



书栈(BookStack.CN)

目 录

致谢

本书缘起

前言

为什么要学习Vuejs?

什么是单页应用

与React, Angular的对比

被微信和淘宝所看重

使用Vuejs的知名项目和公司

原生的Vuejs

Vuejs 初体验

Webpack + Vuejs实战

安装前准备: NVM, NPM 与Node

安装前准备: Git Bash

Webpack介绍

安装webpack与vuejs

项目文件夹基本结构

定义一个页面

语法简写说明

页面渲染过程

视图中的渲染

模板指令directive

页面间的参数传递

路由router

使用样式

双向绑定

表单的绑定

表单的提交

component 初级知识

部署和发布

打包和部署

在nginx上配置，解决跨域问题

如何debug

基本命令(vue, vue-cli)

进阶知识

js的作用域和this

mixin

computed properties

component 进阶知识

slot

vuex状态管理器

生命周期

最佳实践

事件Event

与CSS预处理器结合使用

自定义Directive

实战周边

微信支付

用户的注册和微信授权

与App的结合使用

发送http请求

在vue中操作什么属性，在接口中就要存在该属性的key.

实战项目

准备1: 文字需求

准备2: 需求原型图

准备3: 微信的多种账号

准备4: 其他

整体项目的搭建

用户的注册和微信授权

登陆状态的保持

首页-轮播图

底部Tab

商品列表页

商品详情页

购物车

微信支付

回顾

相关工具

vue-devtools

如何学习文档

查看Vuejs API

致谢

当前文档 《Vue.js 中文教程》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2019-06-17。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN)，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

内容来源：申思维 http://vue_book.siwei.me/

文档地址：http://www.bookstack.cn/books/happy_book_vuejs

书栈官网：<http://www.bookstack.cn>

书栈开源：<https://github.com/TruthHun>

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

本书缘起

这个教程，是根据我公司过去两年多的实际项目经验写的。

我从2016年经营自己的软件公司。 到2018年6月，我们做了近三十个项目。 这些项目中，对于手机端的web的呼声很高。大部分都要求在手机端使用web打开。

在使用Vuejs之前， 我考察过Angular (包括1.x, 2.x 版本)， React, Meteor，这几个框架都不行。

要么是学习曲线陡峭，概念复杂，把简单的事情复杂化 (例如Angular) ,

要么就是编码风格不好，前后端代码混写一起，(例如React, Meteor)

而Vuejs是当时在stackoverflow等国外技术站点上被一致看好的技术。

第一次使用是在2016年4月。 Vuejs 1.x的时候。 我们发现入门特别快，稍微有一定web开发经验的程序员，在一周内就可以上手做项目，认真学习的话，一个月就可以达到高手水平 (快速的开发项目)， 两三个月左右就可以达到高级水平 (熟练使用Vuex, 自己写component等)

这么快的上手速度，在其他语言中是不可想象的。 根据我的实际体会， 使用Angular 入门需要一个月。 使用React入门速度也没有这么快。

总之，越是简洁的框架，就越好学。

后来我们在项目中使用它，一发不可收拾。 只要是个h5项目，就可以很好的用起来。 快速开发，快速迭代，性能杠杠的。

而且，额外的好处，是可以非常好的跟Native App的开发结合。在IOS上可以做到完美呈现， 让人无法分辨哪个页面是原生，哪个页面是H5做的。

学习目标

本教程是我公司的员工培训教程，可以在极短的时间内(例如一周)让人上手Vuejs项目。 让人：

1. 看得懂代码
2. 可以写一些基本的功能
3. 可以调试，部署

这就算入门H5开发了。

使用说明

如果世界上文档分成两类：

- guide, 教程型文档。
- api, 接口型文档。

本文档就是入门的教程型文档。 在线查看地址: http://vue_book.siwei.me/

本教程的代码，都来源于这个demo: https://github.com/sg552/vue_js_lesson_demo

以及： code_example 目录。

本书使用gitbook写就，可以自行编译，（安装环境和编译命令，参考：<http://siwei.me/blog/posts/gitbook-gitbook>）

本书中的出现的命令行，都统一以 `$` 作为开始， 例如：

```
1. $ npm install vue-cli
```

各位对命令行不熟悉的同学，记得在敲命令的时候，跳过最前面的 `$` 即可。

版本说明

截止到2016年6月底， Vuejs的版本是 `2.5.16`。本书中的大部分例子都是在该版本下演示的。

如果您是一名没有任何工作经验的新人，在windows 环境下，建议使用 sublime(免费) + git bash (免费)，就可以运行本书中的所有例子了。

如果您是一名有工作经验的老鸟，那么Linux， Mac则是非常好的选择~

那么，我们开始一段的令人兴奋的学习历程吧！

前言

随着移动电话的普及，和微信的流行，很多的wap(h5)应用也随时流行了。例如：微店，微站。以及各个app中包含的h5页面。

手机的硬件特点是：

- 硬件设备差。同主频的手机CPU性能往往是台式机的10分之一。（手机的供电跟台式机设备差的很远）
- 网络速度慢。4G 网络很多时候下载速度只有几百K. 可能一个微信中的网页打开要很久。

所以，使用传统的web技术开发的网页，在手机端的表现往往特别差，因为传统技术的特点是：

- 点击某个链接/按钮，或者提交表单后，web页面整体刷新。
- js/css 的请求往往很多，过百是很常见的事儿。

每次页面整体刷新，都要导致浏览器重新加载对应的内容。特别卡顿。

另外，加载的内容也很多。很多传统页面的css/js 都多达上百个，每次打开页面，需要发送上百次请求。如果页面中包含了websocket 等内容，打开速度就更慢了。

苹果的机器表现还好，ios的设备打开Web页面速度很快，android机则大部分都很慢。这个是由于手机设备操作系统、软件以及智能硬件决定的。

所以，单页应用（Single page app，简称 SPA）则体现出了巨大的优势：

- 页面是局部刷新的，响应速度快，不需要每次都加载所有的js/css，
- 前后端分离，前端(手机端) 不受(服务器端的) 开发语言限制。

所以越来越多的app采用了 SPA的架构。

如果你的项目要用在h5上，那么一定使用单页应用框架。

Angular, React, Vue.js都是很好的框架。

我们在我公司的实际项目中，都使用Vue.js. 效果非常好。开发速度快，维护效率高。招人也方便。

本文跟官方文档不同。是根据实际项目经验，以 培养新人的角度来写的，所以会有这样的特点：

- 我认为“很少使用的”技术，略过
- 只讲解最常见的知识
- 在章节上，按照入门的难易度从简单到复杂。

学习的时候，务必要有个开发环境，看一点儿文档，就动手敲一敲代码。

否则光看是看不懂的。

本书中的所有示例源代码，都可以在 https://github.com/sg552/happy_book_vuejs 上找到。

为什么要使用Vuejs?

在本章中，我们会从若干角度来思考这个问题。

Single Page App

可以认为，世界上的Web 应用分两类：

1. 传统Web页面应用
2. 单页应用

传统Web页面

就是我们打开浏览器，整个页面都会打开的应用。

例如，我的个人网站 <http://siwei.me>，就是一个最典型的“传统Web应用”，每次点击其中的任意一个链接，都会引起页面的整个刷新。如下图所示：

github - 邀请伙伴后，需要修改权限

2018-06-20 15:02

访问量: 34

分类: 技术

在github 的organization 中，默认权限是read .

需要管理员 进入到 organization -> settings 中，选择 member privilage ， 然后把member的权限，从

[« ruby/rails - 根据浏览器的语言，来自动识别](#)

Name	Status
application-50c040004fde00753430100005340320.css	200 OK
/assets	
formatting-ec53a0525fb04a18d77bb957bd928285.css	200 OK
/assets	
theme-8830d623340bbbb8ecccbf3aa7b1f6a0.css	200 OK
/assets	
frontend-aaeb420e435264d2d4a52828dd967b49.css	200 OK
/assets/refinery/blog	
modernizr-min-3ce1c8d18a888ccb8b2c088735aaff76.js	200 OK
/assets	
application-1810653e7aacadf37cb15c4d97776322.js	200 OK
/assets	
header-4cf66dc6e4e154c259c682cf425d30f0.png	200 OK
/assets/template	
nav-a87cc3553dceb71e8e834fe0716fb5f1.png	200 OK
/assets/template	
postbullets-e85b00531db9f57687bc8fcf91a0d96f.png	200 OK
/assets/template	

10 requests | 3.0 KB transferred | Finish: 815 ms | DOMContentLoaded: 837 ms | Load: 859 ms

从上图中可以看出，传统的页面，每次打开，都要把页面中所有的 “.js”， “.css”， 图片文件，html文件等等所有的资源，都要加载一遍。在上图中的左下角可以看到，本次总共加载了 10个请求，(4个css, 2个js, 3个png图片，还有 1个html文件)。耗时 0.837s .

这个在PC端可以，但是在手机端会巨慢无比。特别是安卓机。

传统页面的特点，就是，下面的任何一个操作，都会引起浏览器对于整个页面的刷新：

1. 点击
2. 提交
3. 触发 `location.href='...'` 这样的js代码的时候。

我们看个最传统的web页面的例子，如下：

```
1. <html>
2. <head>
3.   <script src="my.js"></script>
4.   <style src="my.css"></style>
5. </head>
6. <body>
7.   <img src='my.jpg' />
8.   <p> 你好！ 传统Web页面！ </p>
9. </body>
10. </html>
```

每个浏览器，都会从第一行忠实的解析到最后一行，然后再继续加载 `my.js` , `my.css` , `my.jpg` 这三个外部资源。

其实很好理解。这个就是大家最初想象的Web页面的打开方式。

Single Page App

单页应用，确切的诞生时间不详，可以肯定的是这个概念在2003年就在论坛上被人讨论了。在2002年四月，诞生了一个网站：<http://slashdotslash.com>，就使用了这种思想。

单页应用的精髓，就是点击任何链接，都不会引起页面的整体刷新。只会通过javascript，来替换页面的局部内容。

Ajax : Asynchronous javascript and XML

说到这里，就不得不 提到另一个概念： Ajax .

中文可以称呼为： js的异步请求。 不过国内统一都叫Ajax .

Ajax 的概念是： 每次打开新的网页时， 不要让页面整个刷新，而是由“javascript语言”，发起一个 “http 异步请求”，这个“异步请求”的特点，是不会让当前的网页卡死。

用户可以一边上下滚动页面，播放mp4， 一边等待这个 请求返回数据。 等这个“response” 正常返回后， 由“javascript”控制，来刷新页面的局部内容。

这样的好处是：

1. 大大节省了页面的整体加载时间。 各种 .js, .css 等资源文件，加载一次就够了。
2. 节省了带宽。

3. 同时减轻了客户端和服务端的负担。

在智能手机和 App 应用（特别是微信）流行起来之后，大量的网页都需要在手机端打开，所以 Ajax 的优势体现的淋漓尽致。

虽然 Ajax 的名字本意是“异步 js 与 XML”，不过现在在服务器端返回的数据中，几乎都是使用了“json”，抛弃了“XML”

在 2005 年，国内的程序员论坛开始提及 Web 2.0，其中的 Ajax 技术被人重视。到了 2006 年初，可以说 Ajax 是前端程序员的加薪利器。市面上的所有“前端 Web 程序员”职位都认为 Ajax 是一个巨大的加分项。

可惜当时 jQuery 在国内不是很普及，Prototype 也没有流行起来。我在北京和圈子里的各大公司的同行们交流时，发现大家用的都是“原生的 javascript Ajax”。这种不借助任何第三方框架的代码写起来非常臃肿，累人，而且考虑到浏览器的兼容问题，应用起来很让人头大。

例如，当时的代码样子往往是这样的：

```
1. var xmlhttp = false;
2. /*@cc_on @*/
3. /*@if (@_jscript_version >= 5)
4. try {
5.     xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
6. } catch (e) {
7.     try {
8.         xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
9.     } catch (e2) {
10.        xmlhttp = false;
11.    }
12. }
13. @end */
14. if (!xmlhttp && typeof XMLHttpRequest != 'undefined') {
15.     xmlhttp = new XMLHttpRequest();
16. }
```

上面的代码仅仅是为了兼容各种浏览器。实际上后面还有几十行的冗余代码，才能正式开始干活。

到了 2008 年，国内开始流行 prototype, jquery 之后，发起一个 Ajax 请求的代码精简成几行：

```
1. jQuery.get('http://some_url?para=1', function(data){
2.     // 正常代码
3. })
```

那时候开始， Ajax在国内变得越来越普及。

Angular

第一个SPA的知名框架应该是Angular. 由Google在2010年10月推出。当时的Gmail, google map等应用对于Ajax技术运用到了极致。而Angular框架一经推出，立刻引燃了单页应用这个概念。

尽管后来各种SPA框架层出不穷，在2015年以前，Angular 稳坐SPA的头把交椅。

当下的SPA技术趋势

成为了项目开发必不可少的内容，只要有移动端开发，就会面临两个选择：

1. 要么做成原生App
2. 要么做成SPA H5

无论是IOS端还是Android，都对SPA青睐有加：

1. 打开页面速度特别快。

打开传统页面，手机端往往需要几秒，而SPA则在0.x 秒内。

2. 耗费的资源更少。

因为每次移动端只请求接口数据。和非要不可的图片资源。

3. 对于点击等操作响应更快。

对于传统页面，手机端的浏览器在操作时，点击按钮的话，往往会有0.1s 的卡顿，而使用SPA则不会有卡顿的感觉。

4. 可以保存浏览的历史和状态。

不是每一个Ajax框架都可以由这一点内容。例如大家常用的QQ邮箱，虽然也是页面的局部刷新，但是每次打开不同的邮件时，浏览器的网址不会变化。

而在所有的SPA框架中，都会有专门处理这个问题的模块，往往叫做“router（路由）”。

例如：

`http://mail.my.com/#/mail_from_boss_on_0620` 对应 老板在6月20日发来的邮件。

`http://mail.my.com/#/mail_from_boss_on_0622` 对应 老板在6月22日发来的邮件。

在2011, 2012年，各种SPA框架出现了井喷的趋势，包括“backbone”，“ember.js”等几十上百个不同的框架。

什么是单页应用

到了最近几年， Angular， React， Vue.js是三种最流行的框架。

知名的单页应用(SPA) 框架对比

在学习Vue.js之前，我们要知道为什么要学习它。

目前市面上最知名的SPA框架，包括： Vue.js, React, Angular。我们依次来看一下~

Angular

作为SPA的老大哥，源于Google，在过去若干年发挥了巨大的价值。现在的版本是4.0

它的优点特点是：

1. 业内的第一个SPA框架
2. 实现了前端的MVC解耦
3. 双向绑定。model层的数据发生变化会直接影响View. 反之亦然。

缺点也很明显：

1. 难学，难用，
2. Angular 1.x 的文档很烂。2.0 稍微好一些。

我个人曾经在13年7月做项目用到它，当时我还服务于优酷。

angular 1. 它的文档：directive. 被无数人(老外)吐槽：看不懂。'The worst document that I've ever read'. 3个月没变。(记得当时的页面的很多留言都是一年以前的)

文档不全，没有示例代码。很多东西调试起来也没有专门的工具。

另外，想使用第三方组件的话，需要单独为Angular做适配。例如 jquery-upload 这个大名鼎鼎的前端上传文件的组件。用起来非常不好用。

总之，功能很全面，但是由于学习曲线过于陡峭，上手很慢，维护起来麻烦。

所以现在在论坛上的口风开始下降。

官方网站：<https://github.com/angular> 截止到 2018.6月下旬，github 关注数 3.8万

React

由Facebook推出的SPA框架。是与Vue.js并列的两大框架之一。宣称的特点“Learn once, write anywhere”很吸引人。

优点：

1. 使用js 一种语言就可以写前端(h5, app) + 后端.
2. ReactNative 可以直接运行在手机端，性能很棒，接近于原生App. 并且可以热更新， 免去了手机端App每次都要重新下载安装才行。
3. 周边组件很多。

缺点：

1.html代码(

这样的标签) 需要写在js 文件中， 例如：

```
1. class HelloMessage extends React.Component {  
2.     render() {  
3.         return (  
4.             <div>  
5.                 Hello {this.props.name}  
6.             </div>  
7.         );  
8.     }  
9. }  
10.  
11. ReactDOM.render(  
12.     <HelloMessage name="Taylor" />,  
13.     mountNode  
14. );
```

这样的“多语言混合式编程” 引起的最大特点，是代码难以理解，非常难以开发和调试。

2.把前后端代码写在一起的风格

```
1. //前端代码  
2.  
3. ....  
4.  
5.  
6. // 后端代码  
7. ....
```

所有传统web框架过的人(java, rails, php) 都觉得奇怪.

其他表现尚可。 学习难度低于 Angular, 但是高于Vuejs.

官方网站： <https://github.com/facebook/react> 截止2018年6月底，关注人数是10.4万。

Vuejs

Vuejs(读音同 “View”) 是一个MVVM (Model - View - ModelView) 的SPA框架。

- View: 视图
- Model: 数据
- ModelView: 连接View与Model 的纽带

Vuejs一经推出，就获得了各大社区的好评。 几乎是一边倒的声音。 它的优点是：

1. 简单好学， 好用 .

- Angular: 学习二周到4周
- React: 学习两周
- Vuejs: 三天到一周 .

做的事情都一样 .

2. Angular, React具备的功能， Vuejs都具备。 (React Native 除外)

3. 作者尤雨溪 (个人网站: <http://evanyou.me>) 是中国人. 目前在美国google工作

所以Vuejs 在2014年2月推出的时候，核心文档就具备了两种语言：中文、英文. 这对于母语是汉语的国人来说意义重大。 可以非常快的上手。

官方网站 <https://github.com/vuejs/vue> 截止2018年6月底，关注人数10.5万。 是三大框架中的最高者。

为什么用vue , 不用 react, angular?

我们在评价一个技术的时候，一个最简单的办法，就是看它有多“火”。 这个体现在Github的 stars 数目上。

截止到 2018年6月底， 三个项目的关注数分别是：

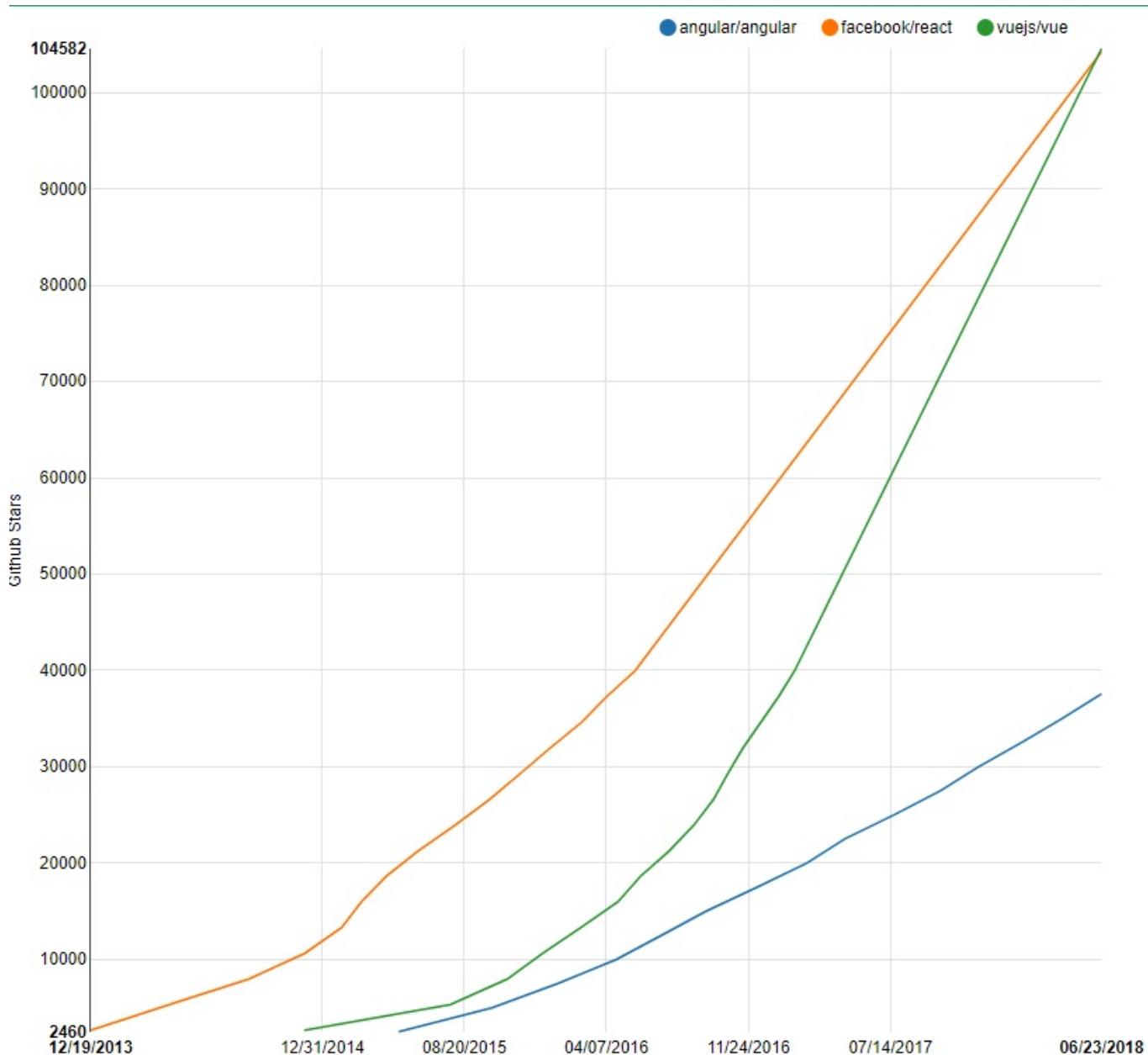
- Angular: 3.8万
- React: 10.4万
- Vuejs: 10.5万

可以看出， Vuejs 第一。

其次，我们看一下 stars 的增长趋势。

根据统计， (<http://www.timqian.com/star-history/#facebook/react&angular/angular&vuejs/vue>)， 截止到2018-6-23,github

star曲线如下图所示，可以看出， Vuejs 的增长势头一直是最高的， React居中， Angular最低：



第三，Vuejs的作者就是中国人，官方文档之一就是中文。（地址：<http://cn.vuejs.org>）这个对于普遍英语水平不好的国人程序员来说，可以非常快的上手，是个巨大的优势。

参考文章：

<https://cn.vuejs.org/v2/guide/comparison.html>

<https://www.quora.com/How-does-Vue-js-compare-to-React-js>

被腾讯和阿里巴巴所青睐

Vue.js的思想，可以说是对国内互联网巨头产生了巨大的冲击。 腾讯的微信，和阿里巴巴的Weex项目的实现方式，跟Vue.js是非常相似的。

可以说，学会了Vue.js，就基本学会了 微信小程序和 阿里巴巴的Weex .

