目录

历史由来

尤雨溪谈Vue.js：“我在 Google 的工作需要在浏览器上进行大量原型设计，于是我想要尽快获得有形的东西。当时有些项目使用了 Angular。Angular 提供了一些用数据绑定和数据驱动来处理 DOM 的方法，所以你不必自己碰 DOM。它也有一些副作用，就是按照它规定的方式来构建代码。对于当时的场景而言实在是太重了。

我想，我可以只把我喜欢的部分从 Angular 中提出来，建立一个非常轻巧的库，不需要那些额外的逻辑。我也很好奇 Angular 的源码到底是怎么设计的。我最开始只是想着手提取 Angular 里面很小的功能，如声明式数据绑定。Vue 大概就是这么开始的。

用过一段时间之后，我感觉我做的东西还有点前途，因为我自己就很喜欢用。于是我花了更多的时间把它封装好，取了一个名字叫做 Vue.js。

2014 年 2 月，我第一次将它作为实际的项目发布在 Github 上，并把链接发送到了 Hacker News 上，它就被顶到了首页，然后它在首页待了好几个小时。后来，我写了一篇文章，分享了 Vue 第一周的使用数据以及我的感受。

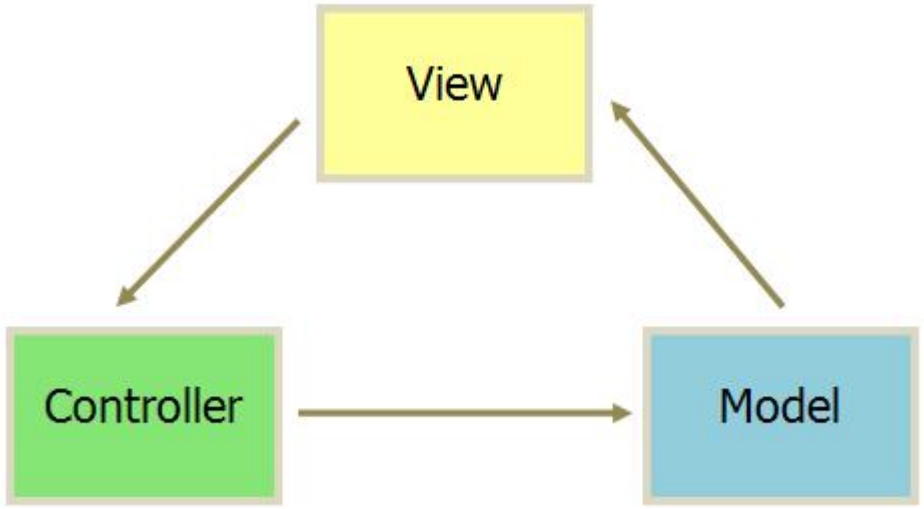
那是我第一次看见这么多人在 Github 上为一个项目打星星。我当时一个星期收获了好几百个星星，整个人都激动坏了。

原文链接：

https://mp.weixin.qq.com/s?__biz=MzA4NjE3MDg4OQ%3D%3D&mid=2650964658&idx=1&sn=20bffec66f8b45002addb417a51ea92d&chksm=843aeed4b34d67c216f167bef8a1fdf85cc9bc50590076668699 09d31ab86a8bd9d005614e4e&mpshare=1&scene=1&srcid=06137IzUoM84mpRfbUu7tDb0

