**[Vue.js 和 MVVM 小细节](https://www.cnblogs.com/onepixel/p/6034307.html)**

MVVM 是Model-View-ViewModel 的缩写，它是一种基于前端开发的架构模式，其核心是提供对View 和 ViewModel 的双向数据绑定，这使得ViewModel 的状态改变可以自动传递给 View，即所谓的**数据双向绑定**。

Vue.js 是一个提供了 MVVM 风格的双向数据绑定的 Javascript 库，专注于View 层。它的核心是 MVVM 中的 VM，也就是 ViewModel。 ViewModel负责连接 View 和 Model，保证视图和数据的一致性，这种轻量级的架构让前端开发更加高效、便捷。

**为什么会出现 MVVM**

我接触MVVM 是在2015年，可以说2015年是MVVM 最火热的一年，而在这之前，我所知道的就是MVC,  MVC 大约是在5年前，也就是2011年的时候接触的，那时候刚学编程语言，学的Java，而Java 中经典的 SSH 框架就用来构建一个标准的MVC 框架。说实话，MVC 用了这么多年，但始终没有很深刻的理解，只停留在用的层面， 一直到接触 Vue.js 之后，研究了MVVM 架构思想，然后再回头看 MVC ，才有一种豁然开朗的感觉~

MVC 即 Model-View-Controller 的缩写，就是 **模型-视图-控制器** , 也就是说一个标准的Web 应用程序是由这三部分组成的：

**View** 用来把数据以某种方式呈现给用户。

**Model** 其实就是数据。

**Controller** 接收并处理来自用户的请求，并将 Model 返回给用户。

在HTML5 还未火起来的那些年，MVC 作为Web 应用的最佳实践是OK 的，这是因为 Web 应用的View 层相对来说比较简单，前端所需要的数据在后端基本上都可以处理好，View 层主要是做一下展示，那时候提倡的是 Controller 来处理复杂的业务逻辑，所以View 层相对来说比较轻量，就是所谓的**瘦客户端**思想。

2010年到2011年，HTML5概念被热炒，受到追捧，2012年，W3C 正式宣布HTML5规范已经正式定稿。2013年我刚进公司就接触了第一个 HTML5 框架 **Sench touch**, 它是一款用来构建移动应用的HTML5 框架，它将前后端彻底分离，前端采用MVC 架构，作为一个独立的项目工程来维护。

**前端为什么要工程化，要使用 MVC**

相对 HTML4 , HTML5 最大的亮点是**它为移动设备提供了一些非常有用的功能**，使得 HTML5 具备了开发App的能力, HTML5开发App 最大的好处就是**跨平台、快速迭代和上线，节省人力成本和提高效率**，因此很多企业开始对传统的App进行改造，逐渐用H5代替Native，到2015年的时候，市面上大多数App 或多或少嵌入都了H5 的页面。

既然要用H5 来构建 App, 那View 层所做的事，就不仅仅是简单的数据展示了，它不仅要管理复杂的数据状态，还要处理移动设备上各种操作行为等等。因此，前端也需要工程化，也需要一个类似于MVC 的框架来管理这些复杂的逻辑，使开发更加高效。 但这里的 MVC 又稍微发了点变化：

**View** UI布局，展示数据。

**Model** 管理数据。

**Controller** 响应用户操作，并将 Model 更新到 View 上。

这种 MVC 架构模式对于简单的应用来看起是OK 的，也符合软件架构的分层思想。 但实际上，随着H5 的不断发展，人们更希望使用H5 开发的应用能和Native 媲美，或者接近于原生App 的体验效果，于是前端应用的复杂程度已不同往日，今非昔比。这时前端开发就暴露出了三个痛点问题：

* 开发者在代码中大量调用相同的 DOM API, 处理繁琐 ，操作冗余，使得代码难以维护。
* 大量的DOM 操作使页面渲染性能降低，加载速度变慢，影响用户体验。
* 当 Model 频繁发生变化，开发者需要主动更新到View ；当用户的操作导致 Model 发生变化，开发者同样需要将变化的数据同步到Model 中，这样的工作不仅繁琐，而且很难维护复杂多变的数据状态。

其实，早期jquery 的出现就是为了前端能更简洁的操作DOM 而设计的，但它只解决了第一个问题，另外两个问题始终伴随着前端一直存在。

**MVVM 的出现，完美解决以上三个问题**

MVVM 由 Model,View,ViewModel 三部分构成，Model 层代表数据模型，也可以在Model中定义数据修改和操作的业务逻辑；View 代表UI 组件，它负责将数据模型转化成UI 展现出来，ViewModel 是一个同步View 和 Model的对象。