这个对于需要不断学习新知识的程序员来说，是非常好的消息。 有大公司支持，这个技术一定是非常有前景的。

微信小程序

微信小程序是微信在 2017年出现的技术， 基于微信。 使用SPA 的开发技术，就可以运行在安装过微信的手机上。

表现效果跟 原生App 几乎一样。

微信小程序的代码特点，文件组织形式，以及各种概念，跟 Vue.js是非常相似的。

阿里巴巴 Weex

Weex 致力于使开发者能基于当代先进的 Web 开发技术，使用同一套代码来构建 Android、iOS 和 Web 应用。

Weex 以及支持了对于vue.js 2.0的直接集成。 也就是说，可以在Weex中直接写Vue.js的代码。 而 Weex跟ReactNative 一样，都是使用Web开发的相似代码，让程序以Native App的形式跑起来。

在这里： <https://weex.apache.org/cn/guide/use-vue.html> Weex 以官方文档的形式告诉大家如何使用vue.js .

用到Vue.js的项目

参考知乎: <https://www.zhihu.com/question/33264609?sort=created>

- 滴滴，还出了一本书：Vue.js权威指南。
- 饿了么，开源了一个基于Vue的UI库(<https://github.com/ElemeFE/element>)
- 阿里的 weex (<https://github.com/alibaba/weex>)
- GitLab <https://about.gitlab.com/2016/10/20/why-we-chose-vue/>
- 大疆
- facebook: <https://newsfeed.fb.com/welcome-to-news-feed?lang=en>
- 新浪微博

更全列表，见：<https://github.com/vuejs/awesome-vue#projects-using-vuejs>

原生的Vuejs

所谓的原生Vuejs，就是独立的Vuejs框架，不与Webpack等框架结合使用。

学习这个比较有必要。在后期看官方文档的时候，很多概念都是用“原生Vuejs”的形式说明的。脱离了其他框架，说明起来更加简明一些。

虽然我们在做项目的时候极少会使用原生Vuejs，但是对于未来的学习大有好处。

急速入门

如果只从体验的角度来看， Vuejs 的安装非常简单。只需要引入一个第三方的js包即可。

```
1. <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
```

下面是一个最简单的例子：

```
1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.     {{ show_my_text }}
8.   </div>
9.   <script>
10.    var app = new Vue({
11.      el: '#app',
12.      data: {
13.        show_my_text: 'Vuejs is the best one page app!'
14.      }
15.    })
16.   </script>
17. </body>
18. </html>
```

上面的代码非常简单，

1. 在head中引入 vuejs 包
2. 在 中，定义了一个
， 可以认为，所有的页面展示，都是在这个
中。 每次我们做任何点击的时候， 整个页面不会刷新， 都是vuejs框架操作代码，对
中的内容进行局部刷新。
3. 后面的 `var app=new Vue...` 这里就是真正的操作代码：

```
1.       var app = new Vue({ // 表示创建了一个Vue对象
2.         el: '#app', // 指定 所有的 代码操作，都是对于 <div id='app' >
3.         的元素来操作的
4.         data: { // data 表示为 Vuejs 管理的变量赋值。 这个变量的名字
5.           是 show_my_text
6.         }
7.       })
8.     }
9.   }
10. }
11. 
```

```
4.           show_my_text: 'Vuejs is the best one page app!'
5.       }
6.   })
```

使用浏览器打开这个页面后， 就可以看到：



源代码

可以在 code_example/hello_world_bare_vuejs 中看到。 并直接运行。

Webpack下的Vuejs

所有的Vuejs 项目，都是在Webpack的框架下进行开发的。

可以说， vue-cli 直接把webpack做了集成。 我们在开发的时候，一边享受着飞一般的开发速度，一边体验着webpack带来的便利。

在本章，会介绍如何使用两者进行开发的基本知识。

学习过程

我们在学习任何一种框架的时候，都是按照循序渐进的顺序来的：

1. 安装
2. 新建一个页面
3. 做一些简单的变量的渲染
4. 实现页面的跳转（路由）
5. 实现页面间的参数的传递（路由）
6. 实现真实的http请求（访问接口）
7. 提交表单
8. 使用一些技巧来让代码层次化（组件）

按照我之前在惠普，联通，移动等公司的讲课经验，只要用一天的时间把本章内容学会，就可以上手开发Vuejs项目了。

同学们只要手头有个电脑，看一个章节，就跟着老师来敲一些代码，就肯定可以在一天内把这些基本的内容消化完。

可以跳过的章节

对于有一定node基础的同学，可以跳过 “NVM的安装”

对于使用Linux/Mac的 同学，可以跳过 “Git Bash的安装”

简写说明

由于本章节是 Webpack + Vuejs 下的实战开发，所以统一使用 “Vuejs” 来代替冗长的 “Webpack + Vuejs” .

例如，

1. 在Vuejs中创建页面需要两步：
- 2.
3. 1. 新建路由
4. 2. 新建vue页面

中的“Vuejs”，指的就是在“Webpack + Vuejs”的项目中。而不是“原始Vuejs”。

例子

本章节中的所有例子，由于都需要跟webpack相结合，所以我把它单独拉出来成为一个项目：

https://github.com/sg552/vue_js_lesson_demo

大家可以下载后直接运行。

1. \$ git clone https://github.com/sg552/vue_js_lesson_demo.git
2. \$ npm install -v
3. \$ npm run dev

然后就可以在 <http://localhost:8080/#/> 中看到效果，如下图所示：



例子列表

- [Hello](#): 显示最基本的Vuejs
- [SayHi](#): 第一个Vue页面
- [SayHiFromVariable](#): 在页面中使用参数
- [BlogList](#): 调用真实接口，并渲染出博客列表
- [Blog](#): 调用真实接口，显示blog文章
- [TwoWayBinding](#): 一个双向绑定的例子。
- [FormInput](#): 表单输入项大全。
- [FormSubmit](#): 提交表单的例子。
- 组件的使用，见：[BlogList](#) 和 [Blog](#)
- [SayHiFromMixin](#): Mixin的例子。
- Vuex的例子：[倒计时页面1](#)
- [倒计时页面2](#)

文字版教程在：https://github.com/sg552/happy_book_vuejs

NVM, NPM 与 Node

NVM, 全名为“Node Version Manager”，是非常好用的Node版本管理器。

这个技术出现的原因，是因为很多时候，我们的不同的项目，node版本是不同的。有的是
5.0.1，有的是 6.3.2

如果node版本不对的话，运行某个应用时，很可能会遇到各种莫名其妙的问题。

所以，我们需要在同一台机器上，同时安装多个版本的node。所以NVM应运而生，很好的帮我们解决了这个问题。

Linux/Mac下的NVM官网：<https://github.com/creationix/nvm>

Windows下的NVM官网：<https://github.com/coreybutler/nvm-windows>

NPM, 全名为“Node Package Manager”。是只要安装了node，就会捆绑安装这个命令。它的作用跟“ruby中的 bundler”，“java中的maven”相同，都是对第三方依赖进行管理的。

安装

Windows 下的安装：

1. 用浏览器打开：<https://github.com/coreybutler/nvm-windows/releases>

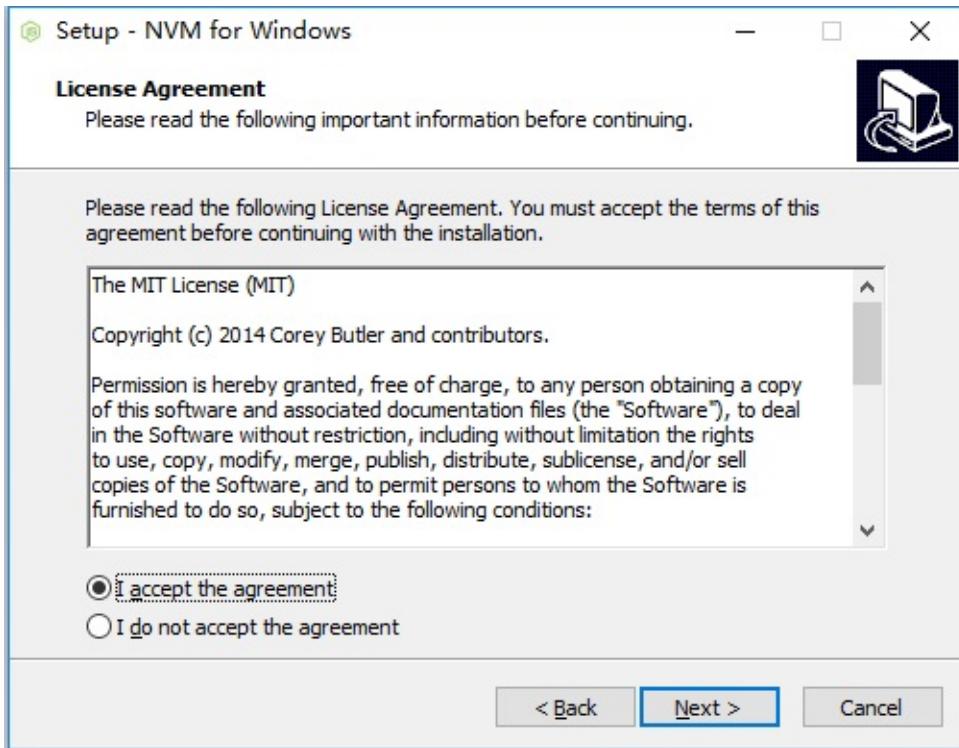
The screenshot shows the GitHub repository page for `nvm-windows`. At the top, there are navigation links for Code, Issues (94), Pull requests (17), Projects (1), Wiki, and Insights. Below these, there are tabs for Releases and Tags, with Releases being the active tab. A specific release is highlighted: "1.1.6" (Pre-release). It was released by `coreybutler` on 16 Jul 2017, with 14 commits to master since this release. The commit hash is `c6c5d2a`. The "Assets" section lists four items: `nvm-noinstall.zip` (1.98 MB), `nvm-setup.zip` (1.73 MB), `Source code (zip)`, and `Source code (tar.gz)`. A note below states: "Leverages Go's native filepath management, which should fix issues with spaces and special characters in path names."

2. 点击最新的release版本下载。例如上图中的“1.1.6 中的nvm-setup.zip”文件。

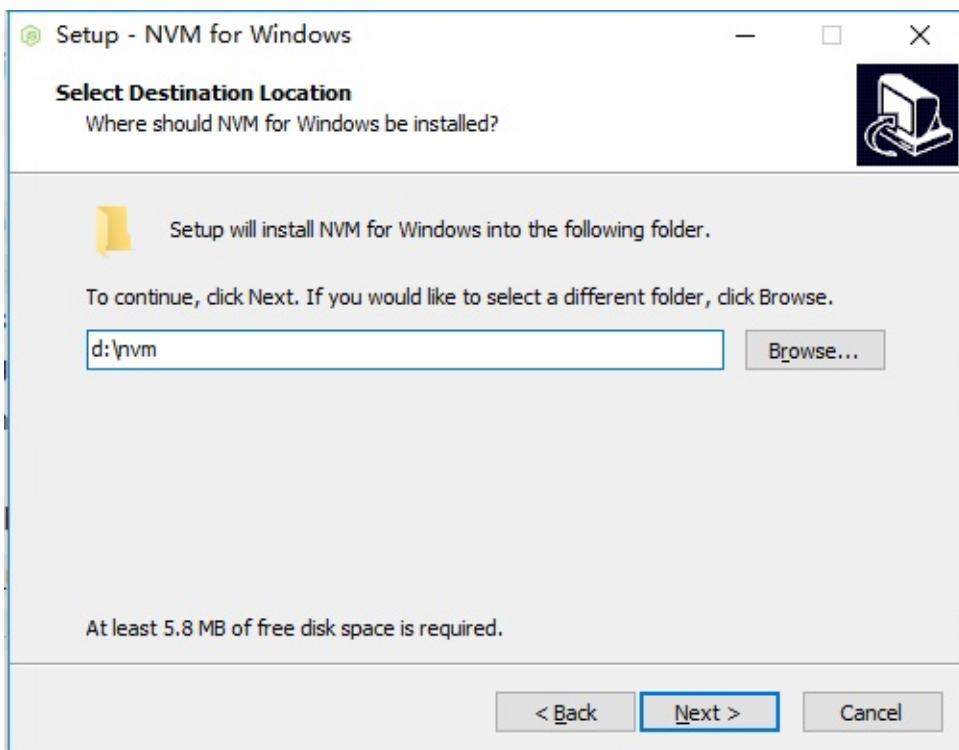
3. 下载后，解压缩这个 `.zip` 文件，双击其中的 `nvm-setup.exe` 文件，就可以开始我们的安装了。如下图：



4. 点击下一步，选择接受，如下图所示：

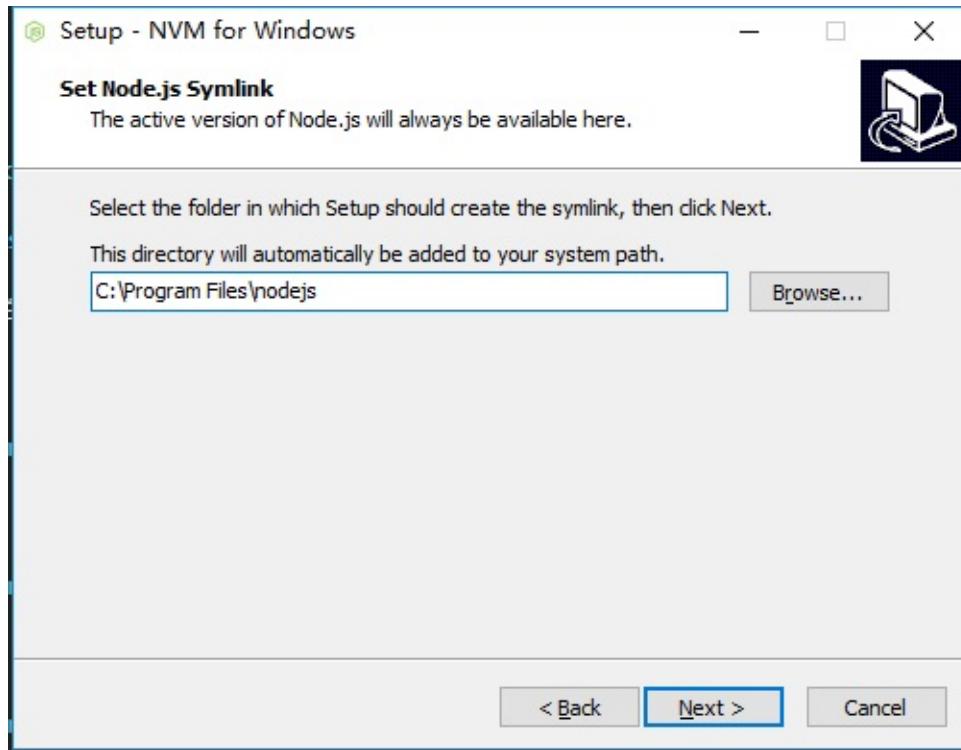


5. 继续下一步，选择安装路径。在我的机器上，我把它安装到 `D:\nvm`。如下图所示：



6. 继续下一步，询问我们把nvm这个命令的快捷方式放在哪里。（symlink的作用同快捷方式，允许我们在任意路径下都可以调用nvm命令）

不用修改，直接下一步，如下图所示：

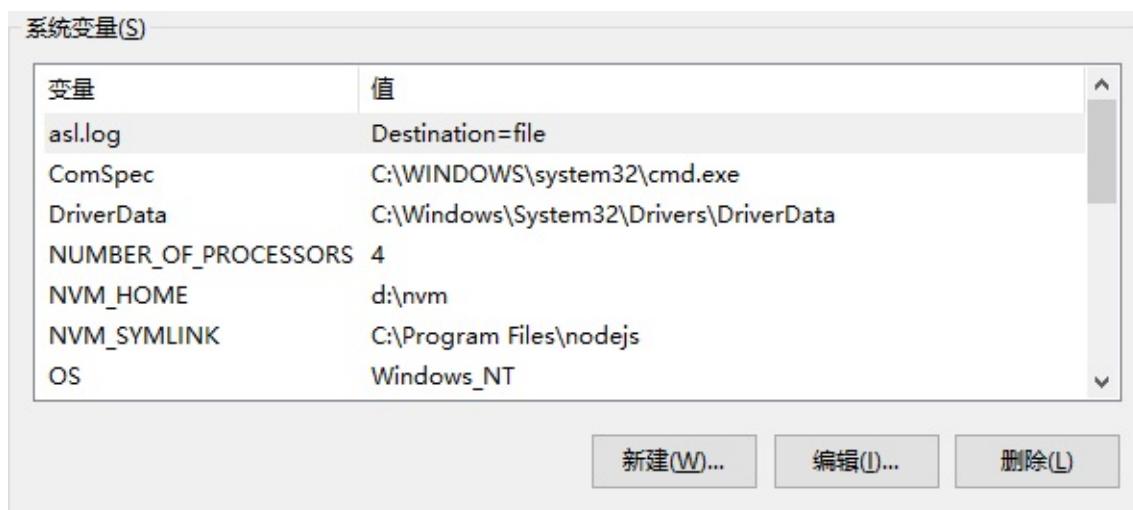


7. 然后会弹出确认安装页面。我们继续点击下一步就可以了。

8. 最后，设置环境变量：

1. NVM_HOME D:\nvm
2. NVM_SYMLINK C:\Program Files\nodejs

对于环境变量的修改： 从控制面板中，进入到“所有控制面板项目”->“高级系统配置”->“环境变量”。 如下图所示：



对于PATH的修改，则是在原有的值的基础上，添加“%NVM_HOME%, %NVM_SYMLINK%”，如下图所示：

windows下的path配置

Linux, Mac下的安装：

1. 下载nvm的源代码。 运行下面命令即可。

```
$ git clone https://github.com/creationix/nvm.git ~/.nvm && cd ~/.nvm && git  
1. checkout `git describe --abbrev=0 --tags`
```

2. Linux, Mac的用户：为脚本设置启动时加载（对于使用windows的同学，可以直接无视第二步，到官网下载exe安装文件就可以了）。

把下面这行代码放到：`~/.bashrc`，或 `~/.bash_profile`，或 `~/.zshrc` 中。

```
1. $ source ~/.nvm/nvm.sh
```

运行

注意： 不能使用 `$ which nvm` 来验证安装是否成功。因为即使成功了也不会返回结果。

直接在命令行下， 输入

```
1. $ nvm
```

就可以了。 如果安装成功的话，会看到一些英文。如下所示：

```
1. Running version 1.1.6.  
2.  
3. Usage:  
4.  
5.   nvm arch           : Show if node is running in 32 or 64 bit mode.  
   nvm install <version> [arch] : The version can be a node.js version or  
6. "latest" for the latest stable version....  
   nvm list [available]      : List the node.js installations. Type  
7. "available" at the end to see what can be ...  
8.   nvm on               : Enable node.js version management.
```