MVVM模式

MVC模式的意思是，软件可以分成三个部分。各部分之间的通信方式如下。

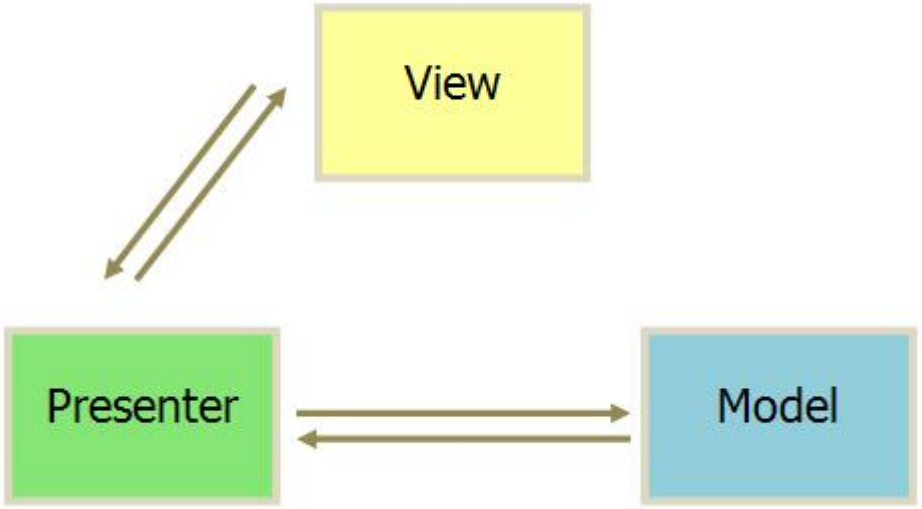


1. View 传送指令到 Controller
2. Controller 完成业务逻辑后，要求 Model 改变状态
3. Model 将新的数据发送到 View，用户得到反馈

所有通信都是单向的。

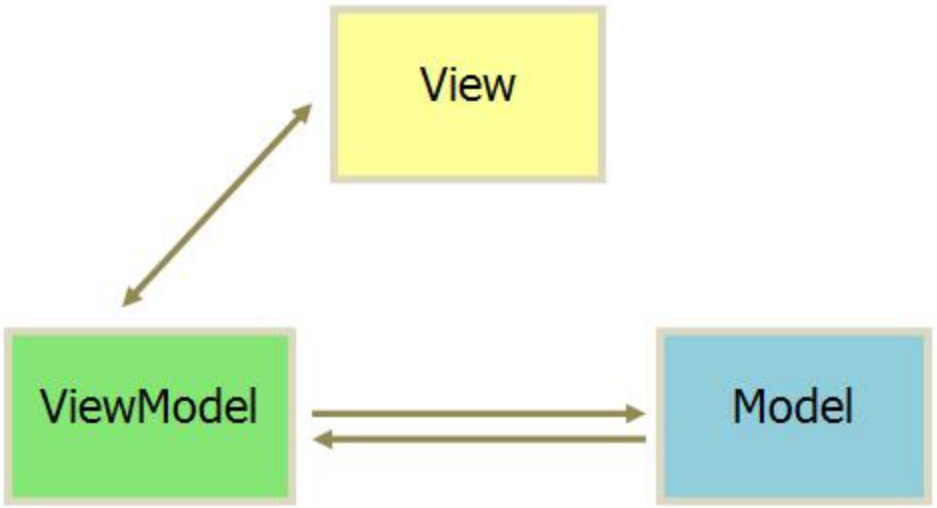
MVVM模式

MVP 模式将 Controller 改名为 Presenter，同时改变了通信方向。



1. 各部分之间的通信，都是双向的。
2. View 与 Model 不发生联系，都通过 Presenter 传递。
3. View 非常薄，不部署任何业务逻辑，称为"被动视图"（Passive View），即没有任何主动性，而 Presenter非常厚，所有逻辑都部署在那里。

