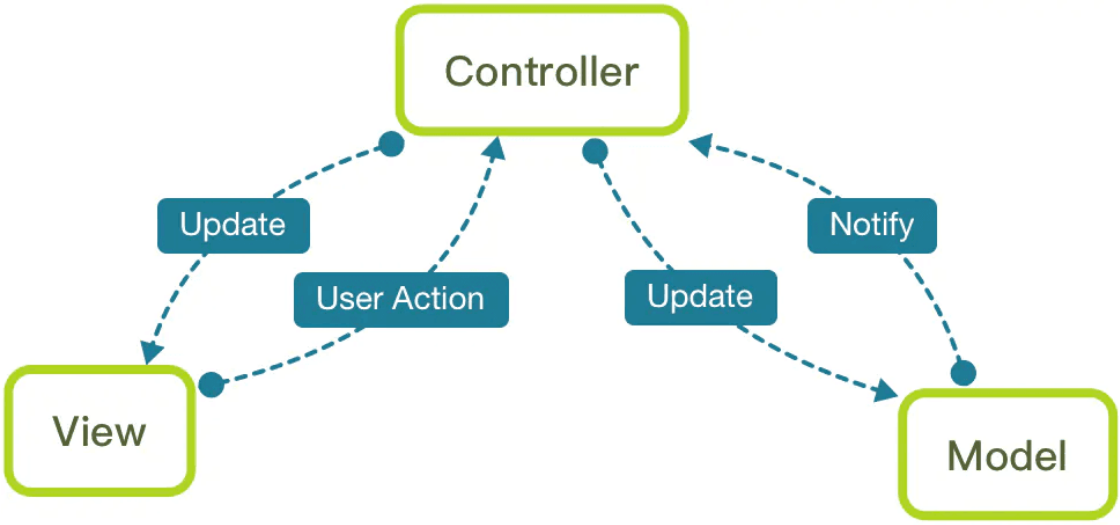
返回目录:全网各大厂 iOS 面试题-题集大全

MVVM 和 MVC 的区别

MVVM 和 MVC 的区别

1. MVC



MVC 的弊端

- 厚重的 View Controller
  - M：模型 model 的对象通常非常的简单。根据 Apple 的文档，model 应包括数据和操作数据的业务逻辑。而在实践中，model 层往往非常薄，不管怎样，model 层的业务逻辑不应被拖入到 controller。
  - V：视图 view 通常是 UIKit 控件（component，这里根据习惯译为控件）或者编码定义的 UIKit 控件的集合。View 的如何构建（PS：IB 或者手写界面）何必让 Controller 知晓，同时 View 不应该直接引用 model（PS：现实中，你懂的！），并且仅仅通过 IBAction 事件引用 controller。业务逻辑很明显不归入 view，视图本身没有任何业务。
  - C：控制器 controller。Controller 是 app 的“胶水代码”：协调模型和视图之间的所有交互。控制器负责管理他们所拥有的视图的视图层次结构，还要响应视图的 loading、appearing、disappearing 等等，同时往往也会充满我们不愿暴露的 model 的模型逻辑以及不愿暴露给视图的业务逻辑。网络数据的请求及后续处理，本地数据库操作，以及一些带有工具性质辅助方法都加大了 Massive View Controller 的产生。

- 遗失（无处安放）的网络逻辑 苹果使用的 MVC 的定义是这么说的：所有的对象都可以被归类为一个 model，一个 view，或是一个 controller。

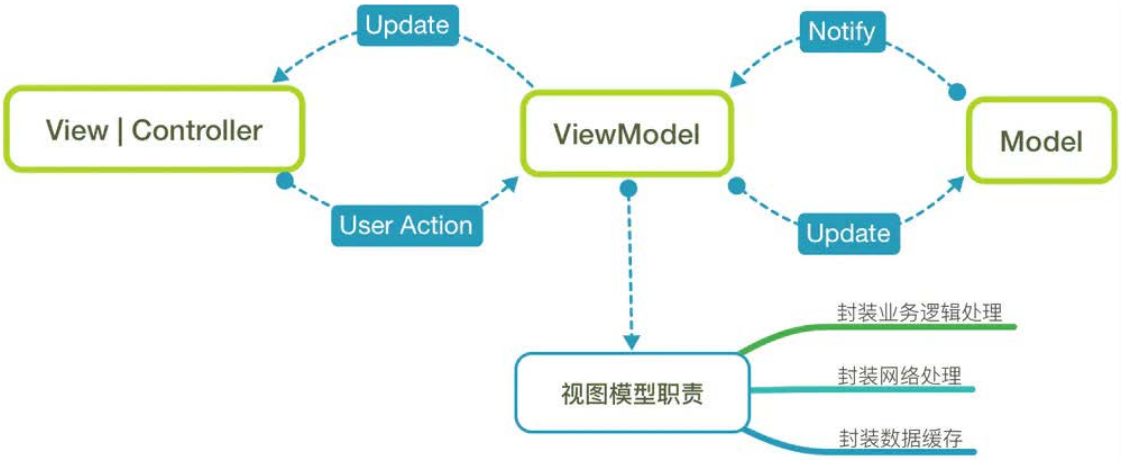
你可能试着把它放在 Model 对象里，但是也会很棘手，因为网络调用应该使用异步，这样如果一个网络请求比持有它的 model 生命周期更长，事情将变的复杂。显然 View 里面做网络请求那就更格格不入了，因此只剩下 Controller 了。若这样，这又加剧了 Massive View Controller 的问题。若不这样，何处才是网络逻辑的家呢？