使用NVM 来安装或者管理node版本

1. 列出所有可以安装的node版本：

windows 下的命令：

```
1. $ nvm list available
```

Linux/Mac下的命令：

```
1. $ nvm list-remote
```

就可以看到可以安装的所有版本. (下面是Windows中的例子。 Linux, mac下的也非常类似)：

```
1. $ nvm list available  
2.  
3. | CURRENT | LTS | OLD STABLE | OLD UNSTABLE |
```

```
4. |-----|-----|-----|-----|-----|
5. | 10.5.0 | 8.11.3 | 0.12.18 | 0.11.16 |
6. | 10.4.1 | 8.11.2 | 0.12.17 | 0.11.15 |
7. | 10.4.0 | 8.11.1 | 0.12.16 | 0.11.14 |
8. | 10.3.0 | 8.11.0 | 0.12.15 | 0.11.13 |
9. | 10.2.1 | 8.10.0 | 0.12.14 | 0.11.12 |
10. | 10.2.0 | 8.9.4 | 0.12.13 | 0.11.11 |
11. | 10.1.0 | 8.9.3 | 0.12.12 | 0.11.10 |
12. | 10.0.0 | 8.9.2 | 0.12.11 | 0.11.9 |
13. | 9.11.2 | 8.9.1 | 0.12.10 | 0.11.8 |
14.

This is a partial list. For a complete list, visit
15. https://nodejs.org/download/release
```

2.列出本地安装好了的版本:

```
1. $ nvm list
```

结果形如:

```
1. $ nvm list
2.
3. * 10.5.0 (Currently using 64-bit executable)
4. 6.9.1
```

在上面的结果中，表示当前系统安装了两个node版本，一个是“6.9.1”，一个是“10.5.0”。默认的node版本是“10.5.0”

3.安装

选择一个版本号，就可以安装了。 wind

```
1. $ nvm install 10.5.0
```

安装好之后，退出命令行并重新进入即可。

4.使用

下面的命令，是为当前文件夹指定 node 的版本。

```
1. $ nvm use 10.5.0
```

对于Linux, Mac, 如果希望为系统全局都使用某个版本, 就可以运行下面的命令:

```
1. $ nvm alias default 10.5.0
```

在Linux, Mac下, 还可以把它放到 `~/.bashrc` 或者 `~/.bash_profile` 中。这样系统每次启动, 都会自动指定node作为全局的版本。

删除NVM

对于Linux, Mac, 直接手动删掉对应的配置文件(如果有的话)即可。

- `~/.nvm`
- `~/.npm`
- `~/.bower`

对于Windows, 则直接在控制面板中卸载软件就可以。

对于国内用户, 加快 NVM 和 NPM 下载速度的办法

由于某些原因, 在国内连接国外的服务器会比较慢, 所以我们使用下面的命令, 就可以在国内的镜像服务器下载node了。感谢淘宝提供~

对于NVM(安装不同的node时使用), 使用 `NVM_NODEJS_ORG_MIRROR` 这个变量作为前缀。

```
1. $ NVM_NODEJS_ORG_MIRROR=https://npm.taobao.org/dist nvm install
```

对于NPM(安装某些npm第三方包时使用), 则用cnpm代替 npm 命令。

```
1. $ npm install -g cnpm --registry=https://registry.npm.taobao.org
```

对于Linux, Mac用户, 可以通过直接创建一个“alias”命令:

```
1. alias cnpm="npm --registry=https://registry.npm.taobao.org \
2.   --cache=$HOME/.npm/.cache/cnpm \
3.   --disturl=https://npm.taobao.org/dist \
4.   --userconfig=$HOME/.cnpmrc"
```

然后, 就可以通过国内的淘宝服务器来安装node的包了, 例如:

```
1. $ cnpm install vue-cli -g
```

Git 在Windows下的使用

在《程序员修炼之道：从小工到专家》这本书中，提到了一个让程序员非常尴尬的局面：老板要看进度，结果程序员拿不出来，只好跟老板撒谎：我的代码被猫吃了。

虽然我们的代码不会被猫吃掉，但是几乎每个程序员都会犯的错误是：在下班的时候突然不小心忘记保存，或者突然断电，结果之前敲了几个小时的代码就没了。

所以，每个程序员必须要对自己的代码做版本控制。

在2009年之前，国内的人大部分都用 SVN。在2010年开始，越来越多的人开始使用Git。

本节专门为 Windows 程序员准备。因为对于 Linux 和 mac 用户来说，Git 都是现成的。一行命令搞定。所以就不赘述了。

为什么要使用 Git Bash

因为 Git Bash 不但提供了 Git，还提供了 bash，一种非常不错的类似于 Linux 的命令行。

在 Windows 环境下，命令行模式跟 Linux/Mac 是相反的。例如：

- Linux/Mac 下：（使用左斜线作为路径分隔符）

```
1. $ cd /workspace/happy_book_vuejs
```

- Windows 下：（使用右斜线作为路径分隔符，并且要分成 C, D... 盘）

```
1. C:\Users\dashi>d:                                (进入到D盘)  
2. D:\>cd workspace\happy_book_vuejs           (进入到对应目录)
```

只要您不是做 “.NET/微信小程序/安卓” 开发，我认为都应该转移到 Linux 平台上。原因是：

你的代码在编译后，会运行在 Linux + Nginx 的服务器中。

最好的办法就是，从现在开始就适应 Linux 的环境。

另外，命令行在绝大多数情况下，比“图形化”的操作界面好用。

安装 git 客户端

我们在 windows 下，选择 Git Bash。

安装前准备: Git Bash

官方网站: <https://gitforwindows.org/>

1. 打开后, 就可以看到LOGO, 如下图所示:

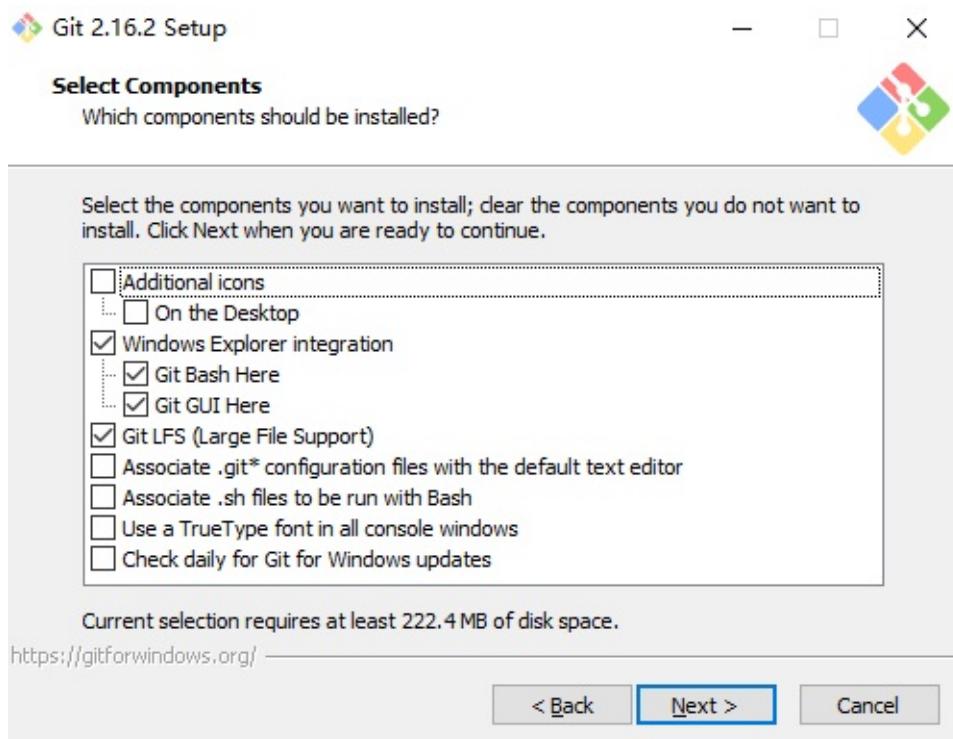
```
git bash logo
```

下载最新版本。 我下载的是 2.16.2

2. 下载后, 运行。 可以看到欢迎页面, 如下图所示:



3. 选择next，看到选择安装的内容页面，我们就用默认的就好，如下图所示：



4. 选择next，看到选择哪个编辑器作为git消息编辑器，

我们可以选择下面四个编辑器：

- nano : 最简单的 Linux 下的编辑器，同 windows下的记事本。 学习曲线是 0
- vim : 需要用一辈子去学习的编辑器。 编辑器之神。 也是我的最爱。
- notepad++ : 加强型记事本。 也很好用。 学习曲线0
- visual studio code: 一款IDE。

对于新手来说, 建议选择 notepad ++, 如下图所示:

```
git bash install1
```

5. 选择next, 询问我们使用什么风格的命令行。

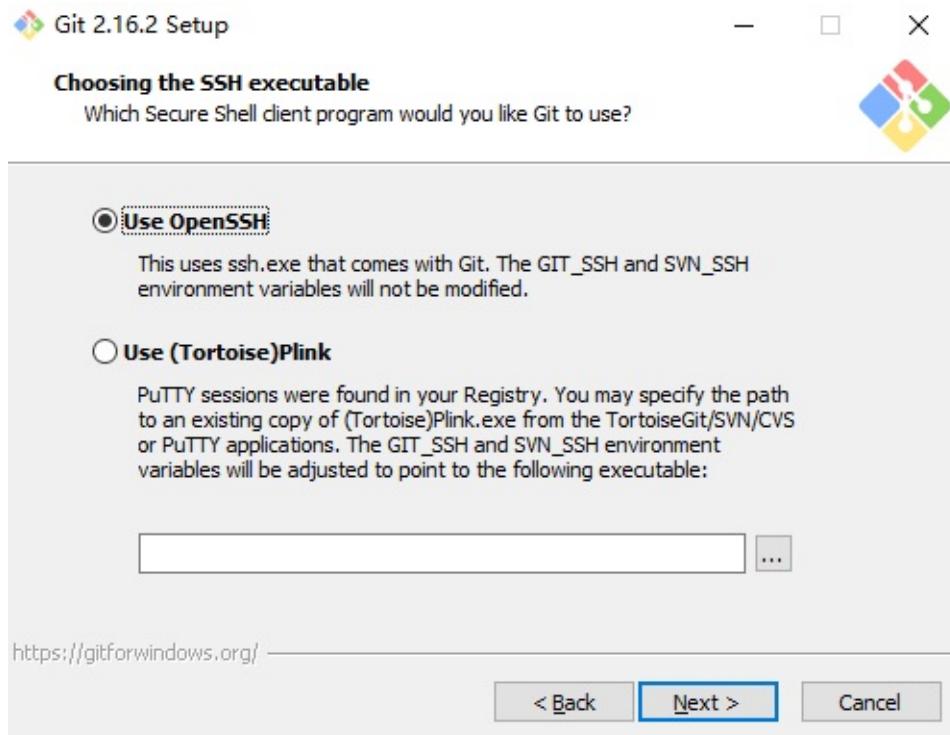
这里建议使用默认的 “Windows Command Prompt”就好了。 如下图所示:

```
git bash install1
```

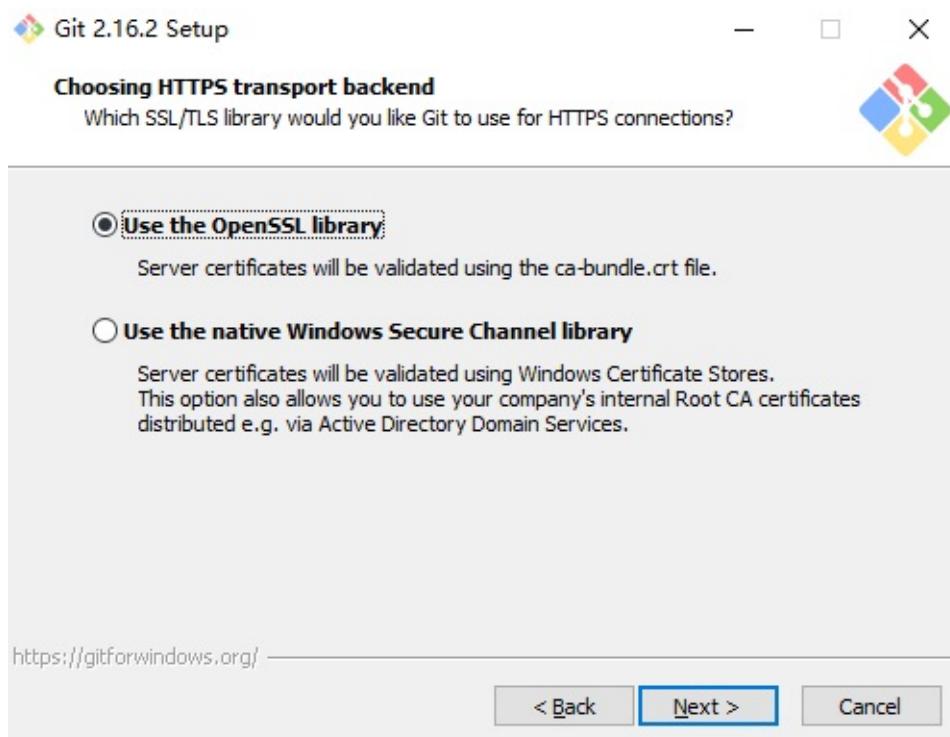
6. 选择next，询问我们使用什么风格的SSH连接程序。

- OpenSSH SSH的首选，这个是Git bash自带的。
- Plink 第三方用户自己安装的SSH连接程序。

如下图所示：



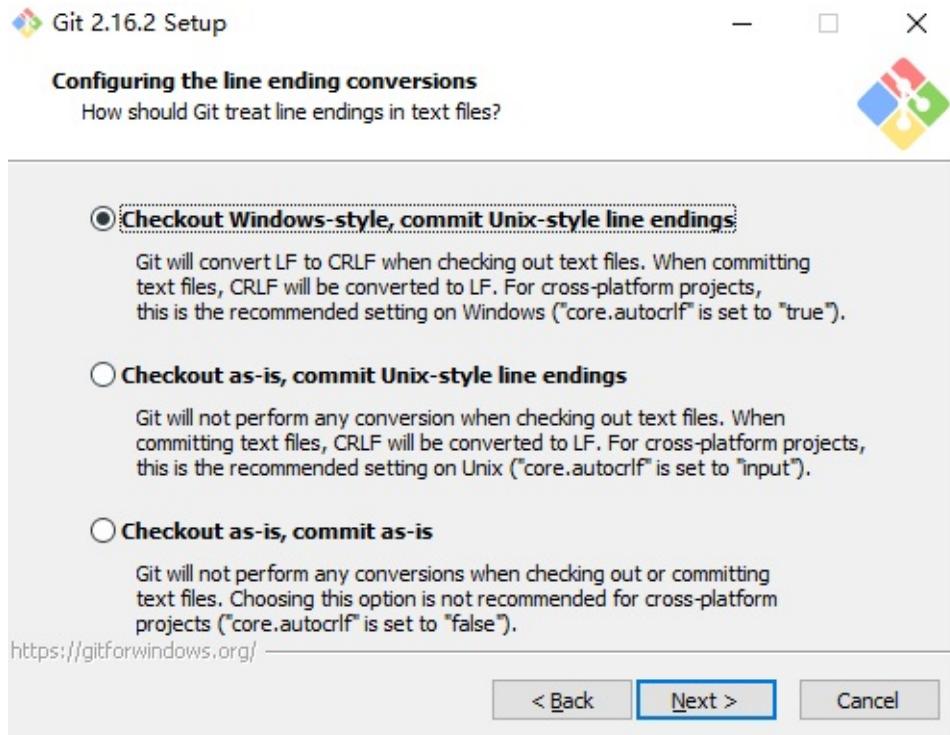
7. 选择next，询问我们使用什么SSH 后端。仍然选择默认的 OpenSSL. 如下图所示：



8. 选择next，询问我们使用什么 checkout/commit 风格。

因为windows跟linux 对待文件的处理是不同的，例如回车在windows下是 \r\n，而在linux下就是 \n .

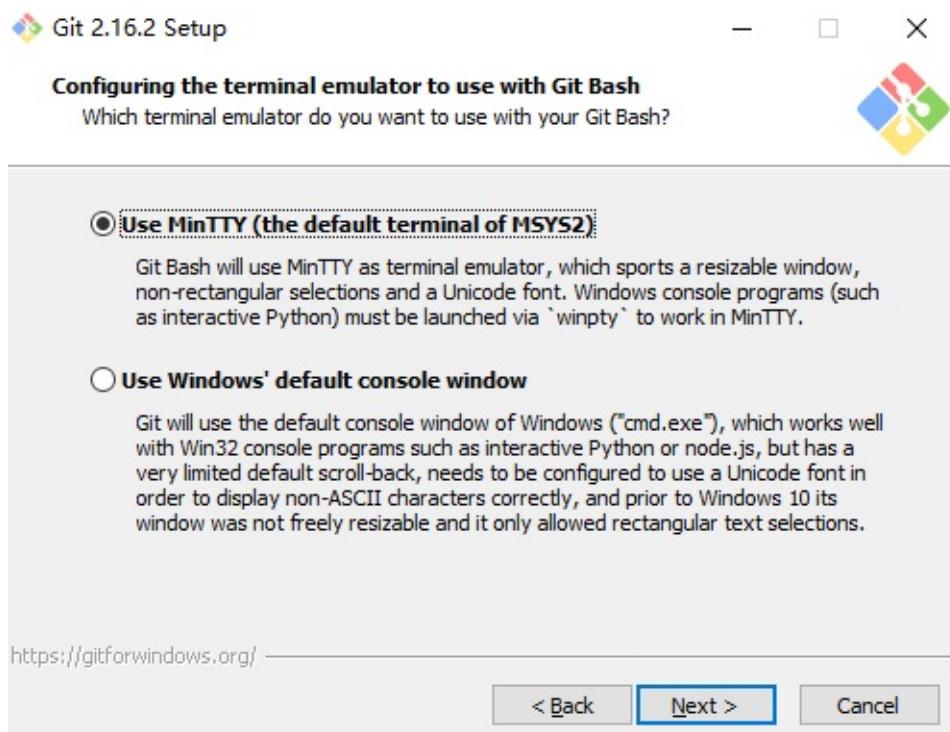
所以，我们这里就选择默认的第一项就好（用windows风格checkout，用unix 风格commit）。如下图所示：



9. 选择next，询问我们用什么风格的console. (命令行)

这里一定要选择“MinTTY”，也就是类似于Linux风格的命令行。可以说，它就是非常著名的Cygwin。

如下图所示：



10. 选择next，询问其他配置项目。直接选择默认就好，如下图所示：

```
git bash install1
```

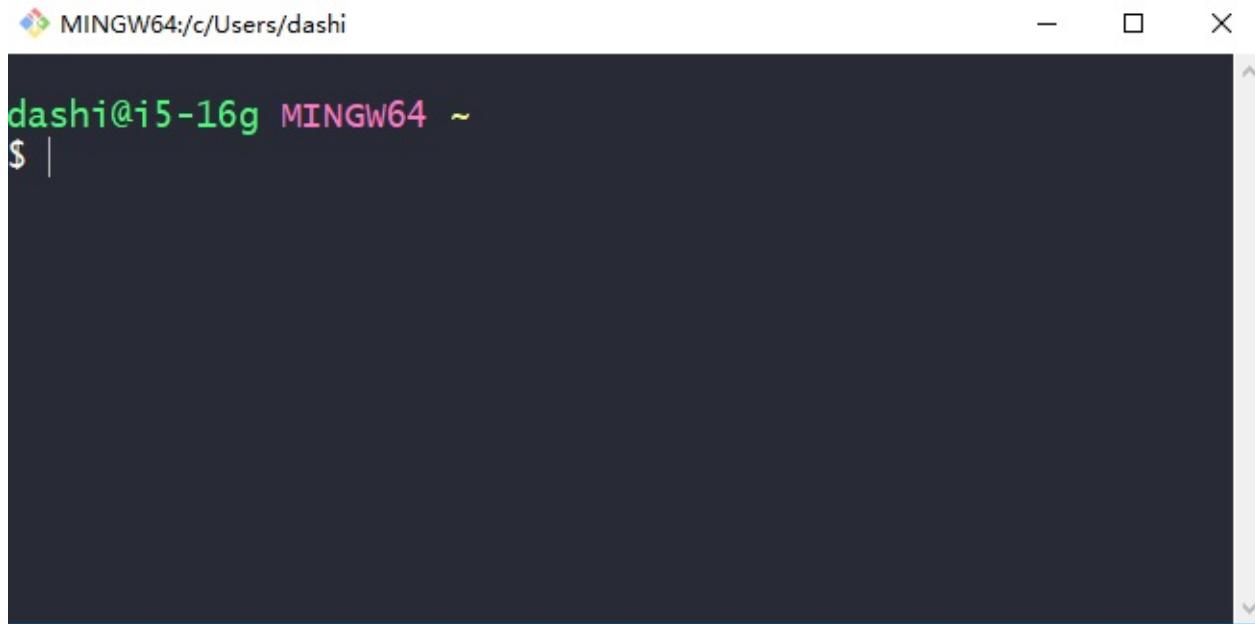
接下来我们就一路next 下去。就可以安装成功了。 如下图所示:



使用 Git Bash

1. 打开git bash

我们打开“Git Bash”，可以看到这个页面，如下图所示：



一片空荡荡。估计习惯了鼠标和我的电脑的同学会非常不习惯。不要紧。我们慢慢来。

2. 查看当前路径: pwd

输入 `$ pwd` 就可以知道当前位置了。

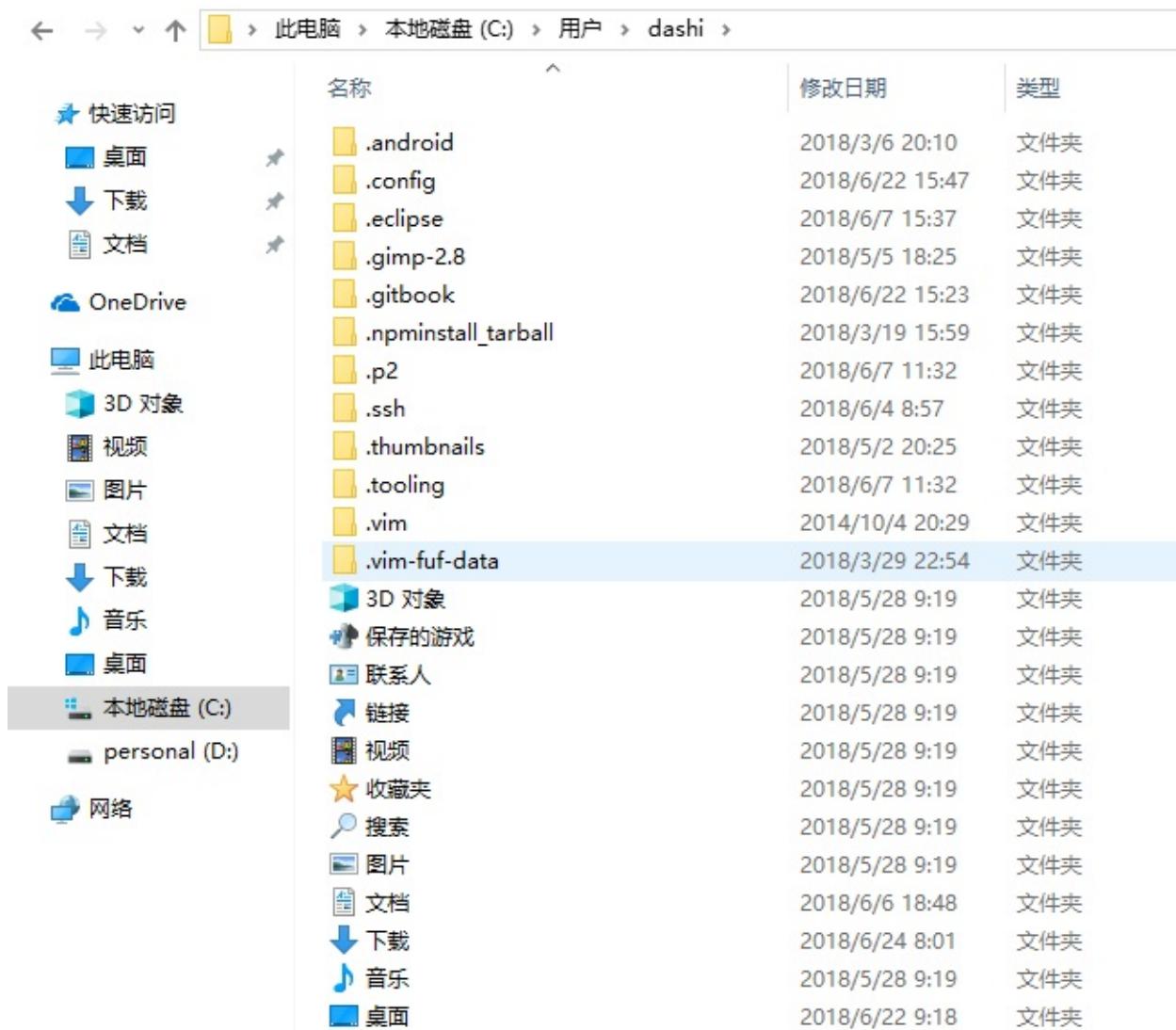
1. dashi@i5-16g MINGW64 ~
2. \$ pwd
3. /c/Users/dashi

在上面的结果中可以看到，

- `dashi` 是我的window 系统的用户名 . (我的外号叫大师。)
- `i5-16g` 是我的电脑在局域网的名字
- `MINGW64` 是操作系统的名字。可以认为它是跟 Linux, Windows, Mac之外的第四种操作系统。
- `$` 是命令行的前缀。后面的 `pwd` 就是我们输入的命令。
- `/c/Users/dashi` 就是当前位置。这个是 Linux风格。实际上它对应的 windows的标准路径是：`C:\Users\dashi`

git bash 每次打开的时候，都是默认的这个“当前用户在windows”中的用户文件夹。

我们在一个窗口中打开这个路径的话，就可以看到我的用户文件夹了。如下图所示：



可以看到，我输入的路径是：`C:\Users\dashi`，结果在GUI中显示的文字是：“> 此电脑 > 本地磁盘(C:) > 用户 > dashi”

3. 切换路径: cd

例如，我想进入到我的工作目录(位于 `D:\workspace\happy_book_vuejs`)，继续写我的vuejs书，就可以这样：

```

1. dashi@i5-16g MINGW64 ~
2. $ cd /d/workspace/happy_book_vuejs/
3.
4. dashi@i5-16g MINGW64 /d/workspace/happy_book_vuejs (master)
5. $

```

所以，大家可以看到，`D:\` 在 Git bash 中对应的地址是：`/d`

这个就是唯一需要注意的点了。

其他的git 的基本知识(`git clone`，`git commit`，`git push` 等) 就不在本书中赘述

了。 对于没有掌握的同学，一定要赶紧学习这个技术。

Webpack

随着 SPA (Single Page App) 单页应用的发展，大家发现，使用的js/css/png等文件特别多。难以管理。文件夹结构很容易混乱。

而且很多时候我们希望项目可以具备： 压缩css， 压缩js， 正确的处理各种js/css的import， 以及相关的模板html文件.

在最初的一段时间里，可以说每个SPA项目发展到一定规模，就会遇到这样的瓶颈和痛点。

为了解决这个问题，就出现了Webpack .

官方网站: <https://webpack.js.org/> github:

<https://github.com/webpack/webpack>， 截止2018-6-25， 关注数是 41918 .

webpack website

它是一个打包工具，可以把 js, css, node module, coffeescript, scss/less，图片等等都打包在一起。这个简直是模块化开发SPA福音！所以现在几乎所有的SPA项目，所有的JS项目，都会用到Webpack。

我们在前面的入门中，看到了Vue.js只需要引入一个外部的js文件就可以工作。不过在我们的实际开发中，情况复杂了很多倍。我们都是统一使用Webpack + Vue.js的方式来做项目的。这样才可以做到“视图”，“路由”，“component”等等的分离，以及快速的打包，部署，项目上线。

功能

它的功能非常强大，对各种技术都提供了支持，仿佛是一个万能胶水，把所有的技术都结合到了一起：

1. 对文件的支持

- 支持普通文件
- 支持代码文件
- 支持文件转url（支持图片）

2. 对JSON的支持

- 支持普通JSON
- 支持JSON5
- 支持CSON

3. 对JS 预处理器的支持

- 支持普通javascript
- 支持Babel（来使用 ES2015+）
- 支持Traceur（来使用 ES2015+）
- 支持TypeScript
- 支持Coffeescript

4. 对模板的支持

- 支持普通HTML
- 支持Pug 模板
- 支持JADE 模板
- 支持Markdown 模板
- 支持PostHTML
- 支持Handlebars

5. 对Style的支持

- 支持普通style
- 支持import
- 支持LESS
- 支持SASS/SCSS
- 支持Stylus
- 支持PostCSS

6. 对各种框架的支持

- 支持Vue.js

- 支持Angular2
- 支持Riot

安装方式

```
1. $ npm install --save-dev webpack
```

使用

由于Webpack自身是支持Vue.js的，Webpack 跟 Vue.js 已经结合到很难分清谁是谁，我们就索性不区分。 知道做什么事儿需要运行什么命令就可以了。

所以我们不必专门来单独学习Webpack，了解Webpack的概念就可以。

在接下来的教程中，同学会看到Webpack + Vue.js 共同开发的方法和步骤。

开发环境的搭建

npm: 3.10.8 (npm > 3.0)

node: v6.9.1 (node > 6.0)

vue: 2.0+

总的来说，只要能安装上node 和 vuejs就可以。

安装NVM 和 node

请看对应章节。

安装git

请看对应章节。

安装vuejs

要同时安装 `vue` 和 `vue-cli` 这两个node package.

运行下面这个命令：

```
1. $ npm install vue vue-cli -g
```

`-g` 表示这个包安装后可以被全局使用。

运行 vue

创建一个基于 webpack 模板的新项目：

```
1. $ vue init webpack my-project
```

注意： 我们使用Vue， 都是在 `webpack` 这个大前提下使用的。

安装依赖：

```
1. $ cd my-project  
2. $ cnpm install
```

在本地，以默认端口来运行：

```
1. $ npm run dev
```

然后就可以看到 在本地已经跑起来了。

```
1. > test_vue_0613@1.0.0 dev /workspace/test_vue_0613
2. > node build/dev-server.js
3.
4. > Starting dev server...
5.
6.
7. DONE Compiled successfully in 12611ms    12:16:47 PM
8.
9. > Listening at http://localhost:8080
```

我们打开 <http://localhost:8080> 就可以看到刚才创建的项目欢迎页，如下图所示：



Hello World!

Welcome to your Vue.js app!

To get a better understanding of how this boilerplate works, check out [its documentation](#). It is also recommended to go through the docs for [Webpack](#) and [vue-loader](#). If you have any issues with the setup, please file an issue at this boilerplate's [repository](#).

You may also want to checkout [vue-router](#) for routing and [vuex](#) for state management.

Webpack下的Vuejs项目文件结构

根据前面章节，我们安装了 webpack, vue-cli，并运行成功，看到了Vuejs的第一个页面。

那么首先，我们需要对Webpack 下的Vuejs有个全面的了解。

在我们运行了命令后，

```
1. $ vue init webpack my-project
```

会生成一个崭新的项目。 它的文件结构如下：

```
1. ▶ build/          // 编译用到的脚本  
2. ▶ config/         // 各种配置  
3. ▶ dist/           // 打包后的文件夹  
4. ▶ node_modules/   // node第三方包  
5. ▶ src/            // 源代码  
6. ▶ static/          // 静态文件，暂时无用  
7. index.html        // 最外层文件  
8. package.json      // node项目配置文件
```

下面我们针对重要的文件和文件夹来依次说明。

build 文件夹

这个文件夹中，保留了各种打包脚本。 是不可或缺的，不要随意修改。

展开后如下：

```
1. ▼ build/  
2.     build.js  
3.     check-versions.js  
4.     dev-client.js  
5.     dev-server.js  
6.     utils.js  
7.     vue-loader.conf.js  
8.     webpack.base.conf.js  
9.     webpack.dev.conf.js  
10.    webpack.prod.conf.js
```

- build.js:

打包使用，不要修改。

- check-versions.js:

检查npm的版本，不要修改。

- dev-client.js 和 dev-server.js:

是在开发时使用的服务器脚本。不要修改。（借助于node这个后端语言，我们在做vuejs开发时，可以通过 `$npm run dev` 这个命令，打开一个小的server，运行vuejs.）

- utils.js

不要修改。做一些css/sass 等文件的生成。

- vue-loader.conf.js

非常重要的配置文件，不要修改。内容是用来辅助加载vuejs用到的css source map等内容。

- webpack.base.conf.js
- webpack.dev.conf.js
- webpack.prod.conf.js

这三个都是基本的配置文件。不要修改。

config 文件夹

跟部署和配置相关。

```
1.  ▼ config/
2.      dev.env.js
3.      index.js
4.      prod.env.js
```

- dev.env.js

开发模式下的配置文件，一般不用修改。

- prod.env.js

生产模式下的配置文件，一般不用修改。

- index.js

很重要的文件， 定义了：

- 开发时的端口（默认是8080），
- 图片文件夹（默认static），
- 开发模式下的 代理服务器。我们修改的还是比较多的。

对于这个文件夹的内容，我们会在后续的章节中陆续进行解释（例如对于某个页面的渲染过程，如何做代理转发）

dist 文件夹

打包之后的文件所在目录，如下。

```

1.  ▼ dist/
2.    ▼ static/
3.      ▼ css/
4.          app.d41d8cd98f00b204e9800998ecf8427e.css
5.          app.d41d8cd98f00b204e9800998ecf8427e.css.map
6.      ▼ js/
7.          app.c482246388114c3b9cf0.js
8.          app.c482246388114c3b9cf0.js.map
9.          manifest.577e472792d533aaaf04.js
10.         manifest.577e472792d533aaaf04.js.map
11.         vendor.5f34d51c868c93d3fb31.js
12.         vendor.5f34d51c868c93d3fb31.js.map
13.         index.html

```

可以看到，对应的css， js， map， 都在这里。

请大家注意 文件名中的无意义的字符串，这个是随机生成的。 目的是为了让文件名发生变化， 方便我们的部署，也方便Nginx服务器重新对该文件做缓存。

- `app.css` : 编译后 的CSS 文件
- `app.js` : 最核心的js 文件。几乎所有的代码逻辑都会打包到这里。
- `manifest.js` : 生成的周边js 文件
- `vendor.js` : 生成的`vendor.js` 文件

另外，每个 `.map` 文件都非常重要。 可以简单的认为，有了 `.map` 文件，我们的浏览器就可以先下载整个的 `.js` 文件，然后在后续的操作中“部分加载” 对应的文件。

切记： 这个文件夹不要放到git中。 因为每次编译之后，这里的文件都会发生变化。

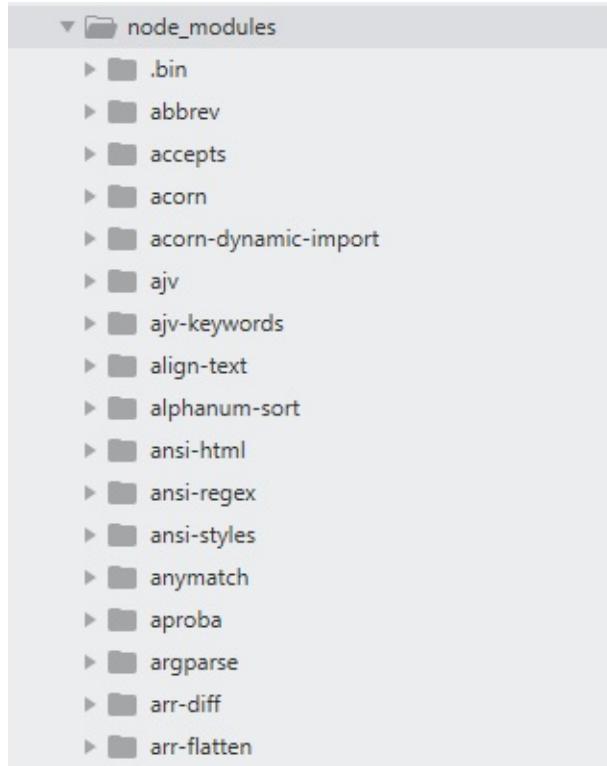
node_modules 文件夹

node项目所用到的第三方包，特别多，特别大。 \$ `npm install` 所产生。所有在 `package.json` 中定义的第三方包都会出现在这里。

`package.json` :

```
1.   // ...
2.   "dependencies": {
3.     "vue": "^2.3.3",
4.     "vue-resource": "1.3.3",
5.     "vue-router": "^2.3.1",
6.     "vuex": "^2.3.1"
7.   },
8.   "devDependencies": {
9.     "autoprefixer": "^6.7.2",
10.    "babel-core": "^6.22.1",
11.    "babel-loader": "^6.2.10",
12.    "babel-plugin-transform-runtime": "^6.22.0",
13.    "babel-preset-env": "^1.3.2",
14.    "babel-preset-stage-2": "^6.22.0",
15.    "babel-register": "^6.22.0",
16.    //...
17.  }
18.  // ...
```

`node_modules` 文件夹里面往往会有几百个文件夹，看起来如下：



这个文件夹不要放到git中。

src 文件夹

最最核心的源代码所在的目录。 展开后如下所示（不同版本的vue-cli生成的目录会稍有不同，不过核心都是一样的）：

```
1.  ▼ src/
2.    ▼ assets/
3.      logo.png
4.    ▼ components/
5.      Book.vue
6.      BookList.vue
7.      Hello.vue
8.    ▼ router/
9.      index.js
10.     App.vue
11.     main.js
```

- assets 文件夹

用到的图片都可以放在这里。

- components

用到的“视图”和“组件”所在的文件夹。（最核心）

- router/index.js

路由文件。 定义了各个页面对应的url.

- App.vue

如果index.html 是一级页面模板的话，这个App.vue就是二级页面模板。所有的其他vuejs页面，都作为该模板的一部分被渲染出来。

- main.js

没有实际的业务逻辑，但是为了支撑整个vuejs框架，存在很必要。

创建一个页面

激动人心的时刻到了！ 接下来我们要通过自己动手，开始下一步的学习！

请各位同学务必准备好电脑。 只有一边学习一边敲代码，才能真正的看到效果。 调试代码的过程是无法脑补出来的。切记切记！

在Vue.js中创建页面需要两步：

1. 新建路由
2. 新建vue页面

新建路由

默认的路由文件是 `src/router/index.js`，打开之后，我们增加两行：

```
1. import Vue from 'vue'  
2. import Router from 'vue-router'  
3.  
4. // 增加这一行，作用是引入 SayHi 这个component  
5. import SayHi from '@/components/SayHi'  
6.  
7. Vue.use(Router)  
8. export default new Router({  
9.   routes: [  
10.     // 增加下面几行，表示定义了 /#/say_hi 这个url  
11.     {  
12.       path: '/say_hi',  
13.       name: 'SayHi',  
14.       component: SayHi  
15.     },  
16.   ],  
17. } )  
18. })
```

上面的代码中，

```
1. import SayHi from '@/components/SayHi'
```

表示从 当前目录下的 `components`，引入 `SayHi` 文件。`@` 表示当前目录。

然后，用下面的代码来定义一个路由：

```
1. routes: [
2.   {
3.     path: '/say_hi',    // 对应 一个url
4.     name: 'SayHi',      // Vuejs 内部使用的名称
5.     component: SayHi  // 对应的 .vue 页面的名字
6.   },
7. ]
```

也就说，每当用户的浏览器访问 http://localhost:8080/#/say_hi 时，就会渲染 `./components/SayHi.vue` 文件。

`name: 'SayHi'` 定义了该路由在Vuejs内部的名称。后面会对此有所解释。

创建一个新的Component

由于上面我们在路由中引用了component: `src/components/SayHi.vue`，接下来，我们要创建这个文件。

代码如下：

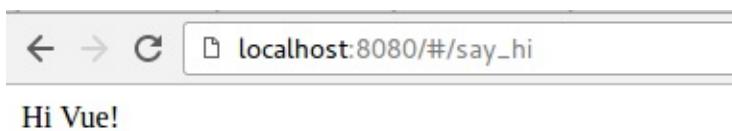
```
1. <template>
2.   <div >
3.     Hi Vue!
4.   </div>
5. </template>
6.
7. <script>
8. export default {
9.   data () {
10.     return { }
11.   }
12. }
13. </script>
14.
15. <style>
16. </style>
```

在上面的代码中：

- `<template></template>` 代码块中，表示的是 HTML模板。里面写的就是最普通的HTML。

- `<script/>` 表示的是 js 代码. 所有的js代码都写在这里. 这里使用的是 EM Script.
- `<style>` 表示所有的 CSS/SCSS/SASS 文件都可以写在这里.