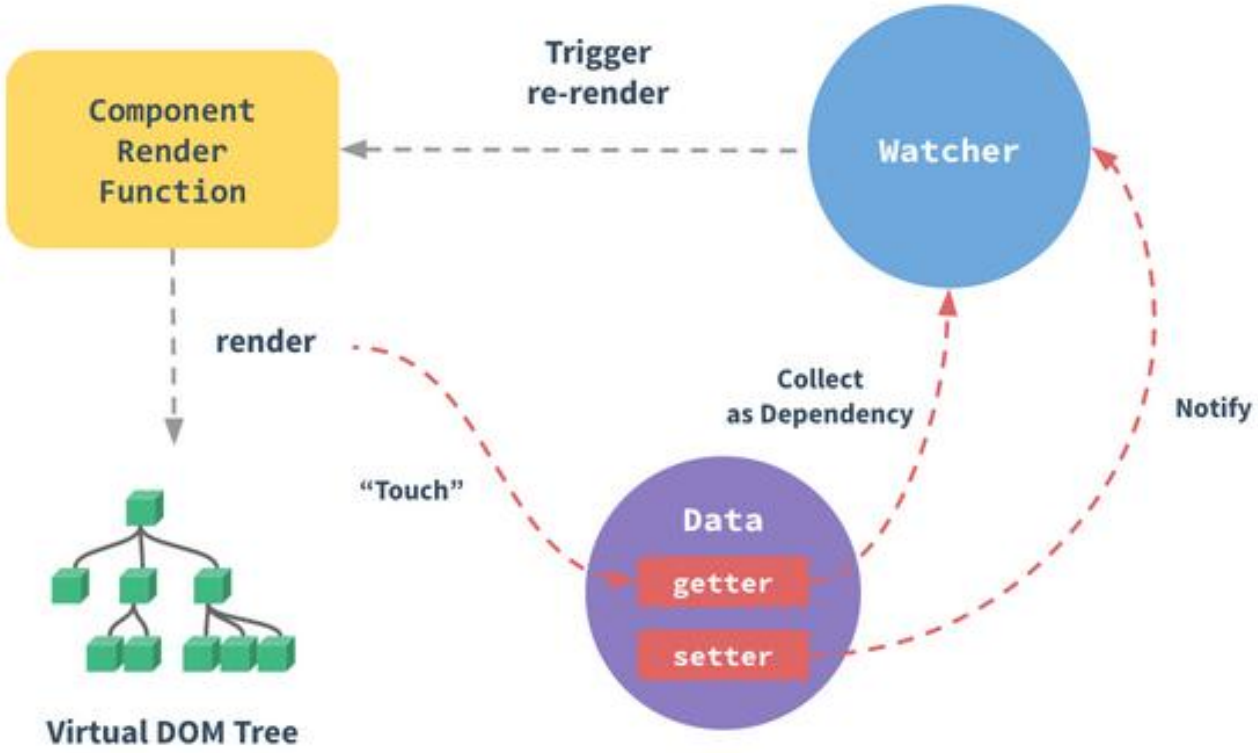
MVVM模式



MVVM模式采用双向绑定（data-binding）：View的变动，自动反映在ViewModel，反之亦然。Vue、Angular 和 Ember 都采用这种模式，相比于Angular，Vue.js提供了更加简洁、更易于理解的API，使得我们能够快速地上手并使用Vue.js。

数据驱动和组件式编程

数据驱动：



程序=数据结构+算法，这是每个程序都耳熟能详的一句话,可在前端这里并不纯粹,因为前端需要跟界面打交道,html+css并没用被抽象成某种在js中使用的数据结构,充当的更多是界面的一种配置,jquery程序员看待他的方式就一块块的ui,用到的时候再$一下,获取之后修改.整个程序写下来是零零散散的节点操作。一个比较实际的情况就是,在ui控件有联动的时候,如果没有一种机制来管理这些ui之间的修改,那么依赖程序员自己去手动管理这些ui的状态,会让人烦不胜烦,且容易出现bug。