- 较差的可测试性

由于 View Controller 混合了视图处理逻辑和业务逻辑，分离这些成分的单元测试成了一个艰巨的任务。

2. MVVM

一种可以很好地解决 `Massive View Controller` 问题的办法就是将 Controller 中的展示逻辑抽取出来，放置到一个专门的地方，而这个地方就是 `viewModel` 。MVVM 衍生于 MVC，是对 MVC 的一种演进，它促进了 UI 代码与业务逻辑的分离。它正式规范了视图和控制器紧耦合的性质，并引入新的组件。他们之间的结构关系如下：



2.1 MVVM 的基本概念

- 在 `MVVM` 中，`view` 和 `view controller` 正式联系在一起，我们把它们视为一个组件
- `view` 和 `view controller` 都不能直接引用 `model`，而是引用视图模型（`viewModel`）
- `viewModel` 是一个放置用户输入验证逻辑，视图显示逻辑，发起网络请求和其他代码的地方
- 使用 `MVVM` 会轻微的增加代码量，但总体上减少了代码的复杂性

2.2 MVVM 的注意事项

- view 引用 viewModel，但反过来不行（即不要在 viewModel 中引入#import UIKit.h，任何视图本身的引用都不应该放在 viewModel 中）（PS：**基本要求，必须满足**）
- viewModel 引用 model，但反过来不行* MVVM 的使用建议
- MVVM 可以兼容你当下使用的 MVC 架构。
- MVVM 增加你的应用的可测试性。
- MVVM 配合一个绑定机制效果最好（PS：ReactiveCocoa你值得拥有）。
- viewController 尽量不涉及业务逻辑，让 viewModel 去做这些事情。
- viewController 只是一个中间人，接收 view 的事件、调用 viewModel 的方法、响应 viewModel 的变化。
- viewModel 绝对不能包含视图 view（UIKit.h），不然就跟 view 产生了耦合，不方便复用和测试。
- viewModel 之间可以有依赖。
- viewModel 避免过于臃肿，否则重蹈 Controller 的覆辙，变得难以维护。

2.3 MVVM 的优势

- 低耦合：View 可以独立于 Model 变化和修改，一个 viewModel 可以绑定到不同的 View 上
- 可重用性：可以把一些视图逻辑放在一个 viewModel 里面，让很多 view 重用这段视图逻辑
- 独立开发：开发人员可以专注于业务逻辑和数据的开发 viewModel，设计人员可以专注于页面设计
- 可测试：通常界面是比较难于测试的，而 MVVM 模式可以针对 viewModel 来进行测试

2.4 MVVM 的弊端

- 数据绑定使得 Bug 很难被调试。你看到界面异常了，有可能是你 View 的代码有 Bug，也可能是 Model 的代码有问题。数据绑定使得一个位置的 Bug 被快速传递到别的位置，要定位原始出问题的地方就变得不那么容易了。
- 对于过大的项目，数据绑定和数据转化需要花费更多的内存（成本）。主要成本在于：
- 数组内容的转化成本较高：数组里面每项都要转化成 Item 对象，如果 Item 对象中还有类似数组，就很头疼。
- 转化之后的数据在大部分情况是不能直接被展示的，为了能够被展示，还需要第二次转化。
- 只有在 API 返回的数据高度标准化时，这些对象原型（Item）的可复用程度才高，否则容易出现类型爆炸，提高维护成本。
- 调试时通过对象原型查看数据内容不如直接通过 NSDictionary/NSArray 直观。

- 同一 API 的数据被不同 View 展示时，难以控制数据转化的代码，它们有可能会散落在任何需要的地方。

3. 总结

- MVC 的设计模式也并非是病入膏肓，无药可救的架构，最起码目前 MVC 设计模式仍旧是 iOS 开发的主流框架，存在即合理。针对文章所述的弊端，我们依旧有许多可行的方法去避免和解决，从而打造一个轻量级的 ViewController。

- MVVM 是 MVC 的升级版，完全兼容当前的 MVC 架构，MVVM 虽然促进了 UI 代码与业务逻辑的分离，一定程度上减轻了 ViewController 的臃肿度，但是 View 和 ViewModel 之间的数据绑定使得 MVVM 变得复杂和难用了，如果我们不能更好的驾驭两者之间的数据绑定，同样会造成 Controller 代码过于复杂，代码逻辑不易维护的问题。

- 一个轻量级的 ViewController 是基于 MVC 和 MVVM 模式进行代码职责的分离而打造的。MVC 和 MVVM 有优点也有缺点，但缺点在他们所带来的好处面前时不值一提的。他们的低耦合性，封装性，可测试性，可维护性和多人协作便利大大提高了开法效率。

- 同时，我们需要保持的是一个拥抱变化的心，以及理性分析的态度。在新技术的面前，不盲从，也不守旧，一切的决策都应该建立在认真分析的基础上，这样才能应对技术的变化。

返回目录:全网各大厂 iOS 面试题-题集大全