现在, 我们可以直接访问 http://localhost:8080/#/say_hi 这个链接了. 打开后, 页面如下图所示:



为页面增加一些样式

接下来, 我们可以为页面加一些样式, 让它变得好看一些:

```
1. <template>
2.   <div class='hi'>
3.     Hi Vue!
4.   </div>
5. </template>
6.
7. <script>
8. export default {
9.   data () {
10.     return { }
11.   }
12. }
13. </script>
14.
15. <style>
16. .hi {
17.   color: red;
18.   font-size: 20px;
19. }
20. </style>
```

注意上面代码中的 `<style>` 标签, 里面跟普通的CSS一样, 定义了样式。

```
1. .hi {  
2.   color: red;  
3.   font-size: 20px;  
4. }
```

刷新浏览器，可以看到，文字加了颜色，如下图所示：



定义并显示变量

如果要在vue页面中定义一个变量，并把它显示出来，就需要事先在 `data` 中定义：

```
1. export default {  
2.   data () {  
3.     return {  
4.       message: '你好Vue！本消息来自于变量'  
5.     }  
6.   }  
7. }
```

可以看到，上面的代码是通过 Webpack的项目来写的。 回忆一下， 原始的Vuejs中是如何定义一个变量(property) 的？

答案是：

```
1. var app = new Vue({  
2.   data () {  
3.     return {  
4.       message: '你好Vue！本消息来自于变量'  
5.     }  
6.   }  
7. })
```

所以，我们可以认为，之前用“原始的Vuejs”的代码中，写的 `new Vue({...})` 中的代码，在

webpack的框架下，都要写到 `export default { .. }` 中来。

完整的代码（`src/components/SayHi.vue`）如下所示：

```
1. <template>
2.   <div>
3.     <!-- 步骤2：在这里显示 message -->
4.     {{message}}
5.   </div>
6. </template>
7.
8. <script>
9. export default {
10.   data () {
11.     return {
12.       // 步骤1：这里定义了变量 message 的初始值
13.       message: '你好Vue! 本消息来自于变量'
14.     }
15.   }
16. }
17. </script>
18.
19. <style>
20. </style>
```

页面打开如下图所示：



Vuejs 中的 ECMAScript

对于稍微有一定编程经验的同学，会发现我们使用的不是“原生的javascript”，而是一种新的语言。这个语言就是 ECMAScript。

严格的说， ECMAScript是Javascript的规范， Javascript是ECMA的实现。

ECMAScript出了 javascript，还有Jscript 和 ActionScript这样的实现（也叫方言）

ECMAScript 也简称叫ES。 版本比较多， 有ES 2015, ES2016, ES2017 等。 很多时候我们用ES6来泛指这三个版本。 从这里： <http://kangax.github.io/compat-table/es6/> 可以看到， ES6 的90%的特性都已经被各大浏览器实现了。 阮一峰先生的《ECMAScript入门》中对ES的历史有着非常精彩的阐述。

具体的细节我们不去深究，我们就暂且认为ECMAScript实现了很多 普通js无法实现的功能。 同时，在Vuejs项目中大量的使用了ES的语法。

下面是一个极简版的ES6的入门。 大家只要可以看懂这些代码，就可以通读本书。

let, var, 常量 与全局变量

声明本地变量，使用 let 或者 var . 两者的区别是：

- var：有可能引起 变量提升，或者 块级作用域的问题.
- let：就是为了解决这两个问题存在的.

最佳实践：多用let，少用var. 遇到诡异变量问题的时候，就查查是不是var的问题.

下面是三个对比：

```
1. var title = '标题'; // 没问题
2. let title = '标题'; // 没问题
3. title = '标题'; // 这样做会报错.
```

记得，在 webpack下的 vuejs中， 使用任何变量，都要使用 var 或者let来声明.

常量：

```
1. const TITLE='标题';
```

对于全局变量，直接在 index.html 中声明即可. 例如：

```
1. window.title = '我的博客列表'
```

导入代码： import

import 用来引入导入外部代码. 如下所示:

```
1. import Vue from 'vue'  
2. import Router from 'vue-router'
```

上面的代码, 目的是引入 vue 和 vue-router (由于他们是在 package.json 中定义了, 所以可以直接 import ... from <包名> . 否则要加上路径)

```
1. import SayHi from '@/components/SayHi'
```

上面这个, 在from后面, 有 @ 符号, 表示是在本地文件系统中, 引入文件. @ 代表 源代码目录, 一般是 src.

在 @ 出现之前, 我们在编码的时候也会这样写:

```
1. import Swiper from '../components/swiper'  
2. import SwiperItem from '../components/swiper-item'  
3. import XHeader from '../components/header/x-header'  
4. import store from '../vuex/store'
```

大量使用了 ... 这样的代码, 会引起代码的混乱. 所以推荐使用 @ 方法.

方便其他代码来使用自己: export default {...}

我们会看到, 在每个 vue文件中的 <script> 中, 都会存在这个代码. 它的作用是方便其他代码对这个代码来引用.

对于Vuejs 程序员, 我们就记住这个写法就好了.

在ES6之前js没有一个统一的模块定义方式, 流行的定义方式有AMD, CommonJS等, 这些方式都是使用一种“打补丁”的形式来实现这个功能, 总给人一种怪怪的感觉. 而ES6从语言层面对定义模块的方式进行了统一。

假设有: lib/math.js 文件内容如下:

```
1. export function sum(x, y) {
```

```

2.   return x + y
3. }
4. export var pi = 3.141593

```

可以看到，`lib/math.js` 文件，可以导出两个东西，一个是 `function sum`，一个是 `var pi`。

我们可以定义一个新的文件，`app.js`，文件内容如下：

```

1. import * as math from "lib/math"
2. alert("2π = " + math.sum(math.pi, math.pi))

```

可以看到，在上面的代码中，可以直接调用 `math.sum` 和 `math.pi` 方法。

我们再看一个例子：新建一个文件 `other_app.js`，内容如下：

```

1. import {sum, pi} from "lib/math"
2. alert("2π = " + sum(pi, pi))

```

可以看到，上面的代码中，通过 `import {sum, pi} from "lib/math"`，可以在后面直接调用 `sum()` 和 `pi`

而 `export default { ... }` 则是暴露出一段没有名字的代码，（不像 `export function sum(a,b){ .. }` 这样有个名字（叫sum）。）

在webpack下的Vue.js，会自动对这些代码进行处理。都属于框架内的工作。各位同学只需要按照这个规则来写代码，就一定没问题。

ES中的简写

我们会发现，这样的代码：

```

1. <script>
2. export default {
3.   data () {
4.     return {}
5.   }
6. }
7. </script>

```

实际上，上面的代码是一种简写形式，它等同于下面的代码：

```

1. <script>
2. export default {
3.   data: function() {
4.     return { }
5.   }
6. }
7. </script>

```

箭头函数 =>

跟coffeescript一样，ES 也可以通过箭头来表示函数。

```
1. .then(response => ...);
```

等同于：

```

1. .then(function (response) {
2.   // ...
3. })

```

这样写法的好处，是强制定义了作用域。使用了 `=>` 之后，可以避免很多由于作用域产生的问题。建议大家多使用。

hash中同名的 key, value的简写

```

1. let title = 'triple body'
2.
3. return {
4.   title: title
5. }

```

等同于：

```

1. let title = 'triple body'
2.
3. return {
4.   title
5. }

```

分号可以省略

例如：

```
1. var a = 1
2. var b = 2
```

等同于：

```
1. var a = 1;
2. var b = 2;
```

解构赋值

我们先定义好一个hash：

```
1. let person = {
2.   firstname : "steve",
3.   lastname : "curry",
4.   age : 29,
5.   sex : "man"
6. };
```

然后，我们可以这样做定义：

```
1. let {firstname, lastname} = person
```

上面一行代码，等价于：

```
1. let firstname = person.firstname
2. let lastname = person.lastname
```

我们可以这样定义函数：

```
1. function greet({firstname, lastname}) {
2.   console.log(`hello, ${firstname}.${lastname}!`);
3. }
4.
5. greet({
6.   firstname: 'steve',
```

```
7.     lastname: 'curry'  
8. });

但是不建议大家这样使用。有一些奇淫技巧的感觉。另外，浏览器和一些第三方支持的不是太好，我们在实际项目中，曾经遇到过与之相关的很奇葩的问题。
```

学习资料

国内比较好的中文教材，是阮一峰编写的 <[>](#)>。书中非常详实的阐述了相关的内容，概念和用法。

我们也可以在网上直接查看这本书的电子版：<http://es6.ruanyifeng.com>

Vue.js 渲染页面的过程和原理

只有知道了一个页面是如何被渲染出来的，我们才可以更好的理解框架和调试代码。下面我们就来仔细看一下这个过程。

渲染过程1. js入口文件

最初的最初，我们要知道 `./build/webpack.base.conf.js` 这个文件，是webpack打包的主要配置文件。一个典型的代码如下：

```
1. var path = require('path')
2. var utils = require('./utils')
3. var config = require('../config')
4. var vueLoaderConfig = require('./vue-loader.conf')
5.
6. function resolve (dir) {
7.   return path.join(__dirname, '.', dir)
8. }
9.
10. module.exports = {
11.   entry: {
12.     app: './src/main.js'
13.   },
14.   output: {
15.     path: config.build.assetsRoot,
16.     filename: '[name].js',
17.     publicPath: process.env.NODE_ENV === 'production'
18.       ? config.build.assetsPublicPath
19.       : config.dev.assetsPublicPath
20.   },
21.   resolve: {
22.     extensions: ['.js', '.vue', '.json'],
23.     alias: {
24.       'vue$': 'vue/dist/vue.esm.js',
25.       '@': resolve('src')
26.     }
27.   },
28.   module: {
29.     rules: [
30.       {
```

```

31.      test: /\.vue$/,
32.      loader: 'vue-loader',
33.      options: vueLoaderConfig
34.    },
35.    {
36.      test: /\.js$/,
37.      loader: 'babel-loader',
38.      include: [resolve('src'), resolve('test')]
39.    },
40.    {
41.      test: /\.(png|jpe?g|gif|svg)(\?.*)?$/,
42.      loader: 'url-loader',
43.      options: {
44.        limit: 10000,
45.        name: utils.assetsPath('img/[name].[hash:7].[ext]')
46.      }
47.    },
48.    {
49.      test: /\.(woff2?|eot|ttf|otf)(\?.*)?$/,
50.      loader: 'url-loader',
51.      options: {
52.        limit: 10000,
53.        name: utils.assetsPath('fonts/[name].[hash:7].[ext]')
54.      }
55.    }
56.  ],
57. }
58. }

```

在上面的代码中，

```

1. module.exports = {
2.   entry : {
3.     app: './src/main.js' // 这里就定义了Vuejs的入口文件
4.   }
5. }

```

很重要， 其中的 `app: './src/main.js'` 就定义了Vuejs的入口文件

知道了这个打包文件，我们就可以知道接下来的事儿了。

渲染过程2. 静态的HTML页面： index.html

因为我们每次打开的都是 ‘<http://localhost/#/>’，实际上打开的文件是 ‘<http://localhost/#/index.html>’

所以找到 `index.html`，可以看到它的内容如下；

```

1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <meta charset="utf-8">
5.     <title>演示Vue的demo</title>
6.   </head>
7.   <body>
8.     <div id="app"></div>
9.   </body>
10. </html>
```

这里的 `<div id="app"></div>`，就是将来会动态变化的内容。特别类似于 Rails 的 layout（模板文件）。

这个 `index.html` 文件是最外层的 模板。

渲染过程3. `main.js` 中的 Vue 定义

我们回头看 `src/main.js`，它的内容如下；

```

1. import Vue from 'vue'
2. import App from './App'
3. import router from './router'
4. import VueResource from 'vue-resource';
5. Vue.use(VueResource);
6.
7. Vue.config.productionTip = false
8. Vue.http.options.emulateJSON = true;
9.
10. new Vue({
11.   el: '#app', // 这里，#app 对应着 <div id=app></div>
12.   router,
13.   template: '<App/>',
14.   components: { App } // 这里，App就是 ./src/App.vue
15. })
```

熟悉 jquery, css selector的同学可以看到， el: '#app' 就是 index.html 中的

```
<div id= app>
```

注意上面的 App.vue , 它会被 main.js 加载， App.vue的内容如下：

```
1. <template>
2.   <div id="app">
3.     <router-view class="view"></router-view>
4.   </div>
5. </template>
6. <script>
7. </script>
8. <style>
9. </style>
```

上面代码中的 <template/> , 这个就是第二层的模板。 可以认为该页面的内容，就是在这个位置被渲染出来的。

上面的代码中，还有个 <div id = 'app'>...</div> , 这个元素跟 index.html 中的是同一个元素（暂且这么理解）

所有的 <router-view> 中的内容，都会被自动替换。

<script> 中的代码则是脚本代码。 至此，整个过程就出来了。

原理

Vuejs 就是最典型的Ajax 工作方式：只渲染部分页面。

浏览器的页面从来不会整体刷新。所有的页面变化都限定在 index.html 中的 <div id="app"> </div> 代码中，如下图所示：

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <meta charset="utf-8">
5.     <title>演示Vue的demo</title>
6.   </head>
7.   <body>
8.     <div id="app"></div>    <!-- 都在这一行 -->
9.   </body>
10.  </html>
```

所有的动作, 可以靠 url 来触发, 例如:

- `/#/books_list` 对应 某个列表页
- `/#/books/3` 对应某个详情页

这个技术, 就是靠 vuejs 的核心组件 `vue-router` 来实现的。

不使用router的技术: QQ邮箱

QQ邮箱是属于: url无法跟某个页面一一对应的项目.

所有页面的跳转,都无法根据url 来判断

最大的特点: 无法保存页面的状态, 难以调试, 无法根据url直接进入某个页面。

视图中的渲染

前面我们介绍了项目的运行 (hello world), 文件夹的结构, 以及index.html中的内容是如何一点点的渲染出来的。下面, 我们来学习下Vue.js中对于视图的操作。

渲染某个变量

假定我们定义了一个变量:

```
1. <script>
2. export default {
3.   data () {
4.     return {
5.       my_value: '默认值',
6.     }
7.   },
8. }
9. </script>
```

我们就可以这样来显示它:

```
1. <div>{{my_value}}</div>
```

完整代码如下: ([src/components/Hello.vue](#))

```
1. <template>
2.   <div>
3.     {{message}}
4.   </div>
5. </template>
6.
7. <script>
8. export default {
9.   data () {
10.     return {
11.       message: '你好Vue! 本消息来自于变量'
12.     }
13.   }
14. }
15. </script>
```

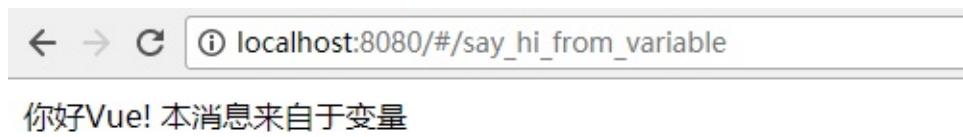
```

16.
17. <style>
18. </style>

```

可以看到，上面的代码显示定义了 `message` 这个变量，然后把它在 `<h1> {{ message }}` 中显示出来。

打开浏览器 `http://localhost:8080/#/say_hi_from_variable`，就可以看到下图所示：



方法的声明和调用

声明一个方法：`show_my_value`

```

1. <script>
2. export default {
3.   data () {
4.     return {
5.       my_value: '默认值',
6.     }
7.   },
8.   methods: {
9.     show_my_value: function(){
10.       // 注意下面的 this.my_value, 要用到this这个关键字.
11.       alert('my_value: ' + this.my_value);
12.     },
13.   }
14. }
15. </script>

```

调用上面的方法

```

1. <template>
2. <div>
3.   <input type='button' @click="show_my_value()" value='...' />

```

```

4.    </div>
5.  </template>

```

对于有参数的方法，直接传递参数就好了。例如：

```

1.  <template>
2.  <div>
3.    <input type='button' @click="say_hi('Jim')" value='...'/>
4.  </div>
5. </template>
6. <script>
7. export default {
8.   data () {
9.     return {
10.       my_value: '默认值',
11.     }
12.   },
13.   methods: {
14.     say_hi: function(name){
15.       alert('hi, ' + name)
16.     },
17.   }
18. }
19. </script>

```

上面代码中，

```
1. <input type='button' @click="say_hi('Jim')" value='...'/>
```

就会调用 `say_hi` 这个方法，传入参数 ‘Jim’。

事件处理： v-on

我们看到，很多时候，`@click` 等同于 `v-on:click`，下面两个是一样的：

```

1. <input type='button' @click="say_hi('Jim')" value='...'/>
2. <input type='button' v-on:click="say_hi('Jim')" value='...'/>

```

视图中的Directive（指令）

我们在学习java的时候，知道有jsp 页面， 对于.net语言，有 .asp, aspx页面， 对于ruby，有erb这样的页面。

在Vuejs中，我们也有类似的编程能力。

但是由于 Vuejs 是一种框架，所以它的稍微特殊一些，只能与标签做结合使用。 叫做Directive. (指令)

我们之前看到的 `v-on` , `v-bind` , `v-if` , `v-for` 等等，只要是以 `v` 开头的，都是 Directive.

原理：

1. 用户在浏览器中输入网址，回车。
2. 浏览器加载所有的资源（js, html内容。） 此时尚未解析。
3. 浏览器加载Vuejs
4. Vuejs程序被执行，发现 页面中的 Directive，进行相关的解析
5. HTML中的内容被替换。 展现给用户。

所以，我们在开发一个Vuejs项目的时候，会看到大量的Directive。 这里的基础务必打好。

前提： 在directive中要使用表达式(Expression)

- 表达式： `a > 1` （有效）
- 普通的语句： `a = 1` ; （这是个声明，不会生效）
- 流控制语句： `return a;` （不会生效）

在所有的Directive中，都只能使用表达式。

正确的：

```
1.  
2. <div v-if="a > 100">  
3. </div>
```

错误的：

```
1. <div v-if="return false">  
2. </div>
```

循环：v-for

完整的例子如下：

```

1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.     <p>Vue.js周边的技术生态有： <p>
8.     <br/>
9.     <ul>
10.       <li v-for="tech in technologies">
11.         {{ tech }}
12.       </li>
13.     </ul>
14.   </div>
15.   <script>
16.     var app = new Vue({
17.       el: '#app',
18.       data: {
19.         technologies: [
20.           "nvm", "npm", "node", "webpack", "ecma_script"
21.         ]
22.       }
23.     })
24.   </script>
25. </body>
26. </html>

```

可以注意到，上面代码中的 `technologies` 是在 `data` 中被定义的：

```

1. data: {
2.   technologies: [
3.     "nvm", "npm", "node", "webpack", "ecma_script"
4.   ]
5. }

```

然后在下面代码中，`technologies` 被循环显示：

```

1. <li v-for="tech in technologies">
2.   {{tech}}
3. </li>

```

用浏览器打开上面代码后，如下图所示：



Vuejs周边的技术生态有：

- nvm
- npm
- node
- webpack
- ecma_script

判断：v-if

条件Directive，是由 `v-if` , `v-else-if` , `v-else` 配合完成的。下面是个完整的例子：

```

1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.     <p>我们要使用的技术是：<p>
8.
9.     <div v-if="name === 'vuejs'">
10.       Vuejs !
11.     </div>
12.     <div v-else-if="name === 'angular'">
13.       Angular
14.     </div>
15.     <div v-else>
16.       React
17.     </div>
18.   </div>

```

```

19.   <script>
20.     var app = new Vue({
21.       el: '#app',
22.       data: {
23.         name: 'vuejs'
24.       }
25.     })
26.   </script>
27. </body>
28. </html>

```

注意,

上面的代码中，`v-if` 后面的引号中，是 `name === 'vuejs'`，`==` 是 ECMAScript 的语言，表示严格的判断。（由于 JS 的 `==` 有先天的缺陷，所以我们在 95% 的情况下，都是用三个等号的形式）

用浏览器打开上面代码后，如下图所示：



我们要使用的技术是：

Vuejs！

v-if 与 v-for 的优先级

当 `v-if` 与 `v-for` 一起使用时，`v-for` 具有比 `v-if` 更高的优先级。也就是说，Vue.js 会先执行 `v-for`，再执行 `v-if`。

下面是个完整的例子：

```

1.  <html>
2.  <head>
3.    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4.  </head>
5.  <body>
6.    <div id='app'>

```

```

7.      <p> 全部的技术是： {{technologies}} </p>
8.      <p> v-for 与 v-if 的结合使用，只打印出 以"n"开头的技术： <p>
9.      <ul>
10.         <li v-for="tech in technologies" v-if="tech.indexOf('n') === 0">
11.             {% raw %}{{% endraw %}tech}
12.         </li>
13.     </ul>
14.   </div>
15.   <script>
16.     var app = new Vue({
17.       el: '#app',
18.       data: {
19.         technologies: [
20.           "nvm", "npm", "node", "webpack", "ecma_script"
21.         ]
22.       }
23.     })
24.   </script>
25. </body>
26. </html>

```

可以看到，在下面的代码中，`v-if` 与 `v-for` 结合使用了，先是做了循环 `tech in technologies`，然后对当前的 循环对象 `tech` 做了判断：