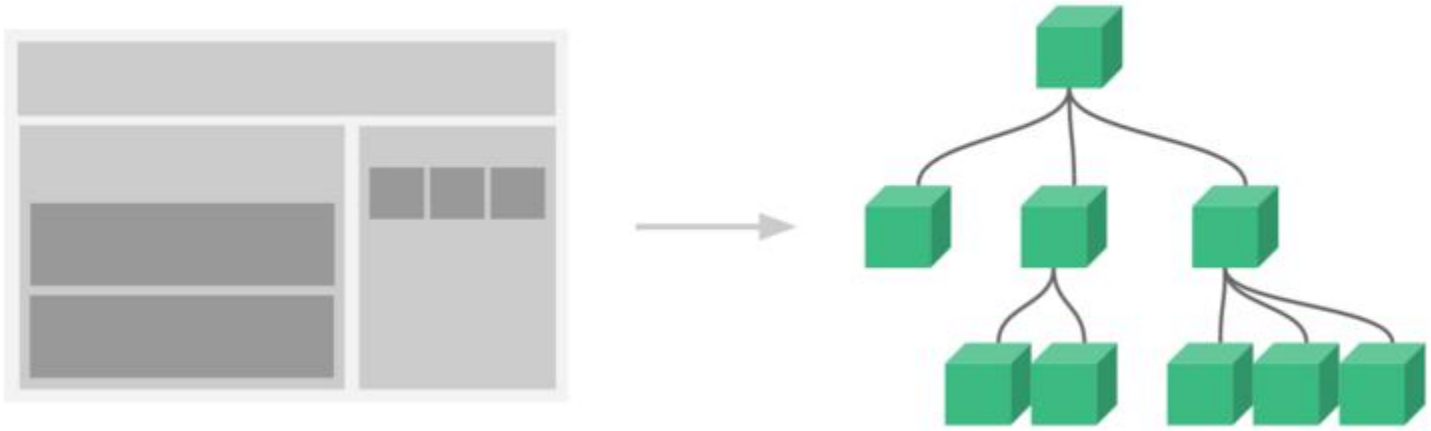
总结一下基于操作dom的前端开发方式：

拼界面->找到dom节点->修改属性->检测是否有其他影响的节点->根据刚刚修改的dom节点更新自己的状态

那么上面的那句话就变成了 ： 前端程序 = 拼界面+操作ui+算法

vue或者angular这些mvvm框架给了前端另一种思路,完全基于数据驱动的编程。如果你之前已经习惯了用jQuery操作DOM，学习Vue.js时请先抛开手动操作DOM的思维，因为Vue.js是数据驱动的，你无需手动操作DOM。Vue采用一种数据绑定的方式,自动绑定dom节点的属性.这样就把你从操作dom节点的繁琐过程中解脱出来了,你只要专注于数据的状态,ui更新的事情你不需要去管了,不管是样式还是内容,可见性还是切换class,框架帮你把关注点从传统的dom操作转移到了数据,回归编程的本质:程序=数据结构+算法. 这也是mvvm框架最大的思路上的突破。

组件式编程



这个理念不是来源于vue, 把web组件式开发发扬光大的应该是react了,组件开发是一种朴素的开发思想,分而治之,大型系统拆分成一个个的小模块小组件,分配给不同的人。额外的好处是顺便能复用这个组件。

理解组件的思想可以类比函数。一个函数包含哪些东西呢？

1.形参

2.局部变量

3.函数名

4.返回值

那对应到vue中又是什么呢?

函数	vue组件
形参	slot,props
局部变量和局部函数	data,methods
函数名	name
return	template

Vue项目创建

方式：

1、可以直接<script>引入vue.js创建vue项目

2、使用vue-cli脚手架创建新项目

这边选择用第二种方式创建vue项目(减少配置时间)

步骤：

1、安装node环境

下载地址为：https://nodejs.org/en/

检查是否安装成功：cmd命令窗口，输入命令node-v,输出版本号说明安装成功

为了提高效率，可以使用淘宝的镜像：http://npm.taobao.org/

输入npm install -g cnpm –registry=https://registry.npm.taobao.org，即可安装npm镜像，以后再用到npm的地方直接用cnpm来代替就好了

检查是否安装成功cnpm -v

2、搭建vue项目环境

1、全局安装vue-cli

npm install -g vue-cli

Vue项目创建

2、进入你的项目目录，创建一个基于 webpack 模板的新项目: vue init webpack 项目名

这样一个vue项目就建好了，下面直接命令行进入该项目目录下

cnpm install //安装依赖

cnpm run dev //项目运行并预览项目效果

项目创建完成可以根据自己的需求安装组件库(推荐)

