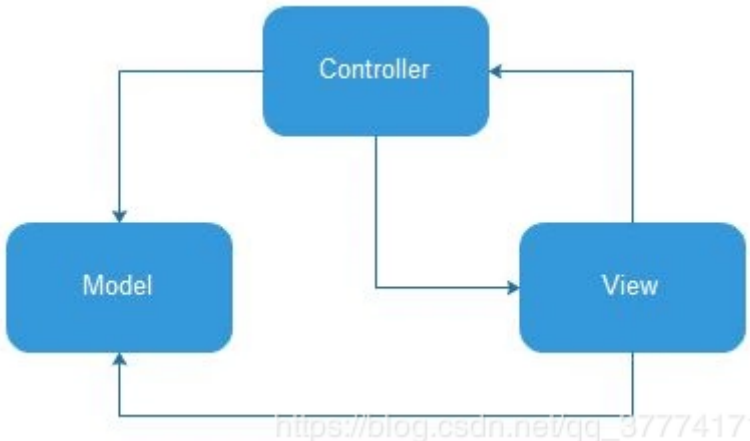
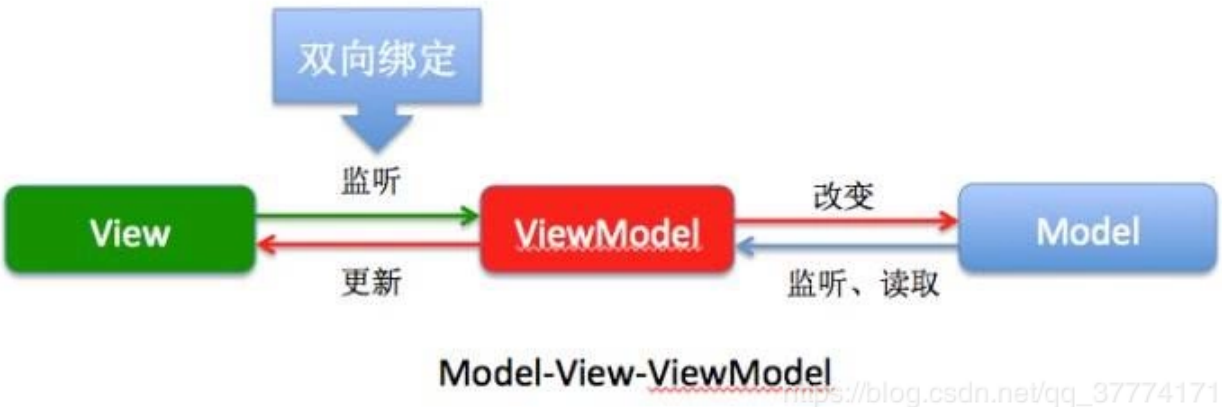
Vue的MVVM设计模式是什么？导致了和jQuery的那些区别

如果你对前端有过了解的话一定知道MVVM和MVC这两种设计模式，而且很有可能对mvp也有一些了解。

MVC即model，view，control，jQuery就是采用的这种设计模式，熟悉jQuery的同学恐怕早就对$()深通恶绝了吧。



MVVM即model，view，viewmodel，它是数据驱动模式，即所有的一切通过操作数据来进行，而尽量避免操作dom树。



Model-View-ViewModel

换句话说，我们不关注dom的结构，而是考虑数据该如何储存，用户的操作在view通过viewmodel进行数据处理，分情况是否通过ajax与model层进行交互，再返回到view层，在这个过程中view和viewmodel的数据双向绑定使得我们完全的摆脱了对dom的繁琐操作，而是专心于对用户的操作进行处理，避免了MVC中control层过厚的问题。

现在我们把一个网页应用抽象一下，那么HTML中的DOM其实就是视图，一个网页就是通过DOM的组合与嵌套，形成了最基本的视图结构，再通过CSS的修饰，在基本的视图结构上“化妆”让他们看起来更加美观。最后涉及到交互部分，就需要用到JavaScript来接受用户的交互请求，并且通过事件机制来响应用户的交互操作，并且在事件的处理函数中进行各种数据的修改，比如说修改某个DOM中的innerHTML或者innerText部分。

我们把HTML中的DOM就可以与其他的部分独立开来划分出一个层次，这个层次就叫做视图层。**Vue 的核心库只关注视图层**

我们为什么要把视图层抽取出来并且单独去关注它呢?