```

1. <li v-for="tech in technologies" v-if="tech.indexOf('n') === 0">
2.     {% raw %}{{% endraw %}tech}
3. </li>

```

用浏览器打开上面代码后，如下图所示：



全部的技术是： ["nvm", "npm", "node", "webpack", "ecma_script"]

v-for 与 v-if 的结合使用，只打印出 以"n"开头的技术：

- nvm
- npm
- node

v-bind

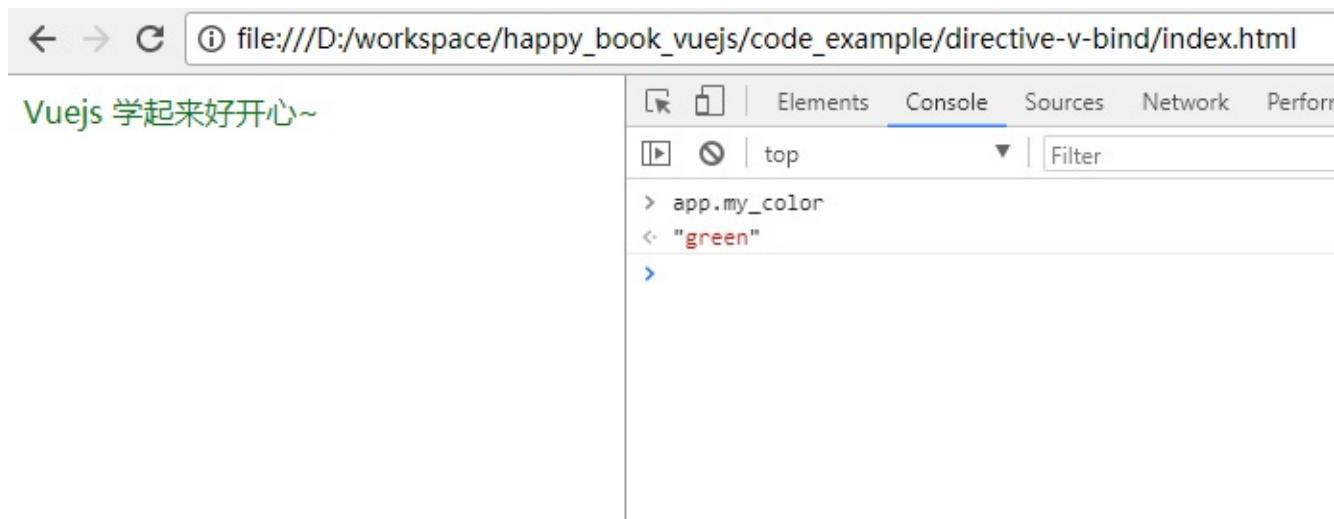
v-bind 指令用于把某个属性绑定到对应的元素的属性。 请看例子：

```

1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.     <p v-bind:style="'color:' + my_color">Vuejs 学起来好开心~ </p>
8.   </div>
9.   <script>
10.    var app = new Vue({
11.      el: '#app',
12.      data: {
13.        my_color: 'green'
14.      }
15.    })
16.   </script>
17. </body>
18. </html>
```

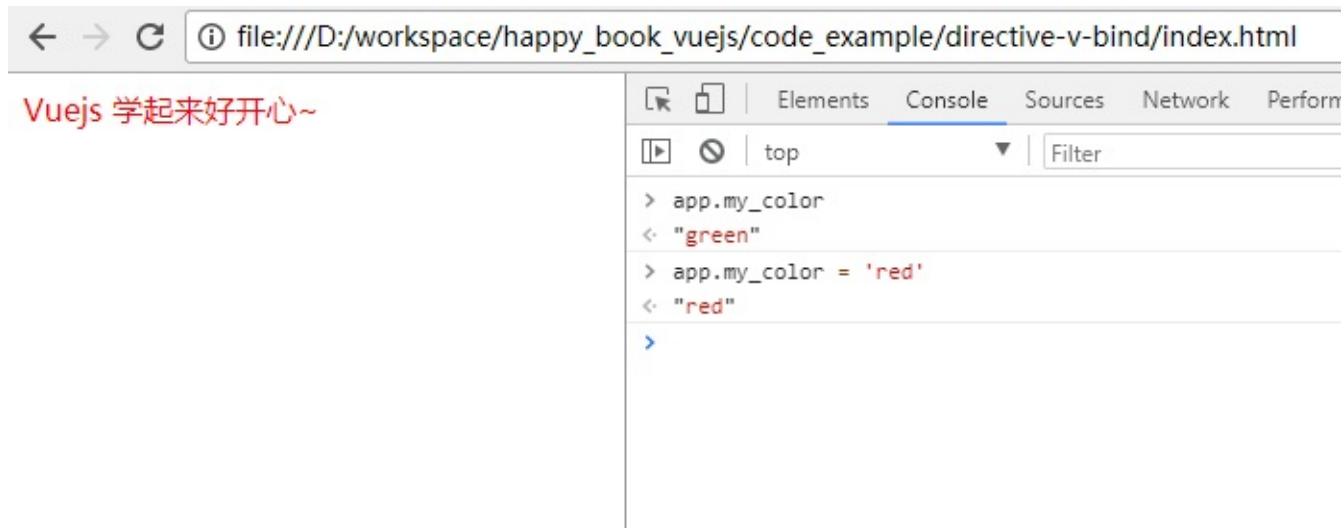
在上面的代码中，可以看到， 通过 **v-bind**， 把 **<p>** 元素的style的值， 绑定成了 **'color: ' + my_color** 这个表达式。当 **my_color** 的值发生变化的时候， 对应 **<p>** 的颜色就会发生变化。

例如： 默认页面打开后， 是绿色的。 如下图所示：



如何知道变量 **my_color** 已经绑定到了 **<p>** 上了呢？ 我们在 **console**中做修改， 让

`app.my_color = "red"`，就可以看到对应的文字的颜色，变成了红色，如下图所示：



对于所有的属性，都可以使用 `v-bind`，例如：

```
1. <div v-bind:style='...>
2. </div>
```

会生成：

```
1. <div style='...> </div>
```

```
1. <div v-bind:class='...> </div>
```

会生成：

```
1. <div class='...> </div>
```

```
1. <div v-bind:id='...> </div>
```

会生成：

```
1. <div id='...> </div>
```

V-on

`v-on` 指令用于触发事件。例如：

```

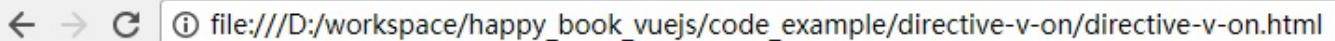
1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.     {{ message }}
8.     <br/>
9.     <button v-on:click='highlight' style='margin-top: 50px'>真的吗</button>
10.  </div>
11.
12.  <script>
13.    var app = new Vue({
14.      el: '#app',
15.      data: {
16.        message: '学习Vuejs使我快乐~'
17.      },
18.      methods: {
19.        highlight: function() {
20.          this.message = this.message + '是的，工资还会涨~!'
21.        }
22.      }
23.    })
24.  </script>
25. </body>
26. </html>

```

上面的代码中，通过 `v-on:click` 的声明，当

被点击(click)后，就会触发 `highlight` 这个方法。

点击前的页面：

① file:///D:/workspace/happy_book_vuejs/code_example/directive-v-on/directive-v-on.html

学习Vuejs使我快乐~

真的吗

点击后的页面：

① file:///D:/workspace/happy_book_vuejs/code_example/directive-v-on/directive-v-on.html

学习Vuejs使我快乐~ 是的， 工资还会涨~!

真的吗

v-on 后面可以接HTML的标准事件， 例如：

- click (单击鼠标左键)
- dblclick (双击鼠标左键)
- contextmenu (单机鼠标右键)
- mouseover (指针移到有事件监听的元素或者它的子元素内)
- mouseout (指针移出元素，或者移到它的子元素上)
- keydown (键盘动作：按下任意键)
- keyup (键盘动作：释放任意键)

对于 **v-on** 的更多说明，请看“Event”的对应章节。

v-on的简写

v-on:click 往往会写成 **@click**， **v-on:dblclick** 也会写成 **@dblclick**，所以同学们看代码的时候要了解~

v-model 与双向绑定

v-model 往往用来做“双向绑定”(two way binding)，这个双向绑定的含义是：

1. 可以通过表单（用户手动输入的值）来修改某个变量的值
2. 可以通过程序的运算来修改某个变量的值，并且影响页面的展示。

双向绑定可以大大的方便我们的开发。例如，我们做一个天气预报的软件时，需要在前台展示“当前温度”，如果后台代码做了某些操作后，发现“当前温度”发生了变化，

如果没有双向绑定这个技术，我们的代码就会比较膨胀，做各种条件判断。有了双向绑定的话，就可以做到后台变量一变化，前台的对于该变量的展示就会发生变化。

下面是完整的页面源代码：

```

1.                                     <html>
2.                                     <head>
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js">
3.                                     </script>
4.                                     </head>
5.                                     <body>
6.                                     <div id='app'>
```

```

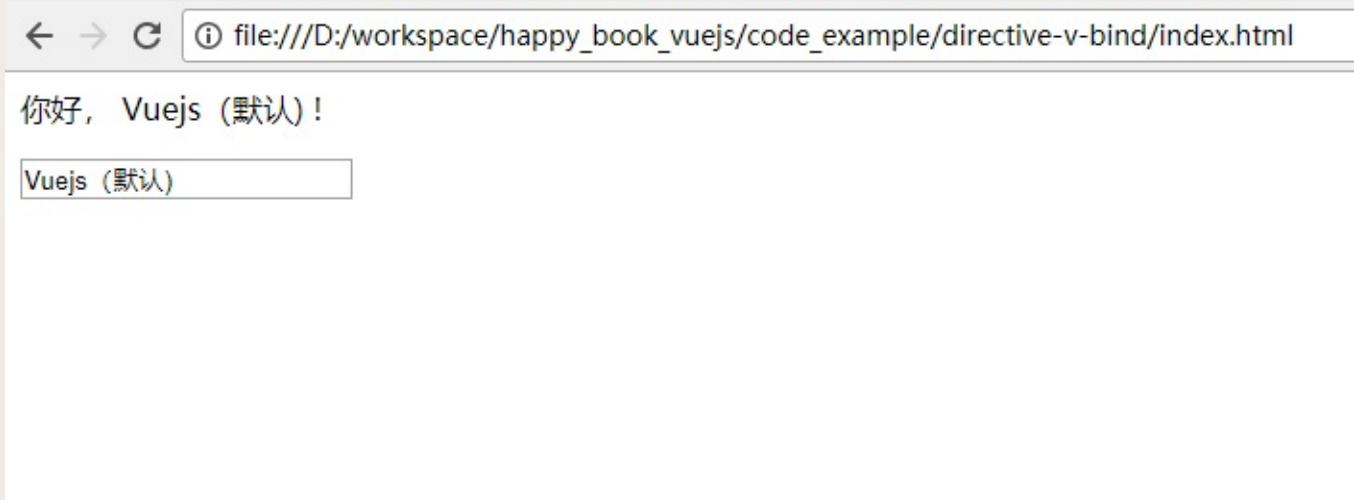
7.          <p> 你好,  {{ raw }}{{ name }} ! </p>
8.          <input type='text' v-model="name" />
9.        </div>
10.       <script>
11.         var app = new Vue({
12.           el: '#app',
13.           data: {
14.             // 下面一行代码不能省略。 这里声明了 name 这个变量
15.             name: 'Vuejs (默认)'
16.           }
17.         })
18.       </script>
19.     </body>
20.   </html>

```

上面的代码中，可以看到，我们：

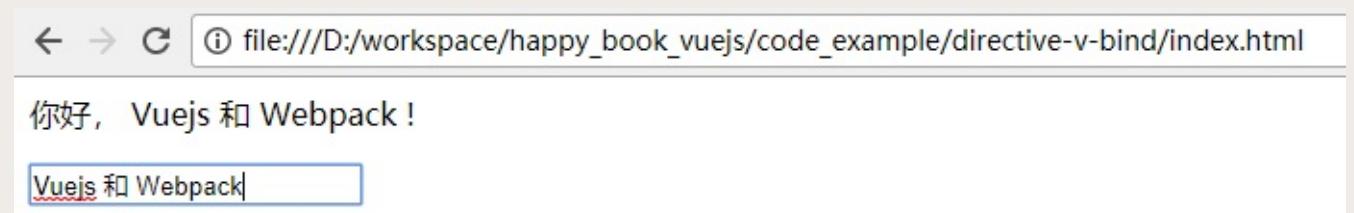
1. 使用了 `<input type='text' v-model="name" />` 来把变量 `name` 绑定在了 `<input>` 这个输入框上（可以在那里看到 `name` 的值）
2. 使用了 `<p> 你好, {{ raw }}{{ name }} ! </p>` 来把变量 `name` 也显示在页面上。
3. 在初始化中，使用 `data: { name: '...' }` 的方式，对变量 `name` 进行了初始化。

使用浏览器打开该html页面，可以看到，最初的状态如下图所示：



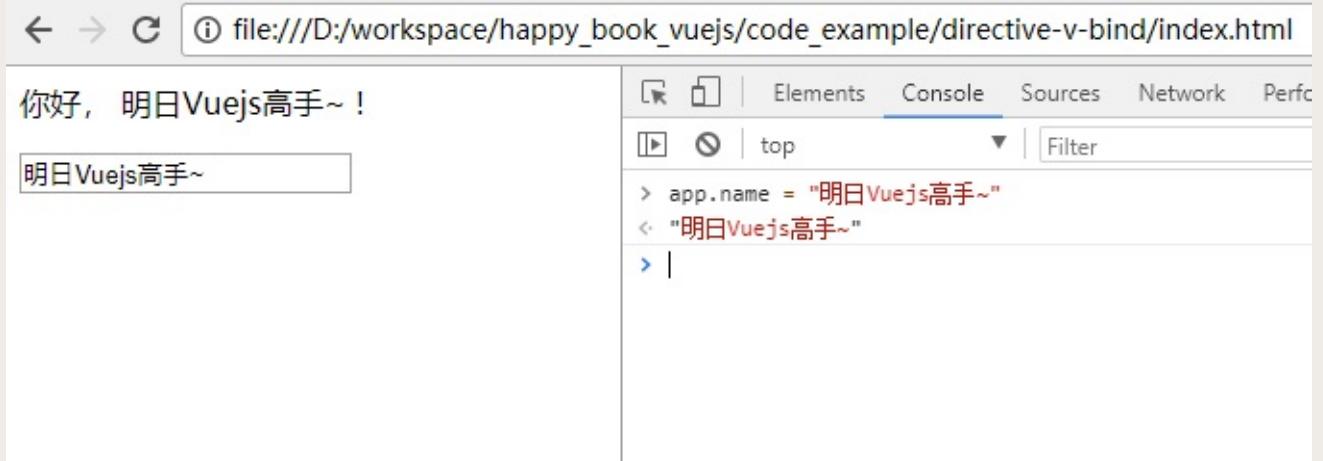
可以看到，上图中的文字显示的是：“你好，Vuejs！”

然后，我们在输入框中，把内容改成：“Vuejs 和 Webpack”，于是页面就发生了变化，如下图所示：



这样就说明，我们通过输入框来改变某个变量的值，是成功的。

然后，我们打开浏览器的“开发者工具”（建议用Chrome, Chrome下的操作方式是：按F12）。在console中输入：`app.name = "明日Vuejs高手"`，就会看到下图所示：



这样就说明，我们通过运算来改变某个变量的值，是成功的。

不同页面间的参数的传递

在普通的web开发中，参数传递有以下几种形式：

1. url: `/another_page?id=3`
2. 表单: `<form>...</form>`

而在Vuejs中，不会产生表单的提交（这会引起页面的整体刷新）。所以有两种：

1. url . 同传统语言。参数体现在url中。
2. vuejs 内部的机制。（无法在url 中体现，可以认为是由js代码隐式实现的）

我们用一个实际的例子说明。

我们之前实现了“博客列表页”，接下来我们要实现：点击博客列表页中的某一行，就显示博客详情页。

1. 回顾：现有的接口

我已经做好了一个接口：文章详情页。地址为：

`/interface/blogs/show`

该接口接收一个参数：`id`。使用 `http GET` 请求来访问。

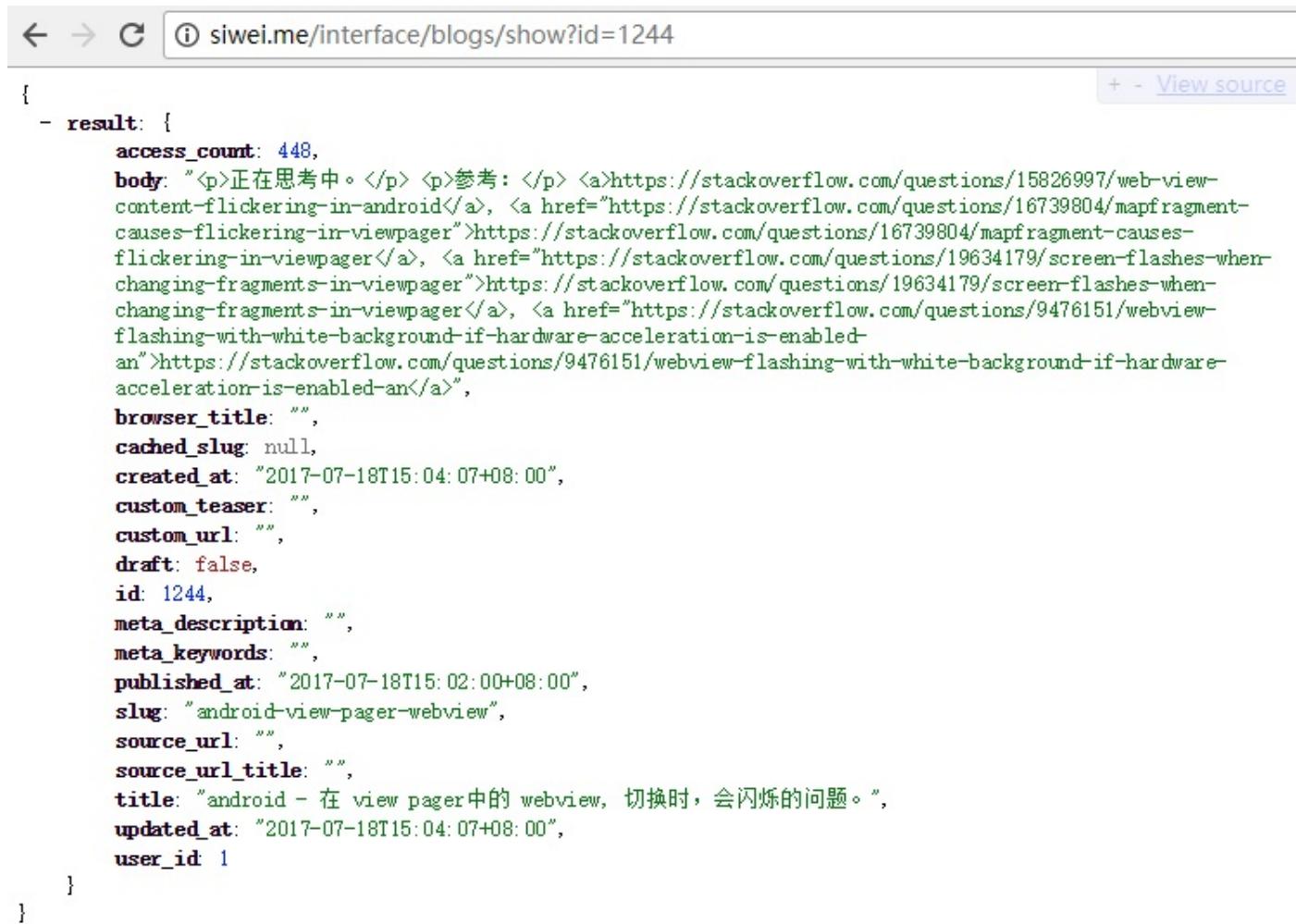
下面是该接口的一个完整形式：

`http://siwei.me/interface/blogs/show?id=1244`

下面是返回结果：

```
1. {
2.     "result": {
3.         "body": "<p>这个问题很常见，解决办法就是禁止硬件加速...</p>",
4.         "id": 1244,
5.         "title": "android - 在 view pager中的 webview，切换时，会闪烁的问题。"这个问题
6.         很常见，解决办法就是禁止硬件加速...</p>
7.     }
}
```

在浏览器中看起来如下图所示：



```

{
  - result: {
    access_count: 448,
    body: "<p>正在思考中。</p> <p>参考：</p> <a href='https://stackoverflow.com/questions/15826997/web-view-content-flickering-in-android'>https://stackoverflow.com/questions/15826997/web-view-content-flickering-in-android</a>, <a href='https://stackoverflow.com/questions/16739804/mapfragment-causes-flickering-in-viewpager'>https://stackoverflow.com/questions/16739804/mapfragment-causes-flickering-in-viewpager</a>, <a href='https://stackoverflow.com/questions/19634179/screen-flashes-when-changing-fragments-in-viewpager'>https://stackoverflow.com/questions/19634179/screen-flashes-when-changing-fragments-in-viewpager</a>, <a href='https://stackoverflow.com/questions/9476151/webview-flashing-with-white-background-if-hardware-acceleration-is-enabled-an'>https://stackoverflow.com/questions/9476151/webview-flashing-with-white-background-if-hardware-acceleration-is-enabled-an</a>",
    browser_title: "",
    cached_slug: null,
    created_at: "2017-07-18T15:04:07+08:00",
    custom_teaser: "",
    custom_url: "",
    draft: false,
    id: 1244,
    meta_description: "",
    meta_keywords: "",
    published_at: "2017-07-18T15:02:00+08:00",
    slug: "android-view-pager-webview",
    source_url: "",
    source_url_title: "",
    title: "Android – 在 view pager 中的 webview，切换时，会闪烁的问题。",
    updated_at: "2017-07-18T15:04:07+08:00",
    user_id: 1
  }
}

```

2. 显示博客详情页

我们需要新增Vue页面：`src/components/Blog.vue`，用于显示博客详情。