pc端

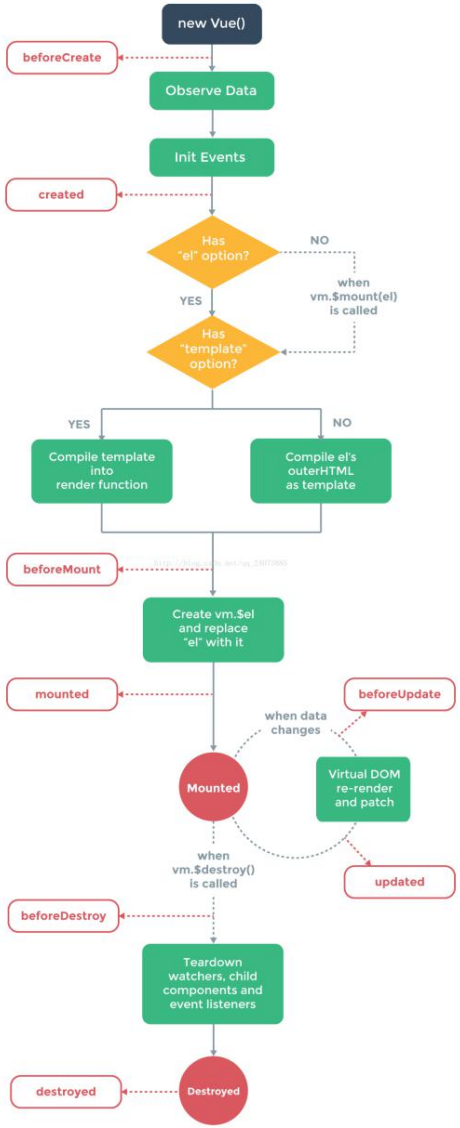
elementui 网址：http://element-cn.eleme.io/#/zh-CN

iView 网址：https://www.iviewui.com/

vue-element-admin 网址：https://panjiachen.github.io/vue-element-admin-site/zh

wap端：vux 、Vant（有赞团队）、Mint UI（饿了么团队）

生命周期


new Vue()
beforeCreate
Observe Data
Init Events
created
Has "el" option?
NO
when vm.$mount(el) is called
YES
Has "template" option?
YES
NO
Compile template into render function
Compile el's outerHTML as template
beforeMount
Create vm.$el and replace "el" with it
mounted
beforeUpdate
when data changes
Mounted
Virtual DOM re-render and patch
when vm.$destroy() is called
updated
beforeDestroy
Teardown watchers, child components and event listeners
destroyed
Destroyed

生命周期

每个 Vue 实例在被创建之前都要经过一系列的初始化过程。例如，实例需要配置数据观测(data observer)、编译模版、挂载实例到 DOM ，然后在数据变化时更新 DOM 。在这个过程中，实例也会调用一些 **生命周期钩子** ，这就给我们提供了执行自定义逻辑的机会。

它可以总共分为8个阶段：

1. beforeCreate:在实例初始化之后，数据观测(data observer) 和 event/watcher 事件配置之前被调用。
2. created:实例已经创建完成之后被调用。在这一步，实例已完成以下的配置：数据观测(data observer)，属性和方法的运算， watch/event 事件回调。然而，挂载阶段还没开始，$el 属性目前不可见。
3. beforeMount:在挂载开始之前被调用：相关的 render 函数首次被调用。**该钩子在服务器端渲染期间不被调用。**
4. mounted: el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.$el 也在文档内。**该钩子在服务器端渲染期间不被调用。**

