结合vue、react、angular谈谈MVC、MVP、MVVM框架

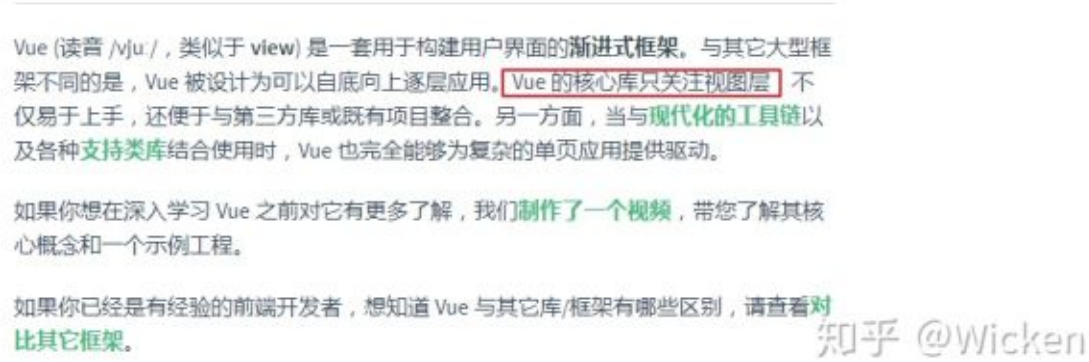
.icken
间歇性踌躇满志，持续性搬砖苦力

首先，在谈这个话题之前， 我们有必要了解一下库和框架的区别。

我们先来看react官网以及vue官网对他们的定位：

react：

vue：



react我们不说了，官网上明明白白说了，人家是一个library，用于构建用户界面。

vue的官方文档是说vue的核心库也只是关注视图（View）层。

所以，实际上来说，和angular有完整的解决方案不同，狭义的vue.js和react.js实际上只是library，还并不是一个framework，因为他们没有一整套的解决方案。

换句话来说，现在我们所讨论的vue、react，都是我们将他们武装之后，他们有了一整套解决方案了。成为了一个框架，我们再来讨论他们的架构模式。

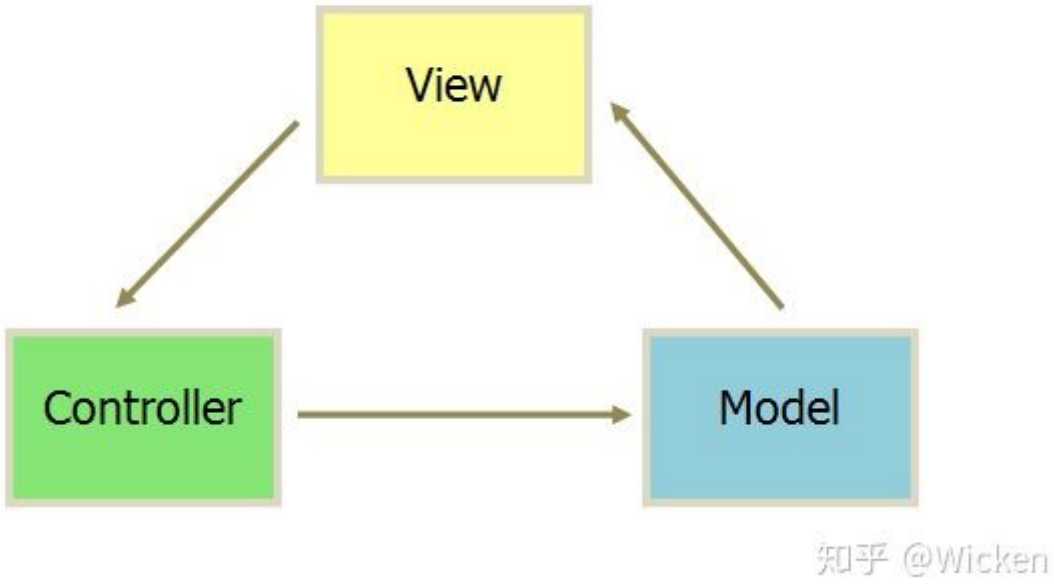
下面两层是不变的，先提前声明：

Model（数据模型）：原始数据模型的管理。

View（视图）：用户接触操作的页面。

MVC:

Controller（应用逻辑控制层）：将用户的操作反馈给Model，通知其进行数据更新，业务逻辑的中心。



我们可以先暂时抛开框架，MVC的流程大概就是，html（View）操作，告知js（Controller）要更新数据（Model）啦，js（Controller）经过请求也好啥也好，更新了数据（Model），然后再告诉html(View)找指定的UI节点去更新数据。当然这里也可以直接由js（Controller）发起对数据（Model）的更新，流程差不多也是一样的。

其实流程列出来我们就可以看到，这样的架构模式在早期的web应用中可以适应的很好。因为早期的web应用，页面的作用基本也就作为数据展示使用。Model层可以将数据处理好后通知View层渲染，就像jquery拿到ajax数据之后找到元素一顿innerHtml啥的。

但随着web的发展，业务逻辑的复杂，可以发现这种架构模式以下两个问题：

1、View更新的时候，必须要通过Controller去更新一遍Model；同样的Model更新的时候，也要去更新一遍视图。此时开发者是在同时维护View层和Model层。当页面复杂的时候，开发者不得不做许多繁琐的工作来保证数据的状态、页面的展示都是正确的。

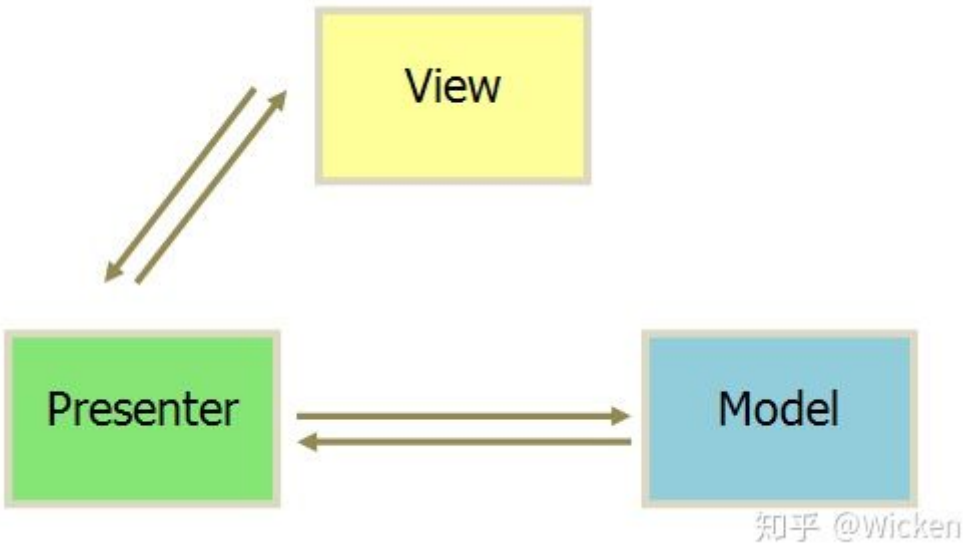
2、View层与Model层耦合，复用性差。比方说，我点击一个按钮，更新了Model并将数据渲染为List；这是我再点击一个按钮，同样更新这份数据但是渲染为Table。这个时候，由于之前逻辑已经连成一块，我们不得不再写一套渲染代码。

3、同样是由于View和Model耦合，数据流会十分混乱。比如改变了Model，这时View的更新又触发了另一个Controller，使得另一个Model又更新了，这就会使数据流像意大利面条一样缠在一起。

MVP:

诶这个时候我们想，好像这个Controller并没有什么卵用啊，就传递一下信号就完事儿了。不行，活干的这么少，让他再多干点！

如果我们能将数据返回给Controller，让Controller来控制View的渲染，那么，View和Model不就释放了吗？于是，MVP模式诞生了，操作流如下图所示：