```

1.  <template>
2.    <div >
3.      <div>
4.        <p> 标题 : {{ blog.title }} </p>
5.        <p> 发布于 : {{blog.created_at }}</p>
6.        <div>
7.          {{ blog.body }}
8.        </div>
9.      </div>
10.     </div>
11.   </template>
12.
13. <script>
14. export default {
15.   data () {

```

```

16.     return {
17.       blog: {}
18.     },
19.   },
20.   mounted() {
21.     this.$http.get('api/interface/blogs/show?
22.       id=' + this.$route.query.id).then((response) => {
23.       this.blog = response.body.result
24.     }, (response) => {
25.       console.error(response)
26.     });
27.   }
28. </script>
29.
30. <style>
31. </style>

```

上面代码中：

- `data(){ blog: {}}` 用来初始化 blog这个页面用到的变量.
- `{blog.body}` , `{blog.title}` 等, 用来显示blog相关的信息.
- `mounted...` 中, 定义了发起http的请求.
- `this.$route.query.id` 获取url 中的id参数. 例如: `/my_url?id=333` , 那么 '333' 就是取到的结果. 具体的定义看下面的路由。

3. 新增路由

修改：`src/router/index.js` 文件. 添加如下代码:

```

1. import Blog from '@/components/Blog'
2.
3. export default new Router({
4.   routes: [
5.     // ...
6.     {
7.       path: '/blog',
8.       name: 'Blog',
9.       component: Blog
10.    }

```

```

11.    ]
12.  )

```

上面的代码，就是让Vue.js 可以对形如 <http://localhost:8080/#/blog> 的url进行处理。 对应的vue文件是 `@/components/Blog.vue`。

4. 修改博客列表页—跳转方式1：使用事件

我们需要修改博客列表页，增加跳转事件：

修改：`src/components/BlogList.vue`，如下代码所示：

```

1. <template>
2. ...
3.   <tr v-for="blog in blogs">
4.     <td @click='show_blog(blog.id)'>{{blog.title }}</td>
5.   </tr>
6. ...
7. </template>
8. <script>
9. export default {
10.   methods: {
11.     show_blog: function(blog_id) {
12.       this.$router.push({name: 'Blog', query: {id: blog_id}})
13.     }
14.   }
15. }
16. </script>

```

在上面代码中，

- `<td @click='show_blog(blog.id)'...</td>` 表示：该 `<td>` 标签在被点击的时候，会触发一个事件：`show_blog`，并且以当前正在遍历的 `blog`对象的 `id` 作为参数。
- `methods: {}` 是比较核心的方法，vue页面中用到的事件，都要写在这里。
- `show_blog: function...` 就是我们定义的方法。该方法可以通过 `@click="show_blog"` 来调用。
- `this.$router.push({name: 'Blog', params: {id: blog_id}})` 中：
 - `this.$router` 是vue的内置对象。表示路由。
 - `this.$router.push` 表示让vue跳转。跳转到 `name: Blog` 对应的vue页面。参数是

id: blog_id .

演示

(TODO 这里的GIF要换成静态图片)打开“博客列表页”，就可以看到对应的所有文章。然后点击其中一个的标题，就可以打开对应的文章详情页了。

如下图GIF动画所示：



不经过HTML转义, 直接打印结果.

我们发现，H T M L 的源代码在页面显示的时候被转义了，如下：



标题： android - android studio的最有用快捷键： 补全代码后直接跳到行末：ctrl + shift + enter

发布于： 2017-06-22T14:31:06+08:00

<p>很多时候， I D E 帮我们自动补全代码， 如下：</p> <pre>for(int i =0 ; i < some_array.length; i++光
标位置)</pre> <p>注意上面的 "光标位置"， 如果光标在这里， 可以直接输入： </p> <p>ctrl +
shift + enter ,</p> <p>于是上面的代码就会变成: </p> <pre>for(int i =0 ; i < some_array.length; i++){
光标位置 } </pre>

所以， 我们把它修改一下， 不要转义：

```
1. <template>
2. ....
3.     <div v-html='blog.body'>
4.     </div>
5. ....
6. </template>
```

上面的 `v-html` 就表示不转义。

效果如下图：



标题： android - 使用tablayout + view pager 实现 底部tab (bottom tab)

发布于： 2017-06-22T13:48:02+08:00

参考：<http://blog.csdn.net/wufeng55/article/details/53080602>

和 <http://blog.csdn.net/wufeng55/article/details/53064763>

安卓实现底部tab 貌似有多种方法， 包括：

radiogroup(很少用了)

tabhost

tablayout

下面是使用tab_layout的实现方法.

1. 建立 TabFragment:

```
package test.testandroidbottomtab;

import android.os.Bundle;
import android.support.annotation.Nullable;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
```

修改博客列表页—跳转方式2： 使用v-link

<router-link> 比起 会好一些，理由如下：

无论是 HTML5 history 模式还是 hash 模式，它的表现行为一致，所以，当你要切换路由模式，或者在 IE9 降级使用 hash 模式，无须作任何变动。

在 HTML5 history 模式下，router-link 会拦截点击事件，让浏览器不再重新加载页面。

当你在 HTML5 history 模式下使用 base 选项之后，所有的 to 属性都不需要写（基路径）了。

```
1.  <td>
2.    <router-link :to="{name: 'Blog', query: {id: blog.id}}">
3.      {{blog.id}}
4.    </router-link>
5.  </td>
```

然后，就可以看到，生成的HTML形如：

```
1. <a href="#/blog?id=1239" class="">
2.     1239
3. </a>
```

点击之后，有同样的跳转功能。

感兴趣的同学，可以查看：<https://router.vuejs.org/zh-cn/api/router-link.html>

路由

路由是所有前端框架中都必须具备的元素。 它定义了对于那个URL（页面），应该由那个文件来处理。

在Vue.js中，路由专门独立成为了一个项目：vue-router.

基本用法

每个vue页面，都要对应一个路由。例如，我们要做一个“博客列表页”，那么，我们需要两个东西：

1. vue文件，例如：`src/components/books.vue` 负责展示页面
2. 路由代码，让 `/#/books` 与 上面的vue文件对应。

下面的代码就是一个完整的路由文件：

```

1. import Vue from 'vue'
2. import Router from 'vue-router'
3.
4. // 增加这一行，作用是引入 SayHi 这个component
5. import SayHi from '@/components/SayHi'
6.
7. Vue.use(Router)
8. export default new Router({
9.   routes: [
10.
11.     // 增加下面几行，表示定义了 /#/say_hi 这个url
12.     {
13.       path: '/say_hi',
14.       name: 'SayHi',
15.       component: SayHi
16.     },
17.   ]
18. })

```

写法是固定的。其中的：

- `path`：定义了链接地址，例如：`/#/say_hi`
- `name`：为这个路由加个名字，方便以后引用，例如：`this.$router.push({name: 'SayHi'})`
- `component`：该路由由哪个component来处理。

跳转到某个路由时，带上参数

有路由，就会有参数，我们看一下路由是如何处理参数的。

1. 对于普通的参数

例如：

```
1. routes: [
2.   {
3.     path: '/blog',
4.     name: 'Blog'
5.   },
6. ]
```

在视图中，我们这样做：

```
1. <router-link :to="{name: 'Blog', query:{id: 3} }">User</router-link>
```

当用户点击这个代码生成的 html 页面时，就会触发跳转。

在 `<script/>` 中，也可以这样做：

```
1. this.$router.push({name: 'Blog', query: {id: 3}})
```

都会跳转到：`/#/blog?id=3` 这个url.

对于在路由中声明的参数

例如：

```
1. routes: [
2.   {
3.     path: '/blog/:id',
4.     name: 'Blog'
5.   },
6. ]
```

在视图中，我们这样做：

```
1. <router-link :to="{name: 'Blog', params: {id: 3} }">User</router-link>
```

在script中, 也可以这样做:

```
1. this.$router.push({name: 'Blog', params: {id: 3}})
```

都会跳转到: `/#/blog/3` 这个url.

根据路由获取参数

Vue的路由中, 参数获取有两种方式: query 和 params

获取普通参数

对于 `/#/blogs?id=3` 中的参数, 这样获取:

```
1. this.$route.query.id // 返回结果3
```

获取路由中定义好的参数

对于 `/#/blogs/3` 这样的参数, 对应的路由应该是:

```
1. routes: [
2.   {
3.     path: '/blogs/:id', // 注意这里的 :id
4.     ...
5.   },
6. ]
```

这个 named path, 就可以通过下面代码来获取id:

```
1. this.$route.params.id // 返回结果3
```

使用样式

样式用起来特别简单，直接写到 `<style>` 段落里面即可。如下代码所示：

```

1. <template>
2.   <div class='hi'>
3.     Hi Vue!
4.   </div>
5. </template>
6.
7. <script>
8. export default {
9.   data () {
10.     return { }
11.   }
12. }
13. </script>
14.
15. <style>
16. .hi {
17.   color: red;
18.   font-size: 20px;
19. }
20. </style>

```

用浏览器打开上述代码，就可以看到一个红色的，字体大小为20px的“Hi Vue!”。如下图所示：



使用全局

```

1. <style >
2. td {
3.   border-bottom: 1px solid grey;
4. }
5. </style>

```

使用局部的css

```
1. <style scoped>
2. td {
3.   border-bottom: 1px solid grey;
4. }
5. </style>
```

这段CSS只对当前的 component 适用。

也就是说，当我们有两个不同的页面： page1, page2，如果两个页面中都定义了某个样式（例如上面的 `td` ）的话，是不会互相冲突的。

因为Vuejs 会这样解析：

```
1. page1 的DOM：
2. <div data-v-7cf41e ... ></div>
3.
4. page2 的DOM:
5. <div data-v-3389dfw ... ></div>
```

而我们使用的 “scoped style” ，就可以存在于不同的页面 (component)上了！

双向绑定

双向绑定这个概念现在越来越普及。

在Angular出现的时候, 就作为宣传的王牌概念。现在几乎是个js前端框架, 就有这个功能。它的概念是:

某个变量, 定义于 `<script/>`, 需要展现在 `<template/>` 中的话:

1. 如果在代码层面进行修改, 那么页面的值就会发生变化
2. 如果在页面进行修改(例如在input标签中), 那么代码的值就会发生变化.

一个演示例子.

在我们的项目中, 增加一个 vue页面: `src/components/TwoWayBinding.vue`

```
1. <template>
2.   <div>
3.     <!-- 显示 this.my_value 这个变量 -->
4.     <p>页面上的值: {{my_value}} </p>
5.
6.     <p> 通过视图层, 修改my_value: </p>
7.     <input v-model="my_value" style='width: 400px' />
8.
9.     <hr/>
10.    <input type='button' @click="change_my_value_by_code()" value='通过控制代码修改my_value' />
11.    <hr/>
12.    <input type='button' @click="show_my_value()" value='显示代码中的my_value' />
13.  </div>
14. </template>
15.
16. <script>
17. export default {
18.   data () {
19.     return {
20.       my_value: '默认值',
21.     }
22.   },
23.   methods: {
24.     show_my_value: function(){
```

```

25.     alert('my_value: ' + this.my_value);
26.   },
27.   change_my_value_by_code: function(){
28.     this.my_value += "， 在代码中做修改， 666."
29.   }
30. }
31. }
32. </script>

```

上面的代码中，显示定义了一个变量“my_value”，这个变量可以在 `<script/>` 中访问和修改，也可以在 `<template/>` 中访问和修改。

- 在代码(`<script/>`)中访问的话，就是 `this.my_value`
- 在视图(`<template/>`)中访问的话，就是 `<input v-model=my_value />`

所以，这个就是双向绑定的方法。

接下来，修改路由文件：`src/router/index.js`：

```

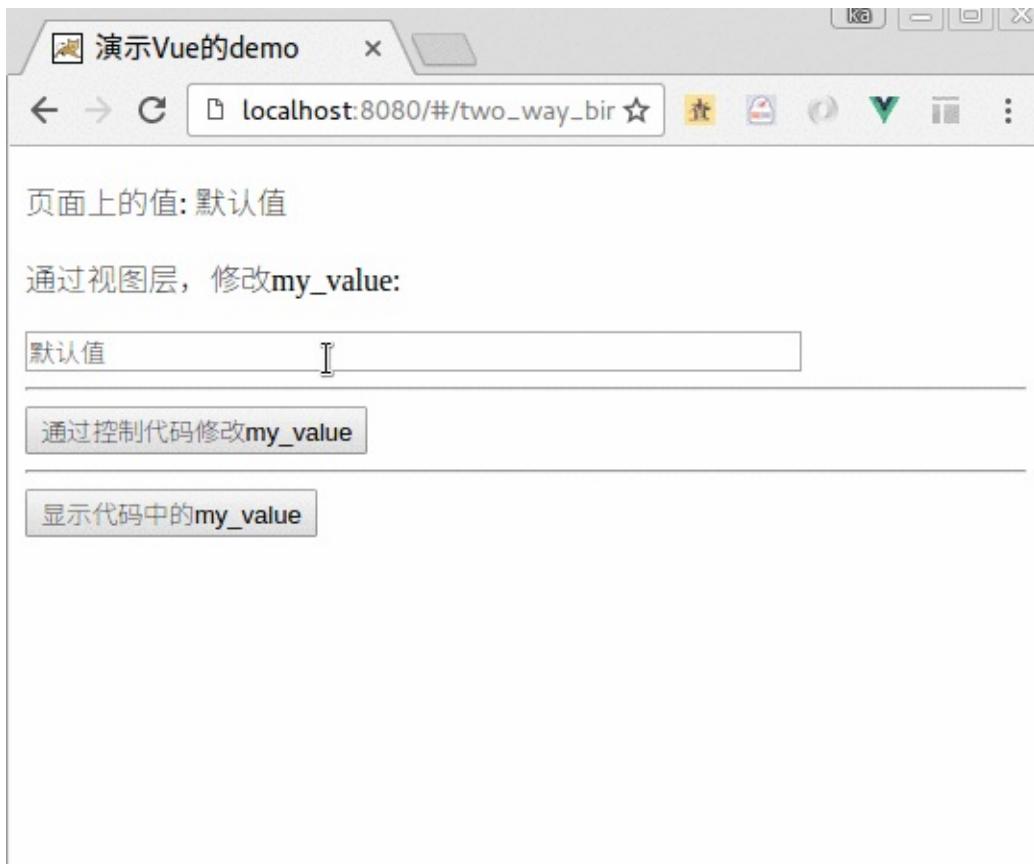
1. import TwoWayBinding from '@/components/TwoWayBinding'
2.
3. export default new Router({
4.   routes: [
5.     {
6.       path: '/two_way_binding',
7.       name: 'TwoWayBinding',
8.       component: TwoWayBinding
9.     }
10.   ]
11. })

```

然后，就可以用浏览器访问路径：http://localhost:8080/#/two_way_binding

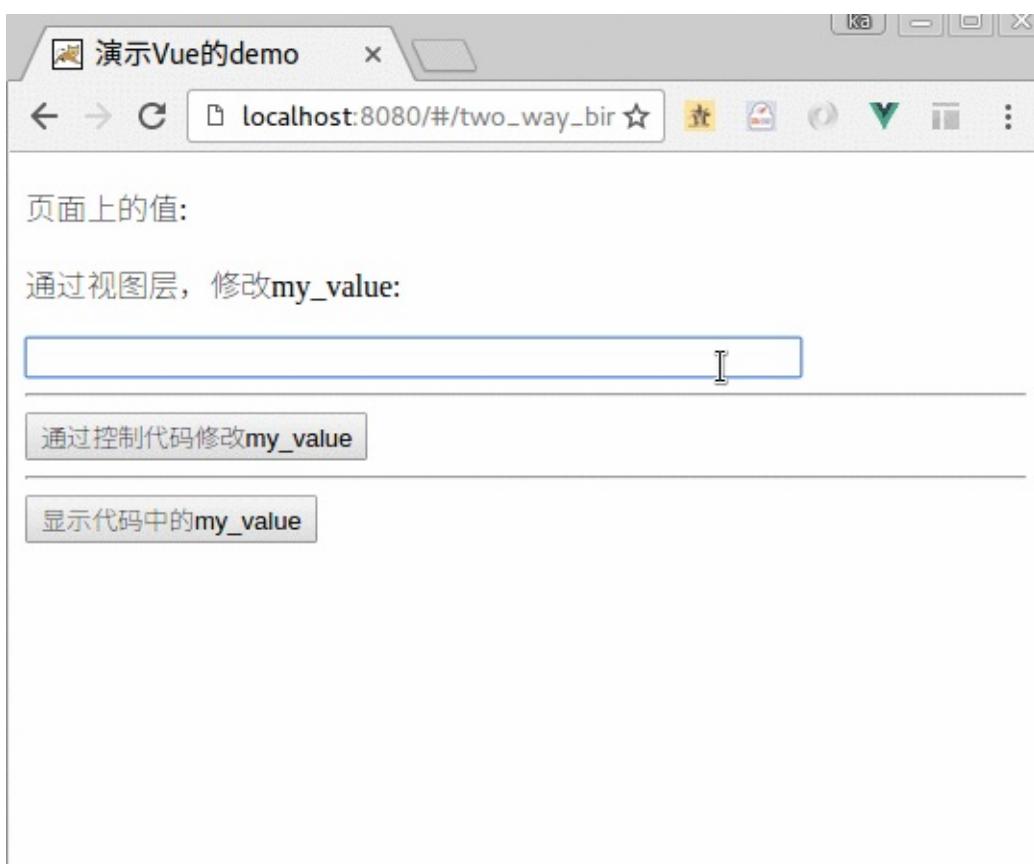
效果1：通过页面，修改js代码的值，可以看到，代码中的 `my_value` 一边，视图中的 `my_value` 就发生变化。

如下图所示：



效果 2：通过代码层面的改动，影响页面的值。

如下图所示：



所以，这个特性是Vue.js自带的。我们不需要刻意学它，只需要知道它可以达到这个目的，具备这个特

双向绑定

性，就可以了。

以后我们会发现，Vue.js 等前端框架中，这种思想和现象特别常用。

表单项的绑定

基本上，所有的表单项，无论是 `<input/>`，还是 `<textarea/>`，都需要使用 `v-model` 来绑定。

表单项： `input`, `textarea`, `select` 等.

使用`v-model`来绑定 输入项

```
1. <input v-model="my_value" style='width: 400px' />
```

就可以在代码中获取到 `this.my_value` 的值 .

表单项的完整例子

```
1. <template>
2.   <div>
3.
4.     input: <input type='text' v-model="input_value"/>,
5.     输入的值 : {{input_value}}
6.     <hr/>
7.     text area: <textarea v-model="textarea_value"></textarea>,
8.     输入的值 : {{textarea_value}}
9.     <hr/>
10.    radio:
11.      <input type='radio' v-model='radio_value' value='A' /> A,
12.      <input type='radio' v-model='radio_value' value='B' /> B,
13.      <input type='radio' v-model='radio_value' value='C' /> C,
14.      输入的值 :
15.      {{radio_value}}
16.      <hr/>
17.      checkbox:
18.        <input type='checkbox' v-model='checkbox_value'
19.          v-bind:true-value='true'
20.          v-bind:false-value='false'
21.          /> ,
22.        输入的值 :
23.        {{checkbox_value}}
24.        <hr/>
```

```

25.     select:
26.       <select v-model='select_value'>
27.         <option v-for="e in options" v-bind:value="e.value">
28.           {{e.text}}
29.         </option>
30.       </select>
31.       输入的值 : {{select_value}}
32.
33.     </div>
34.   </template>
35.
36.   <script>
37.     export default {
38.       data () {
39.         return {
40.           input_value: '',
41.           textarea_value: '',
42.           radio_value: '',
43.           checkbox_value: '',
44.           select_value: 'C',
45.           options: [
46.             {
47.               text: '红烧肉', value: 'A'
48.             },
49.             {
50.               text: '囊包肉', value: 'B'
51.             },
52.             {
53.               text: '水煮鱼', value: 'C'
54.             }
55.           ]
56.         }
57.       },
58.       methods: {
59.     }
60.   }
61. </script>

```

对于select 的option, 使用 `v-bind:value` 来绑定option的值.

效果如图:

localhost:8080/#/form_input

input : , 输入的值 : apple

text area : , 输入的值 : banana

radio: A, B, C, 输入的值 : C

checkbox: , 输入的值 : true

select: 输入的值 : B

动图如下：

input : , 输入的值 :

text area : , 输入的值 :

radio: A, B, C, 输入的值 :

checkbox: , 输入的值 :

select: 输入的值 : C

Modifiers (后缀词)

.lazy

可以让输入后不会立刻变化， 而是等焦点彻底离开后（触发 `blur()` 事件后）才会触发视图层的值的变化。

使用方式：

```
1. <input type='text' v-model.lazy="input_value"/>
```

这个可以用在某些需要等待用户输入完字符串才需要给出反应的情况，例如“搜索”。

.number

强制要求输入数字

使用方式：

```
1. <input type='text' v-model.lazy="input_value" type="number"/>
```

.trim

强制对输入的值进行去掉 前后的空格。

使用方式：

```
1. <input type='text' v-model.trim="input_value" />
```

表单的提交

大家要切记这一点： 在任何 Single Page App中，js代码都不会产生一个传统意义的form表单提交！（这会引起整个页面的刷新）

所以，我们往往用事件来实现。（桌面开发思维）

例如，在远程有个接口，可以接受别人的留言：

- URL: http://siwei.me/interface/blogs/add_comment
- 参数： `content` : 留言的内容.
- 请求方式： `POST`
- 返回结果：

```
1. {"result":"ok","content":"(留言的内容)"}
```

我们可以先用POSTMAN来确认一下：

The screenshot shows the Postman application window. The top navigation bar includes 'Postman', 'Runner', 'Import', 'Builder' (which is selected), 'Team Library', and various status indicators like 'SYNC OFF'. The main interface has a search bar with 'http://siwei.me/interface/blogs/add_comment' and a 'Send' button. Below it, the 'Body' tab is selected under 'form-data'. A table shows a single entry: 'content' with value 'very good!'. At the bottom, the response status is 'Status: 200 OK' and 'Time: 628 ms'. The 'Pretty' view shows the JSON response: {"result": "ok", "content": "very good!"}.

例如，下面的代码，就是把输入的表单，提交到我们的后台。

新增加一个文件：`/src/components/FormSubmit.vue`，内容如下：

```

1. <template>
2.   <div>
3.     <textarea v-model='content'>
4.     </textarea>
5.     <br/>
6.     <input type='button' @click='submit' value='留言' />
7.   </div>
8. </template>
9. <script>
10.
11. export default {
12.   data () {
13.     return {
14.       content: ''
15.     }
16.   },
17.   methods: {
18.     submit: function(){
19.       this.$http.post('/api/interface/blogs/add_comment',
20.         {
21.           content: this.content
22.         }
23.       )
24.       .then((response) => {
25.         alert("提交成功!, 刚才提交的内容是：" + response.body.content)
26.       },
27.         (response) => {
28.           alert("出错了")
29.         }
30.       )
31.     }
32.   }
33. }
34. </script>

```

上面的代码中：

```

1.   <textarea v-model='content'>
2.   </textarea>

```

就是待输入的表单项。

```
1. <input type='button' @click='submit' value='留言' />
```

则是按钮，点击后会触发提交表单的函数 `submit` .