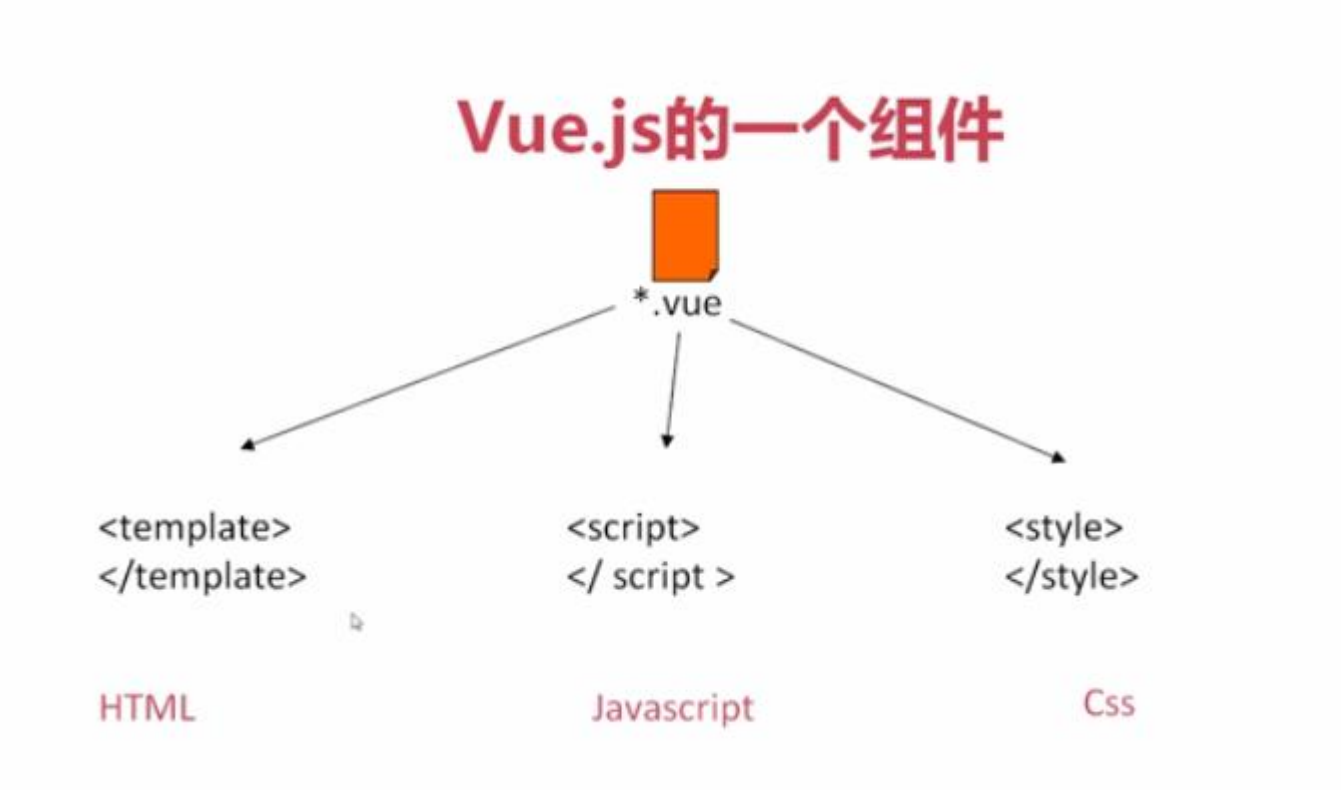
5.beforeUpdate:数据更新时调用，发生在虚拟 DOM 重新渲染和打补丁之前。你可以在这个钩子中进一步地更改状态，这不会触发附加的重渲染过程。**该钩子在服务器端渲染期间不被调用。**

6.updated:由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。当这个钩子被调用时，组件 DOM 已经更新，所以你现在可以执行依赖于 DOM 的操作。然而在大多数情况下，你应该避免在此期间更改状态。如果要相应状态改变，通常最好使用计算属性或 watcher 取而代之。**该钩子在服务器端渲染期间不被调用。**