因为在像知乎这种页面元素非常多，结构很庞大的网页中，数据和视图如果全部混杂在一起，像传统开发一样全部混合在HTML中，那么要对它们进行处理会十分的费劲，并且如果其中有几个结构之间存在藕断丝连的关系，那么会导致代码上出现更大的问题，这什么问题呢?
你是否还记得你当初写JQuery的时候，有写过 $ ('#xxx').parent().parent().parent()这种代码呢？当你第一次写的时候，你觉得页面元素不多，不就是找这个元素的爸爸的爸爸的爸爸吗，我大不了在注释里面写清楚这个元素的爸爸的爸爸的爸爸不就好了。但是万一过几天之后你的项目组长或者你的产品经理突然对你做的网页提出修改要求，这个修改要求将会影响页面的结构，也就是DOM的关联与嵌套层次要发生改变，那么$ ('#xxx').parent().parent().parent()可能就会变成 $('#xxx').parent().parent().parent().parent().parent()了。
这还不算什么，等以后产品迭代越来越快，修改越来越多，而且页面中类似的关联和嵌套DOM元素不止一个，那么修改起来将非常费劲。而且JQuery选择器查找页面元素以及DOM操作本身也是有性能损失的，可能到时候打开这个页面，会变得越来越卡，而你却无从下手。
当你在编写项目的时候遇到了这种问题，你一定会抱怨，为什么世上会有HTML这种像盗梦空间一样的需要无数div嵌套才能做出页面的语言，为什么当初学JQuery看中的是它简洁的DOM操作，现在却一点也不觉得它有多简洁，难道我学的是假的JQuery？为什么写个代码这么难，你想砸电脑，你想一键盘拍在产品狗的脑袋上，责怪他天天改需求才让你原本花清香茶清味的代码变得如此又臭又长。
这个时候如果你学过Vue.js，那么这些抱怨将不复存在。

3.Vue.js有那些优点?

- 声明式，响应式的数据绑定
- 组件化的开发
- Virtual DOM

响应式的数据绑定

1.jQuery首先要获取到dom对象，然后对dom对象进行进行值的修改等操作

2.Vue是首先把值和js对象进行绑定，然后修改js对象的值，Vue框架就会自动把dom的值就行更新。

3.可以简单的理解为Vue帮我们做了dom操作，我们以后用Vue就需要修改对象的值和做好元素和对象的绑定，Vue这个框架就会自动帮我们做好dom的相关操作

4.这种dom元素跟随JS对象值的变化而变化叫做单向数据绑定，如果JS对象的值也跟随着dom元素的
值的变化而变化就叫做双向数据绑定

<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <title>vue.js测试 - 代码之美专栏</title> <!-- author:昌维 代码之美 https://zhuanlan.zhihu.com/codes -->

<script src="https://unpkg.com/vue/dist/vue.js"></script> </head> <body> <input type="text" name="" value="" placeholder="在这里输入文字，下面会跟着变化" v-model="message"> <hr> {{ message }} <script type="text/javascript"> var app = new Vue({ el: '#app', data: { message: 'Hello Vue!' } }) </script> </body> </html>
是不是会发现一个很神奇的现象，文本框里面输入的文字和后面的p标签中的内容一起变化?
换句话说，p标签里面通过{{ message }}这个写法与input标签中的value绑定在了一起，其中变化，另外一个和它绑定的数据就跟着变化。
结合标题来说，就是vue.js会自动响应数据的变化情况，并且根据用户在代码中预先写好的绑定关系，对所有绑定在一起的数据和视图内容都进行修改。而这种绑定关系，在图上是以input 标签的v-model属性来声明的，因此你在别的地方可能也会看到有人粗略的称vue.js为声明式渲染的模版引擎。

