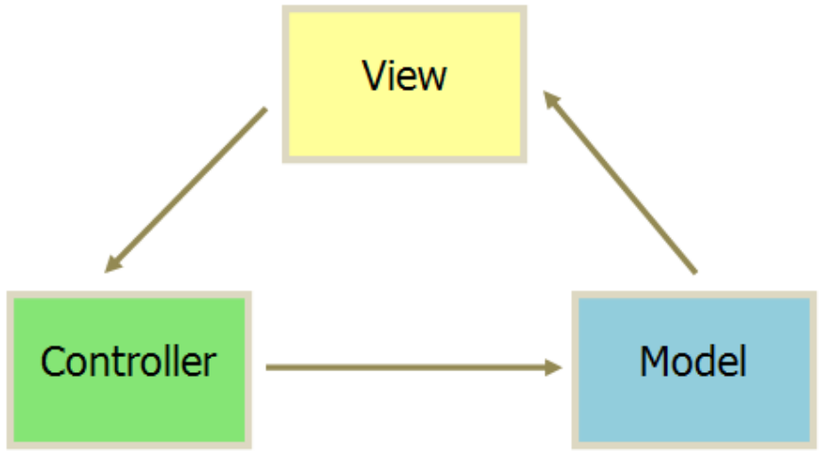
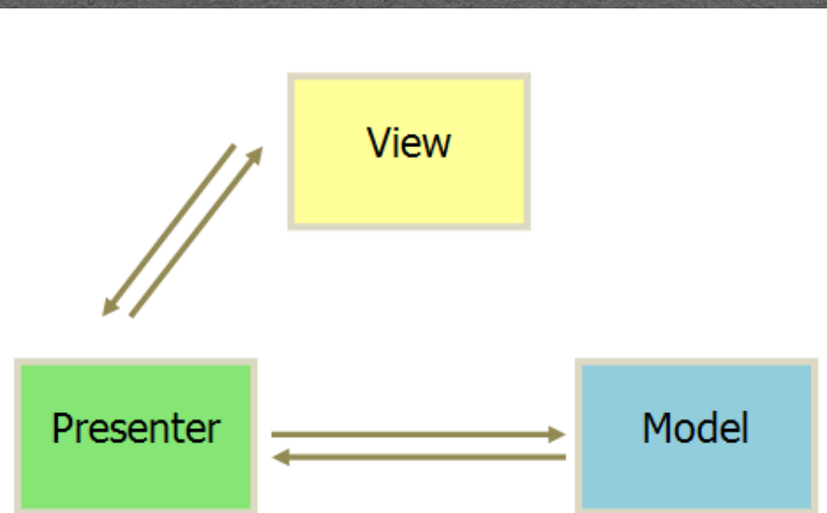
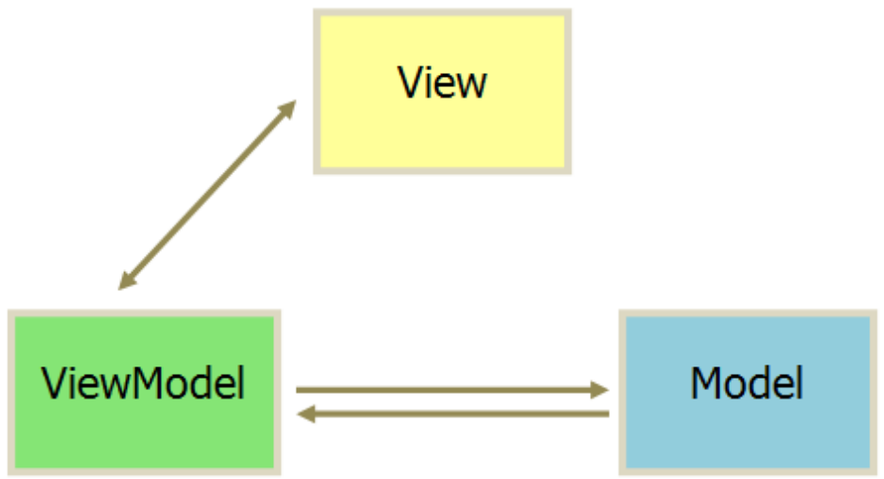
移动端架构初探心得分享