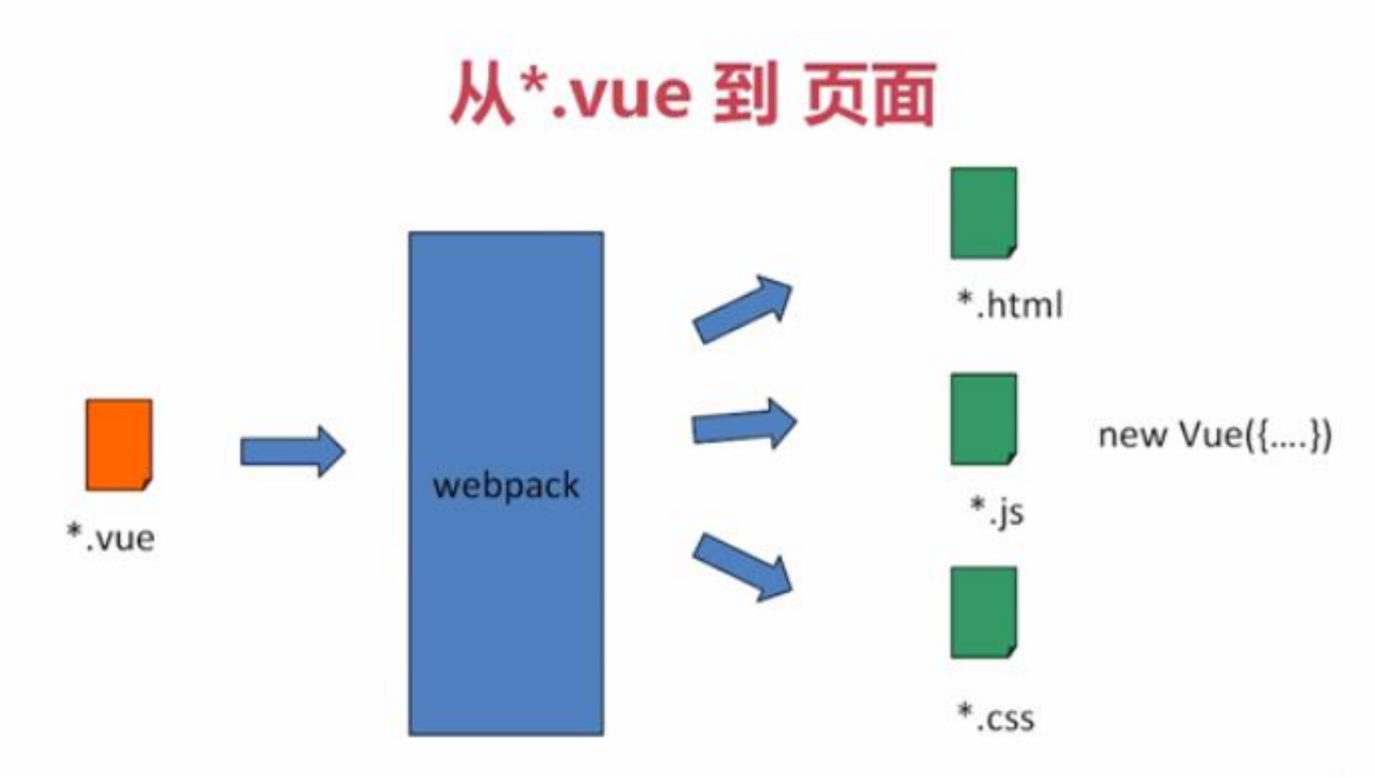
7.beforeDestroy:实例销毁之前调用。在这一步，实例仍然完全可用。**该钩子在服务器端渲染期间不被调用。**

8.destroyed:Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。**该钩子在服务器端渲染期间不被调用。**

从Vue到页面


Vue.js的一个组件
*.vue
<template>
</template>
<script>
</ script >
<style>
</style>
HTML
Javascript
Css

从Vue到页面



Vue组件的重要选项

data:

Vue 实例的数据对象。Vue 将会递归将 data 的属性转换为 getter/setter，从而让 data 的属性能够响应数据变化。**对象必须是纯粹的对象(含有零个或多个的key/value对)**：浏览器 API 创建的原生对象，原型上的属性会被忽略。大概来说，data 应该只能是数据 - 不推荐观察拥有状态行为的对象。

一旦观察过，不需要再次在数据对象上添加响应式属性。因此推荐在创建实例之前，就声明所有的根级响应式属性。

实例创建之后，可以通过 `vm.$data` 访问原始数据对象。Vue 实例也代理了 data 对象上所有的属性，因此访问 `vm.a` 等价于访问 `vm.$data.a`。