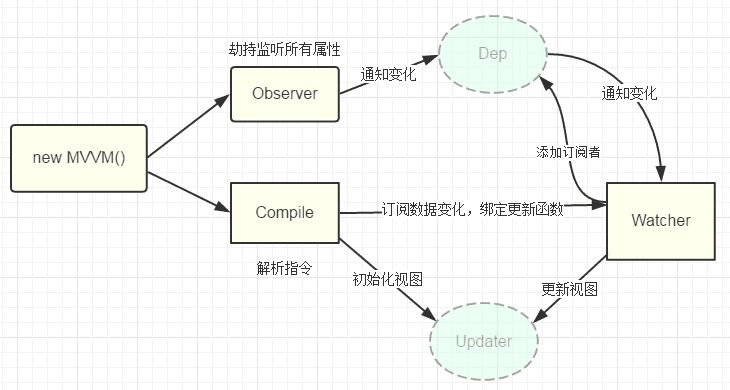
在MVVM架构下，View 和 Model 之间并没有直接的联系，而是通过ViewModel进行交互，Model 和 ViewModel 之间的交互是双向的， 因此View 数据的变化会同步到Model中，而Model 数据的变化也会立即反应到View 上。

ViewModel 通过双向数据绑定把 View 层和 Model 层连接了起来，而View 和 Model 之间的同步工作完全是自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作DOM, 不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理。

**Vue.js 细节**

Vue.js 可以说是MVVM 架构的最佳实践，专注于 MVVM 中的 ViewModel，不仅做到了数据双向绑定，而且也是一款相对比较轻量级的JS 库，API 简洁，很容易上手。Vue的基础知识网上有现成的教程，此处不再赘述， 下面简单了解一下 Vue.js 关于双向绑定的一些实现细节：

Vue.js 是采用 Object.defineProperty 的 getter 和 setter，并结合观察者模式来实现数据绑定的。当把一个普通 Javascript 对象传给 Vue 实例来作为它的 data 选项时，Vue 将遍历它的属性，用 Object.defineProperty 将它们转为 getter/setter。用户看不到 getter/setter，但是在内部它们让 Vue 追踪依赖，在属性被访问和修改时通知变化。



**Observer** 数据监听器，能够对数据对象的所有属性进行监听，如有变动可拿到最新值并通知订阅者，内部采用Object.defineProperty的getter和setter来实现。

**Compile** 指令解析器，它的作用对每个元素节点的指令进行扫描和解析，根据指令模板替换数据，以及绑定相应的更新函数。

**Watcher** 订阅者， 作为连接 Observer 和 Compile 的桥梁，能够订阅并收到每个属性变动的通知，执行指令绑定的相应回调函数。

**Dep** 消息订阅器，内部维护了一个数组，用来收集订阅者（Watcher），数据变动触发notify 函数，再调用订阅者的 update 方法。

从图中可以看出，当执行 new Vue() 时，Vue 就进入了初始化阶段，一方面Vue 会遍历 data 选项中的属性，并用 Object.defineProperty 将它们转为 getter/setter，实现数据变化监听功能；另一方面，Vue 的指令编译器Compile 对元素节点的指令进行扫描和解析，初始化视图，并订阅Watcher 来更新视图， 此时Wather 会将自己添加到消息订阅器中(Dep),初始化完毕。

当数据发生变化时，Observer 中的 setter 方法被触发，setter 会立即调用Dep.notify()，Dep 开始遍历所有的订阅者，并调用订阅者的 update 方法，订阅者收到通知后对视图进行相应的更新。

以上内容是我在大半年的项目实践之后，对MVVM 和 Vue.js 的一点点简单认识和总结，希望对各位有所帮助！

by @[一像素](http://www.cnblogs.com/onepixel/p/6034307.html)  2016.11

分类: [JS](http://www.cnblogs.com/onepixel/category/763999.html" \t "_blank),[Vue](http://www.cnblogs.com/onepixel/category/905569.html)

标签: [MVVM](http://www.cnblogs.com/onepixel/tag/MVVM/), [MVC](http://www.cnblogs.com/onepixel/tag/MVC/), [Vuejs](http://www.cnblogs.com/onepixel/tag/Vuejs/)