dragonli

常见架构


View
Controller
Model

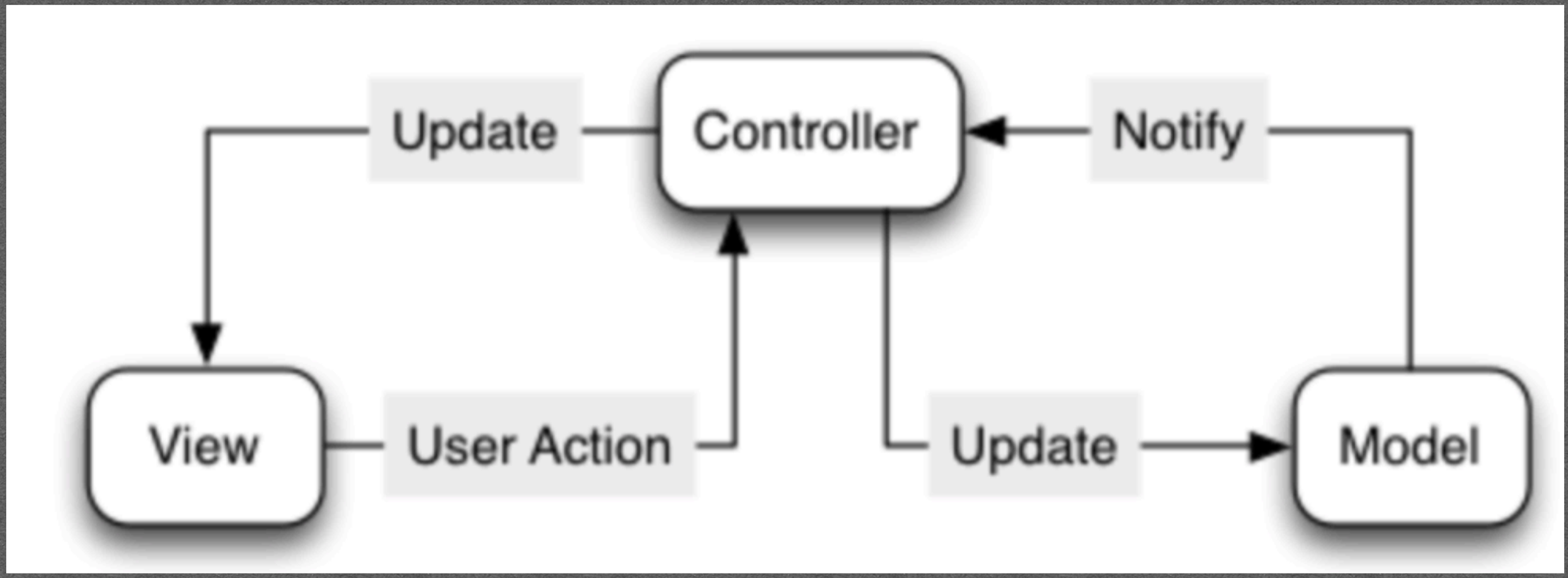

View
Presenter
Model


View
ViewModel
Model

MVC

- Model View Controller
- 苹果说，在MVC下，所有的对象被归类为一个model，一个 view，或一个controller
- Model持有数据，View显示与 户交互的界，Controller协调model和view之间的交互
- Model:数据管者
- View:数据展示者
- Controller:数据加 者(根据业务需求进 数据流调配)

MVC 示意图



- MVC模式最早由Trygve Reenskaug在1978年提出，是施乐帕罗奥多研究中心（Xerox PARC）在20世纪80年代为程序语言SmallTalk发明的一种软件架构

- 如此设计是否存在弊端? 如何解决？

MVC弊端

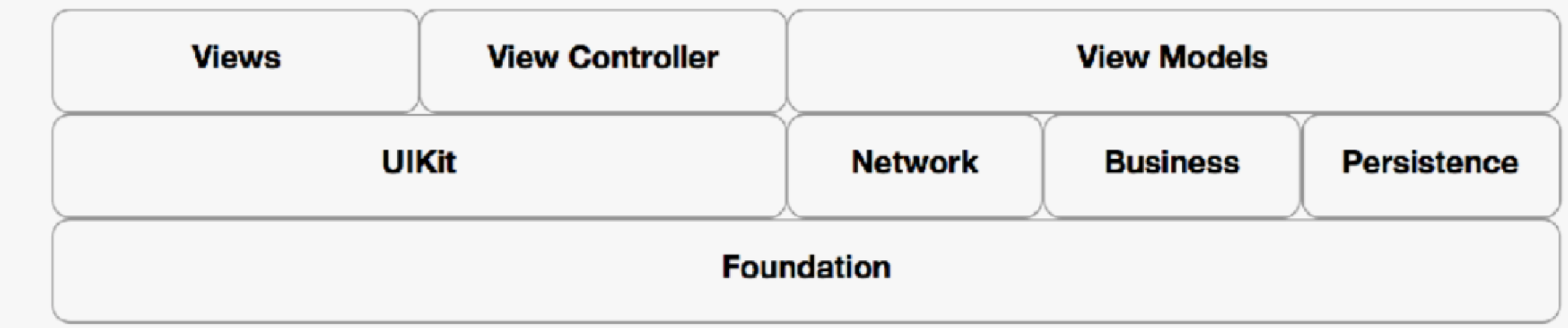
- Massive View Controller
- API通信的代码应该放在哪？
- 如何进单元测试?
- 共识:给Controller瘦身