以 `_` 或 `$` 开头的属性 **不会** 被 Vue 实例代理，因为它们可能和 Vue 内置的属性、 API 方法冲突。你可以使用例如 `vm.$data._property` 的方式访问这些属性。

当一个**组件**被定义， `data` 必须声明为返回一个初始数据对象的函数，因为组件可能被用来创建多个实例。如果 `data` 仍然是一个纯粹的对象，则所有的实例将**共享引用**同一个数据对象！通过提供 `data` 函数，每次创建一个新实例后，我们能够调用 `data` 函数，从而返回初始数据的一个全新副本数据对象。

如果需要，可以通过将 `vm.$data` 传入 `JSON.parse(JSON.stringify(...))` 得到深拷贝的原始数据对象。

Vue组件的重要选项

props:

props 可以是数组或对象，用于接收来自父组件的数据。props 可以是简单的数组，或者使用对象作为替代，对象允许配置高级选项，如类型检测、自定义校验和设置默认值。

methods:

methods 将被混入到 Vue 实例中。可以直接通过 VM 实例访问这些方法，或者在指令表达式中使用。方法中的 `this` 自动绑定为 Vue 实例。

watch:

一个对象，键是需要观察的表达式，值是对应回调函数。值也可以是方法名，或者包含选项的对象。Vue 实例将会在实例化时调用 `$watch()` ，遍历 watch 对象的每一个属性。

computed:

计算属性将被混入到 Vue 实例中。所有 getter 和 setter 的 this 上下文自动地绑定为 Vue 实例。

Vue常用指令

v-bind:绑定变量

v-text:绑定标签内的内容（指定了标签内的内容，相当于innerText）

v-html:以html形式渲染变量内容

v-show：隐藏节点的显示（即display:none）,虽不渲染，但是dom对象一直存在，适用于频繁切换的场景

v-if:隐藏节点显示相当于appendChild，removeChild，直接将dom对象添加或者删除，适用于不频繁切换的场景，与v-if配合使用的有，v-else，v-if-else

v-for:数组遍历，循环产生同一个组件

v-on:节点绑定事件，可简写成@

v-model:为input提供双向绑定功能

v-pre: 跳过这个元素和它的子元素的编译过程。一些静态的内容不需要编辑加这个指令可以加快编辑

<span v-pre>{{ this will not be compiled }}　显示的是{{ this will not be compiled }}

v-bind和v-on是vuejs html模板中经常使用的两个指令。 所以他们提供了他们两人的缩写符号如下：

v-on:click='someFunction'　@click='someFunction'　(两者相同，后者是前者的速记)

v-bind:href='var1'　:href='var1'

注意事项

1、为了使样式私有化（模块化），不对全局造成污染，可以在style标签上添加scoped属性以表示它的只属于当下的模块，但是修改公共组件（三房库或者项目定制的组件）样式不可以放在scoped,scoped往往会造成更多的困难。

scoped三条渲染规则

1、给HTML的DOM节点加一个不重复data属性(形如：data-v-2311c06a)来表示他的唯一性。

2、在每句css选择器的末尾（编译后的生成的css语句）加一个当前组件的data属性选择器（如[data-v-2311c06a]）来私有化样式。

3、如果组件内部包含有其他组件，只会给其他组件的最外层标签加上当前组件的data属性。

2、路由跳转方式

1、<router-link to='需要跳转到的页面的路径>

浏览器在解析时，将它解析成一个类似于<a> 的标签。

2、that.$router.push({ path:'/home',query:{userName:'张三'}})

页面接收参数 this.$route.query.userName

注意事项

```
that.$router.push({ name:'home', params: {userName:'张三'}})
```

页面接收参数 this.$route.params.userName

采用这用方式路由需要路由文件把参数拼到url里面（不然页面刷新参数丢失）

如：

```
{
    path: '/home:userName?',
    name: 'home',
    component: () =>import('@/views/home')
}
```

3、 this.$router.replace() //同push一样，导航后不会留下 history 记录。即使点击返回按钮也不会回到这个页面

3、this.$router.go(n)

向前或者向后跳转n个页面，n可为正整数或负整数