组件化开发

作为整个VUE文档中篇幅最大的部分，组件可是相当的添彩，要不是有组件这么易于复用，不易污染的特性，怕不是我都疯了无数回。
打个比方，我们现在要做一个有一百个页面的项目，其中有三十三个导航栏是A，六十七个导航栏是B，这其中三十三个A导航栏中有一个模块与众不同，可以分为A1,A2,A3,A4……
这个如果用jQuery解决的话，就得自己封装模板插件，且要么写(A,B,A1,A2，An).length个模板，要么模板套模板。
啧啧，累死个狗娘养的了。
这点上，VUE的模板就简单的多，我们先算好要多少个组件，然后看看组件之间有没有相互嵌套，把所有需要的地方都先挖上坑（写好组件标签），并且在组件标签中写好要传入组件的参数，再分别写好各种组件的实现，简简单单的就写好了，即使是嵌套也只是组件标签中套一个组件标签，更简单的改一个传参能够实现。
还记得在传统前端开发的时候，我们都是每个人做一个页面，然后最后套入各种后端模版引擎，比如说PHP的Smarty或者Java的JSP等等。
但是现在我们做单页应用，页面交互和结构十分复杂，一个页面上就有许许多多的模块需要编写，而且往往一个模块的代码量和工作量就非常庞大，如果还按照原先的方法来开发，那么会累死人。而且遇到以后的产品需求变更，修改起来也非常麻烦，生怕动了其中一个div之后，其他div跟着雪崩，整个页面全部乱套，或者由于JavaScript的事件冒泡机制，导致修改一些内层的DOM事件处理函数之后，出现各种莫名其妙的诡异BUG。
在面向对象编程中，我们可以使用面向对象的思想将各种模块打包成类或者把一个大的业务模块拆分成更多更小的几个类。在面向过程编程中，我们也可以把一些大功能拆分成许多函数，然后分配给不同的人来开发。
在前端应用，我们是否也可以像编程一样把模块封装呢？这就引入了组件化开发的思想。
Vue.js通过组件，把一个单页应用中的各种模块拆分到一个一个单独的组件（component）中，我们只要先在父级应用中写好各种组件标签（占坑），并且在组件标签中写好要传入组件的参数（就像给函数传入参数一样，这个参数叫做组件的属性），然后再分别写好各种组件的实现（填坑），然后整个应用就算做完了。

Virtual DOM

现在的网速越来越快了，很多人家里都是几十甚至上百M的光纤，手机也是4G起步了，按道理一个网页才几百K，而且浏览器本身还会缓存很多资源文件，那么几十M的光纤为什么打开一个之前已经打开过，已经有缓存的页面还是感觉很慢呢？这就是因为浏览器本身处理DOM也是有性能瓶颈的，尤其是在传统开发中，用JQuery或者原生的JavaScript DOM操作函数对DOM进行频繁操作的时候，浏览器要不停的渲染新的DOM树，导致页面看起来非常卡顿。

而Virtual DOM则是虚拟DOM的英文，简单来说，他就是一种可以预先通过JavaScript进行各种计算，把最终的DOM操作计算出来并优化，由于这个DOM操作属于预处理操作，并没有真实的操作DOM，所以叫做虚拟DOM。最后在计算完毕才真正将DOM操作提交，将DOM操作变化反映到DOM树上。

对于vue.js的Virtual DOM，目前业界有着褒贬不一的评价。有人认为Vue.js作为一个轻量级框架，引入Virtual DOM会加大Vue.js本身的代码尺寸，也会消耗更多CPU（手机上会更耗电）（注意：消耗更多的CPU并不意味着会更卡，因为JavaScript计算是后台计算，他的计算量还不至于让DOM操作变得卡顿），并且在操作单个DOM元素的时候，反而多了一道计算工序，会更慢。但也有人认为基本上会用Vue.js开发的都是页面中内容很多的元素，肯定操作的DOM量级普遍较大，平均一下还是比较划算的。