核心原则

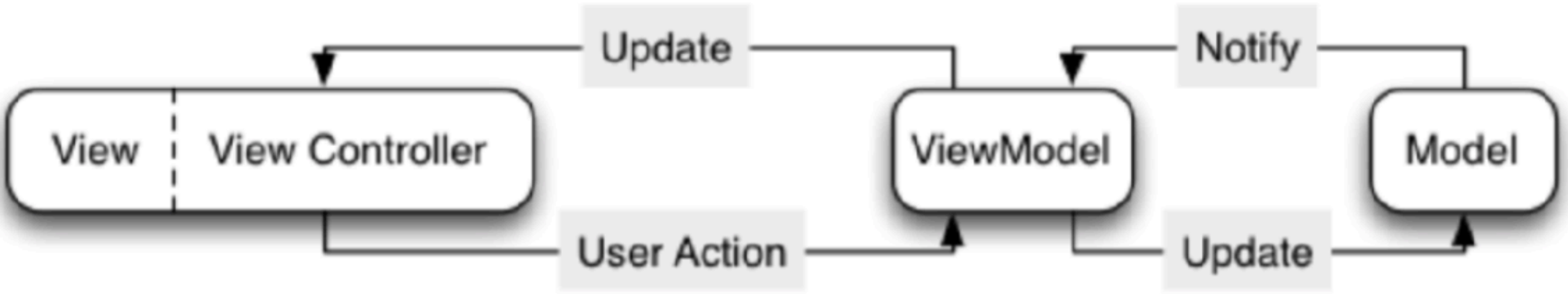
- UI和逻辑相分离
- 术语叫做:“解耦”
- 直的说:分明确
- 进而演化后续的 MVVM,VIPER 等

MVVM

- 什么是MVVM?
- Model View ViewModel
- Where is Controller?



MVVM示意图



MVVM是一种软件架构模式。最早由微软架构师Ken Cooper和Ted Peters开发

MVVM:Model ViewModel Controller View

- View:由MVC中View和Controller组成，负责UI展示，使VM中的属性，(响应用户操作)触发VM中的方法
- ViewModel:从MVC中controller中抽取出来，用于处理(展示)逻辑,提供从model中获取的并转换为view可直接展示的数据,暴露公开的属性和方法供view使
- Model:与MVC中一样，包括数据模型，访问数据库网络请求(业务逻辑)
- Controller:除上面说的view，还负责管视图的生周期，屏幕旋转，将view和VM进配对