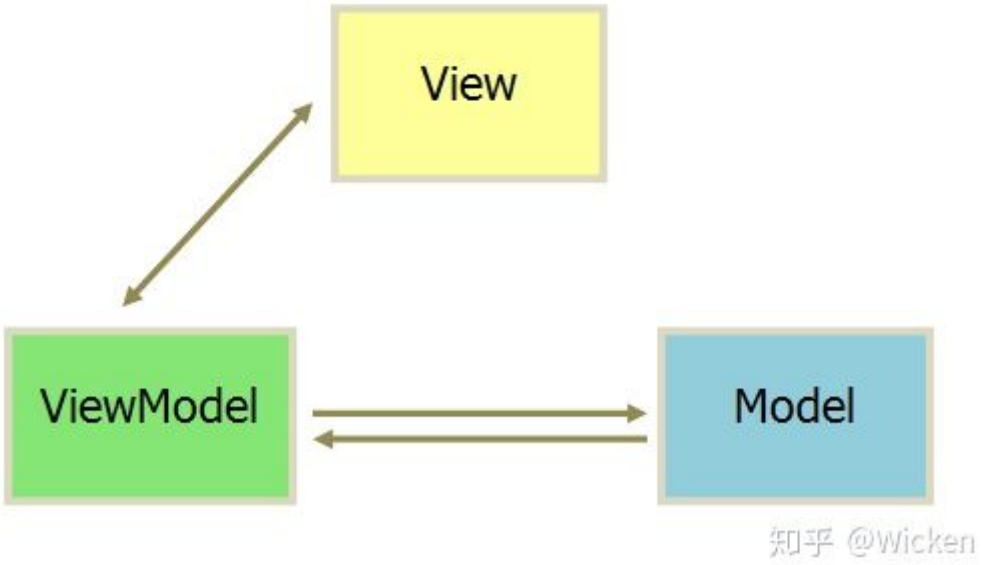
不过此时的Controller层变成了**Presenter**（中介者）层，Presenter层既能将页面操作告知Model进行数据更新，又能在数据更新时负责通知View进行更新视图，使View层与Model层解耦。

针对上述问题2，在MVP架构模式下，Model层将数据返回给Presenter，再由Presenter决定我是渲染Table呢，还是渲染List。这种架构模式下，加强了Presenter的职能，这样就解决了上述问题2、3。

但问题1依然存在啊！开发者依旧需要在Presenter中同时兼顾Dom和Data。

MVVM:

在此基础上，**如果说视图层（View）与数据层（Model）是在某个环境下是绑定的，可以实现通过数据驱动视图**，那么，上述两个问题，就都可以得到解决。于是MVVM诞生了，先看操作流：



在中间的ViewModel层中，会构建一份状态数据，视图层根据其渲染视图。这样， 开发者的精力被释放，只要聚焦在业务逻辑中。所以，我的理解是，**MVVM就是实现了数据绑定的MVP，注意，是绑定，而不是双向绑定！！！（单向数据流和双向数据流）**。

Vue

个人认为，这是毫无争议的一个MVVM框架，对MVVM理念的贯彻也是最显而易见的。

Model层：接口层，原始数据模型。

View层：视图层，template中的html代码。

ViewModel层：基于一个html元素构建的vue实例。将Model中的原始数据模型，构建成一份View层可以使用的视图模型。这个时候，Model层、View层完全解耦。开发者已经完全不需要顾及View的展示更新了，只需要专注业务逻辑以及ViewModel层与Model的交互逻辑就ok。

AngularJs

Model层：接口层，原始数据模型。

View层：html页面。

ViewModel层：基于ng-app构建的应用实例，与vue类似，数据双向绑定机制不同。

React

Model层：接口层，原始数据模型。

View层：编译之后的Dom。

ViewModel层：基于jsx语法，以及react构建的VDom，单向数据流。

这么一看，vue、react、angularJS不就都是MVVM框架吗？唯一的区别就是，VM层中的Model与View，vue与angular是数据双向绑定，而react由于是单向数据流，所以需要手动更改状态。

最后说下感受吧，其实之前一直以为自己是对这三种架构模式心里有底，但现在越看越绕，感觉归根结底就是，**在结合现有框架进行分析的时候，对model 层与 中间层（c，p或vm）的边界界定十分模糊，没有一个清晰的划分。但是，现在想来也没必要这么钻牛角尖，其实吧，每一层专注于每一层的任务，这即是核心。在此基础上的扩展以及如何对代码进行组织管理，是看需求来界定的，这也是框架架构模式不断发展的原因。**

最后的最后：由于只是用过这三个框，个人理解肯定存在局限性和不足的地方，希望各位大佬指正！！！！

参考（看了很多，感觉就这三篇算是干货比较多的）：

前端框架模式的变迁

一篇文章了解架构模式：MVC/MVP/MVVM

MVC，MVP 和 MVVM 的图示

编辑于 2019-04-30