MVVM优点

- 解耦，瘦身，层级明确:view和model的变化互不影响，易于团队协作
- 三高 :可读性，重用性，可维护性
- 可进单元测试，修复bug更容易

MVVM缺点

- MVVM相 MVC代码 有所增加
- MVVM相 MVC在代码编写之前需要有清晰的模式思路
- 没有一个明确的定义，每个人理解可能都不一样

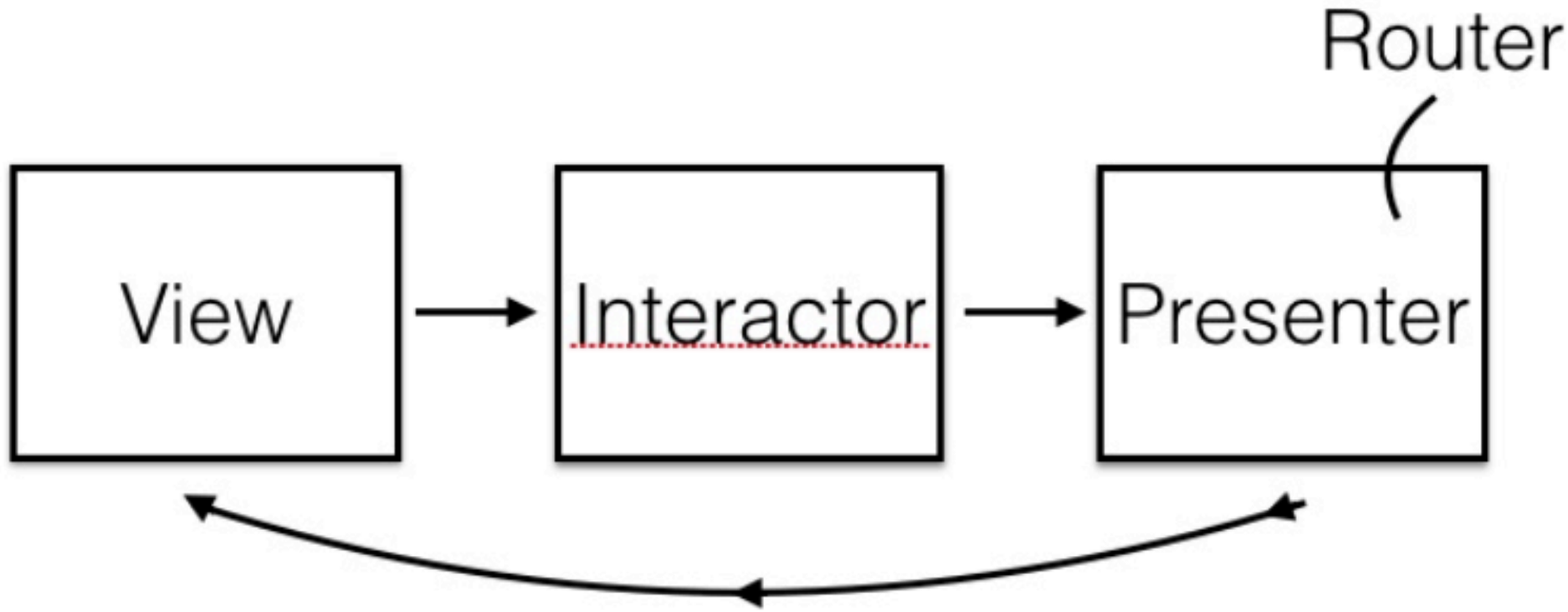
各自优点

Controller	MVC 架构核心, 用于控制数据和视图之间的交互
Presenter	MVP 架构核心, 用于数据和视图之间的中间件
ViewModel	MVVM 双向绑定, 移动端通常仅为单项数据流
Model	MVC中的M, 数据容器, 类似于JAVA中的pojo层
View	MVC中的V, 视图容器, 前端开发重点关注区域

什么是 VIPER

- **V: View 视图**：在这里并不是指传统的`UIView`或其子类，事实上它就是`UIViewController`，在前面所说到，`VIPER`架构主要是将`MVC`架构中的`Controller`进行更加细致的划分，而`View(视图)`层则是主要负责一些视图的显示、布局，用户事件的接受以及转发，基本的显示逻辑处理等等工作。
- **I: Interactor 交互器**：其为`VIPER`的中心枢纽，主要负责交互的工作，例如数据的请求（网络请求、本地持久化层请求）、某些业务逻辑的处理，在这里我们得到的数据是原始数据，需要经过解析处理转换成能够直接应用于视图的视图模型数据，所以我们需要用到了下一层`Presenter(展示器)`。
- **P: Presenter 展示器**：当我们在上一层`Interactor(交互器)`中获得原始数据后，我们需要将数据进行解析处理，比如我们在交互器中进行了网络请求，得到了json数据，若要将json中所包含的内容显示出来，我们则需要将json数据进行解析，展示器就是专注于数据的解析转换，将原始的数据转换成最终能够直接显示在试图上的视图模型数据。此外，展示器中还带有路由器`Router`，可以进行路由的操作。
- **E: Entity 实体模型对象**
- **R: Router 路由器**： 负责视图的跳转，因为使用`VIPER`架构需要进行各层之间的相互绑定，所以视图的跳转不能简单地使用原始的方法。

下面是一张 VIPER 的简单逻辑图：



图中，箭头代表着数据流的传递，我们可以看到，在 VIPER 架构中，数据的流向总是 单向流动 ，在 View 、 Interactor 、 Presenter 三层中形成了一个流动闭环，而在其他的某些架构中，如 MVC 、 MVP 、 MVVM ，它们的数据在中间层会有着双向的流动， VIPER 较它们而言，其更加约束了整个软件的架构，每一层功能特定，数据的流向单一，使得软件在开发中对原架构的高度切合。

参考:https://github.com/TangentW/TanVIPER

业务组件 业务组件 业务组件 业务组件

业务组件 业务组件 业务组件 业务组件

中间组件 中间组件 中间组件

业务公用组件 业务公用组件

业务公用组件 业务公用组件

基础组件 基础组件 基础组件

持续集成

组件化相关说明

1. 继续集成：一个主工程（壳工程），包含所有的内容（整个项目），用于发包或打包测试。
2. 基础组件： 不依赖其他任何组件，独立完成功能。主要有：*与业务无法的功能（如string或data的加密，category的封装）* 对第三方库的封装（如AFNetworking，SDWebImage的封装）
3. 业务公用组件：依赖基础组件或UIKit等系统组件，创建业务共同使用的功能（如分享，支付，网络访问）
4. 中间组件：连接业务公用组件和业务组件，及业务组件之间的互相调用。（如Mediator的组件）
5. 业务组件：单独的业务功能，不依赖其他业务组件。

谢谢观看

https://casatwy.com/iosying-yong-jia-gou-tan-kai-pian.html
https://www.jianshu.com/p/933c24506ac3