```
1.     submit: function(){
2.       this.$http.post('/api/interface/blogs/add_comment',
3.       {
4.         content: this.content
5.       }
6.     )
7.     .then((response) => {
8.       alert("提交成功！， 刚才提交的内容是：" + response.body.content)
9.     },
10.    (response) => {
11.      alert("出错了")
12.    }
13.  )
14. }
```

上面的代码，定义了提交表单的具体函数 `submit` .

- `this.$http.post` 表示 发起的http 的类型是 post.
- `post` 函数的第一个参数是 url, 第二个参数是一个json, `{ content: this.content}` 代表了我们要提交的数据
- `then` 函数的处理同 http get 请求

接下来，我们修改路由文件：`src/router/index.js` , 增加内容如下：

```
1. import FormSubmit from '@/components/FormSubmit'
2.
3. export default new Router({
4.   routes: [
5.     {
6.       path: '/form_submit',
7.       name: 'FormSubmit',
8.       component: FormSubmit
9.     }
10.   ]
11. })
```

访问url：`http://localhost:8080/form_submit` , 输入一段字符串，如下图所示：

localhost:8080/#/form_submit

申老师, 我学习到了 Vuejs的表单的提交了。

留言

点击提交按钮，就可以看到，内容已经提交，并且得到了返回的response，触发了 `alert`，如下图所示：

localhost:8080 显示

提交成功! 刚才提交的内容是: 申老师, 我学习到了 Vuejs的表单的提交了。

确定

Elements Console Sources Network Performance Memory Application Security

View: Group by frame Preserve log Disable cache Offline

Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest

| Name | Status | Type | Initiator | Size |
|-------------------------------------|-----------|------|------------------------|----------------|
| add_comment
/api/interface/blogs | 200
OK | xhr | vue-resource.es2015... | 573 B
136 B |

查看一下返回的json：

```
{"result":"ok","content":"\u7533\u8001\u5e08\uuff0c\u6211\u5b66\u4e60\u5230\u4e86\n1. Vuejs\u7684\u8868\u5355\u7684\u63d0\u4ea4\u4e86\u3002"}
```

至此，完成了一个完整的 输入表单，提交表单的过程。

Component 组件

组件是 Vue.js 中最重要的部分之一。学习需要一定的时间投入。

在“webpack”项目中，每一个页面文件（.vue）都可以认为是一个组件。

在 Vue.js 1.x 中，组件跟 视图 是分别放到不同的文件夹下面的。

在 Vue.js 2.8 以后，所有的视图文件，都保存到‘components’目录下。

可见“组件”这个概念已经越来越重要和普及了。

说明

这里说的内容，跟 官方文档 的“原始组件”不是一个东西。

本章的内容，仅使用于“webpack 项目”中的组件。对应的文档是：[单文件组件](#)

如何查看文档

- 先快速，粗糙的查看“原始组件”：<https://cn.vuejs.org/v2/guide/single-file-components.html>

对所有的概念有所了解。（因为这个原始组件的开发环境跟 webpack 下项目的开发环境不太一样，所以很多以 webpack 作为入门的同学（例如本书读者）会蒙圈）

- 再看“单文件组件”：<https://cn.vuejs.org/v2/guide/single-file-components.html>，就可以对 webpack 项目下的组件有清晰的认识。
- 遇到问题之后，再看“原始组件”，这里包括很多 API 级别的概念和解释。

Component 的重要作用：重用代码

我们可以想象一个场景：有两个页面，每个页面的头部都要有个 LOGO 图片。

那么我们每次都写成原始 HTML 的话，代码会比较重复：

页面1的代码如下：

```
1.  <div class='logo'>
    <img src='http://siweitech.b0.upaiyun.com//image/569/upsz-
2.  fyfkzhs9232258.jpg'>
```

```
3.    </div>
4.    页面1的其他代码
```

页面2的代码如下：

```
1.    <div class='logo'>
2.        <img src='http://siweitech.b0.upaiyun.com//image/569/upsz-
2. fyfkzhs9232258.jpg'>
3.    </div>
4.    页面2的其他代码
```

所以，我们应该把这段代码抽取出来，成为一个新的组件(component)。

组件的创建

新建一个文件：`src/components/Logo.vue`

```
1.  <template>
2.    <div class='logo'>
3.        <img src='http://siweitech.b0.upaiyun.com//image/570/siwei.me_header.png'>
4.    </div>
5.  </template>
```

这个文件定义了一个最最简单的component。

然后，修改对应的页面：

```
1.  <template>
2.    <div >
3.        <my-logo>
4.        </my-logo>
5.        ...
6.    </template>
7.
8.  <script>
9.  import MyLogo from '@/components/Logo'
10.
11. export default {
12.     ...
13.     components: {
14.         MyLogo
```

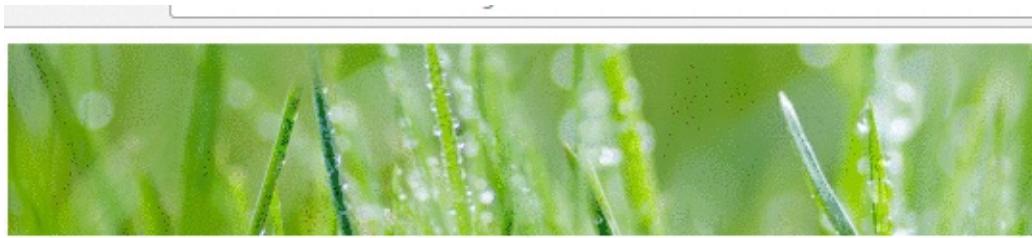
```
15.    }
16. }
```

注意1：上面代码中的 `components: { MyLogo }`，必须是这个写法，它等同于：

```
1. components: {
2.   MyLogo: MyLogo // 前面的MyLogo 是template中的名字
3.           // 后面的MyLogo 是import进来的代码.
4. }
```

注意2：上面代码中定义的组件，虽然名字叫做 `MyLogo`，但是在 `<template/>` 中使用的时候，需要写作 `<my-logo></my-logo>`。

保存代码，我们刷新一下，发现两个页面都发生了变化，如下图：



向组件中传递参数

如果我们希望两个页面，每个页面都有个 `title`，但是内容不同，该怎么办？就需要传递参数给 Component 了。

声明组件的时候， 我们需要这样：

修改文件： `src/components/Logo.vue`

```

1. <template>
2.   <div class='logo'>
3.     <h1>{{title}}</h1>
4.     ...
5.   </div>
6. </template>
7.
8. <script>
9. export default {
10.   props: ['title'] // 加上了这个声明.
11. }
12. </script>
```

可以看到，上面的代码中， 增加了这么几行代码：

```

1. export default {
2.   props: ['title']
3. }
```

上面的代码表示， 为该Component 增加了一个 “property(属性)”， 属性的名字叫做 “title”.

组件接收字符串作为参数

在调用的时候， 传递字符串：

```

1. <my-logo title="博客列表页">
2. </my-logo>
```

就可以了。

组件接收变量作为参数

如果要传递的参数是一个变量， 就要这么写：

```

1. <template>
2.   <my-logo :title="title">
3.   </my-logo>
4.   <input type='button' @click='change_title' value='点击修改标题' /><br/>
```

```

5. </template>
6.
7. <script>
8. export default {
9.   data: function() {
10.     return {
11.       title: '博客列表页',
12.     }
13.   },
14.   methods: {
15.     change_title: function(){
16.       this.title = '好多文章啊(标题被代码修改过了)'
17.     }
18.   },
19. </script>

```

如下图：

博客列表页



[点击修改标题](#)

[1248 vuejs - 提取所有的CSS到单个文件](#)

[1247 vuejs - 解决post请求变成option的请求问题.](#)

[1246 ruby - windows下的安装](#)

[1245 vuejs - mixin的基本用法](#)

[1244 android - 在 view pager中的 webview, 切换时, 会闪烁的问题。](#)

[1243 证照 - 如何开具无行贿犯罪记录证明](#)

[1242 java - ant的基本用法](#)

[1241 java - eclipse的基本用法](#)

[1240 java - eclipse 中, 启动一个项目之前, 要设置好 lib 的各种依赖](#)

[1239 java - linux下启动tomcat](#)

[1238 rspec 不再输出 警告信息：--deprecation-out temp](#)

[1237 android - android studio的是右用拉锁键，如今代码后直接到到行本，](#)

脱离Webpack , 在原生Vuejs中创建 Component

非常简单。 如下面代码所示：

```
1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.     <study-process></study-process>
8.   </div>
9.   <script>
10.    Vue.component('study-process', {
11.      data: function () {
12.        return {
13.          count: 0
14.        }
15.      },
16.      template: '<button v-on:click="count++">我学习到了第 {{ count }} 章.
17.      </button>'
18.    })
19.    var app = new Vue({
20.      el: '#app',
21.      data: {
22.      })
23.    </script>
24.  </body>
25. </html>
```

该代码首先声明了一个component：

```
1.      Vue.component('study-process', {
2.        data: function () {
3.          return {
4.            count: 0
5.          }
6.        },
7.        template: '<button v-on:click="count++">我学习到了第 {{ count }} 章.
8.        </button>'
9.      })
```

可以看出，该component 中定义了一个 `data` 代码段， 里面有个变量 `count` .

然后定义一个 `template` 段落即可。

运维和发布Vue.js项目

传统的公司，特别喜欢一个角色用一个人。 例如在生产流水线上，一个工人只负责扭螺丝，另一个工人只负责喷漆。

很多软件公司也是这样，有人专门负责写代码，有人专门负责运维。

这样的弊端是： 出了问题，程序员摸不到服务器，不知道问题在哪里。 运维同学可以碰服务器，但是看不懂代码，也没法解决问题。

所以传统公司往往怕出问题，因为解决起来特别慢。

现在，越来越多的人意识到，让程序员懂得运维知识，特别重要。 最好的运维，就是程序员自己。 在2011年，国外就开始流行一个词汇： DevOps(Developer + Operations)，也叫敏捷运维。 就是对又懂编程又懂运维的人的称呼。

我们要有追求，做一个会运维的编程好手。 做个DevOps。

就算公司有明确要求，部署专门有运维人员，我们在他们做部署时，也要搬个板凳过去旁观，参与。

运维人员也特别乐于开发人员陪他们一起发布。

你会发现，无论在程序员同伴眼中，还是在运维同学眼中，你变得越来越重要。 随着你的知识的进展，自动化脚本的学习，仿佛每次部署都变得越来越简单~

打包和部署

平时我们开发时，是在本地做开发，每次打开的都是 <http://localhost:8080/#/>。而真实的项目中，我们肯定要把项目部署到某个地方。

所以我们需要对项目进行打包和编译。

另外，在正式上线之前，也要在测试服务器上进行发布，这样才能看到一些平时在 localhost 上看不到的问题。

打包

直接使用下面命令，就可以把vue项目打包：

```
1. $ npm run build
```

这个命令运行过程如下：

```
1. siwei@siwei-linux:/workspace/test_vue_0613$ npm run build
2.
3. > test_vue_0613@1.0.0 build /workspace/test_vue_0613
4. > node build/build.js
5.
6. :: building for production...
7. Starting to optimize CSS...
8. Processing static/css/app.32ddfe6eea5926f8e3c760d764fef3fa.css...
   Processed static/css/app.32ddfe6eea5926f8e3c760d764fef3fa.css, before: 142,
9. after: 74, ratio: 52.11%
10. Hash: f89cd58bdaf8a153e13e
11. Version: webpack 2.6.1
12. Time: 18658ms
                                         Asset      Size  Chunks
13. Chunk Names
                                         static/js/app.d8b9f437c302a7070fe7.js    9.1 kB    0
14. [emitted]  app
                                         static/js/vendor.33c767135f1684f458a7.js    122 kB    1
15. [emitted]  vendor
                                         static/js/manifest.75e2ba037e0bc6934514.js    1.51 kB    2
16. [emitted]  manifest
                                         static/css/app.32ddfe6eea5926f8e3c760d764fef3fa.css    74 bytes    0
17. [emitted]  app
```

```

          static/js/app.d8b9f437c302a7070fe7.js.map      63.5 kB     0
18. [emitted]  app
    static/css/app.32ddfe6eea5926f8e3c760d764fef3fa.css.map  623 bytes     0
19. [emitted]  app
    static/js/vendor.33c767135f1684f458a7.js.map      950 kB     1
20. [emitted]  vendor
    static/js/manifest.75e2ba037e0bc6934514.js.map     14.6 kB     2
21. [emitted]  manifest
                index.html  522 bytes
22. [emitted]
23.
24. Build complete.
25.
26. Tip: built files are meant to be served over an HTTP server.
27. Opening index.html over file:// won't work.

```

可以看到：

- 整个过程耗时 18.658s
- 使用的 webpack 版本是 2.6.1
- 对CSS文件进行了优化. (优化的比率是 52.11%)
- 所有的 .vue 文件, 都被打包编译成了下面的文件:

```

1. $ find ./dist
2.
3. ./static
4. ./static/css
5. ./static/css/app.32ddfe6eea5926f8e3c760d764fef3fa.css
6. ./static/css/app.32ddfe6eea5926f8e3c760d764fef3fa.css.map
7. ./static/js
8. ./static/js/vendor.33c767135f1684f458a7.js.map
9. ./static/js/app.d8b9f437c302a7070fe7.js.map
10. ./static/js/manifest.75e2ba037e0bc6934514.js
11. ./static/js/manifest.75e2ba037e0bc6934514.js.map
12. ./static/js/app.d8b9f437c302a7070fe7.js
13. ./static/js/vendor.33c767135f1684f458a7.js
14. ./index.html

```

其中，包括了 js, css , map 和 index.html

我们需要把它放到 http 服务器上, 例如: nginx , apache.

部署

1. 上传代码到远程服务器

我们使用 `scp` 或者 `ftp` 方式, 可以把代码上传到服务器, 假设我们的服务器是 `linux`,

- 路径是 : `/opt/app/test_vue`
- 服务器ip: `123.255.255.33`
- 服务器ssh端口: `6666`
- 服务器用户名: `root`

```
1. $ scp -P 6666 -r dist root@123.255.255.33:/opt/app
2. index.html                                100%   528      0.5KB/s  00:00
3. app.32ddfe6eea5926f8e3c760d764fef3fa.css    100%    74      0.1KB/s  00:00
4. app.32ddfe6eea5926f8e3c760d764fef3fa.css.map  100%   623      0.6KB/s  00:00
5. vendor.33c767135f1684f458a7.js.map          100%  927KB  927.3KB/s  00:00
6. app.d8b9f437c302a7070fe7.js.map            100%   63KB   62.6KB/s  00:00
7. manifest.75e2ba037e0bc6934514.js           100%  1511     1.5KB/s  00:00
8. manifest.75e2ba037e0bc6934514.js.map       100%   14KB   14.3KB/s  00:00
9. app.d8b9f437c302a7070fe7.js                100%  9323     9.1KB/s  00:00
10. vendor.33c767135f1684f458a7.js            100%  119KB  118.7KB/s  00:00
```

这样, 就把本地的 `dist` 目录, 上传到了远程的 `/opt/app` 目录上.

2. 配置远程服务器

2.1 登陆远程服务器:

```
1. $ ssh root@123.255.255.23 -p 6666
2. (输入密码, 回车)
3.
4. Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-86-generic x86_64)
5.
6. root@my_server:~#
```

2.2 把刚才上传的文件夹重命名成: `vue_demo` :

```
1. # mv /opt/app/dist /opt/app/vue_demo
```

2.3 配置nginx, 使 域名: `vue_demo.siwei.me` 指向该位置:

把下面代码, 加入到nginx的配置文件中(`/etc/nginx/nginx.conf`)

```
1. server {
2.     listen      80;
3.     server_name vue_demo.siwei.me;
4.     client_max_body_size      500m;
5.     charset utf-8;
6.     root /opt/app/vue_demo;
7. }
```

重启nginx之前, 测试一下刚才加入的代码是否有问题:

```
1. # nginx -t
2. nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
3. nginx: configuration file /etc/nginx/nginx.conf test is successful
```

可以看到, 没问题.

2.4 然后重启nginx :

```
1. # nginx -s stop
2. # nginx
```

3. 修改域名配置

nginx跑起来之后, 我们就要配置域名. 否则无法访问.

3.1 我们需要增加个二级域名: vue_demo.siwei.me

以dnspod为例, 需要设置这个二级域名的: A记录. IP地址:

记录类型 A

主机记录 vue_demo

线路类型 默认

关联云资源 是 否

记录值 123.57.235.33

TTL 10分钟

确定 取消

保存 .

3.2 回到命令行，输入 ping 命令，确认：

```
1. $ ping vue_demo.siwei.me
2. PING vue_demo.siwei.me (123.57.235.33) 56(84) bytes of data.
3. 64 bytes from 123.57.235.33: icmp_seq=1 ttl=54 time=5.79 ms
4. 64 bytes from 123.57.235.33: icmp_seq=2 ttl=54 time=6.38 ms
5. 64 bytes from 123.57.235.33: icmp_seq=3 ttl=54 time=9.25 ms
```

说明 我的二级域名 `vue_demo.siwei.me` 已经可以正常指向到 我的服务器ip了。

4. 部署完成！

打开浏览器，访问 http://vue_demo.siwei.me 就可以看到效果了，如下图所示：



例子列表-2017-10-9

- [Hello](#): 显示最基本的Vuejs
- [SayHi](#): 第一个Vue页面
- [SayHiFromVariable](#): 在页面中使用参数
- [BlogList](#): 调用真实接口, 并渲染出博客列表
- [Blog](#): 调用真实接口, 显示blog文章
- [TwoWayBinding](#): 一个双向绑定的例子.
- [FormInput](#): 表单输入项大全.
- [FormSubmit](#): 提交表单的例子.
- 组件的使用, 见: [BlogList](#) 和 [Blog](#)
- [SayHiFromMixin](#): Mixin的例子.
- Vuex的例子: [倒计时页面1](#)
- [倒计时页面2](#)

文字版教程在: https://github.com/sg552/happy_book_vuejs

解决域名问题与跨域问题

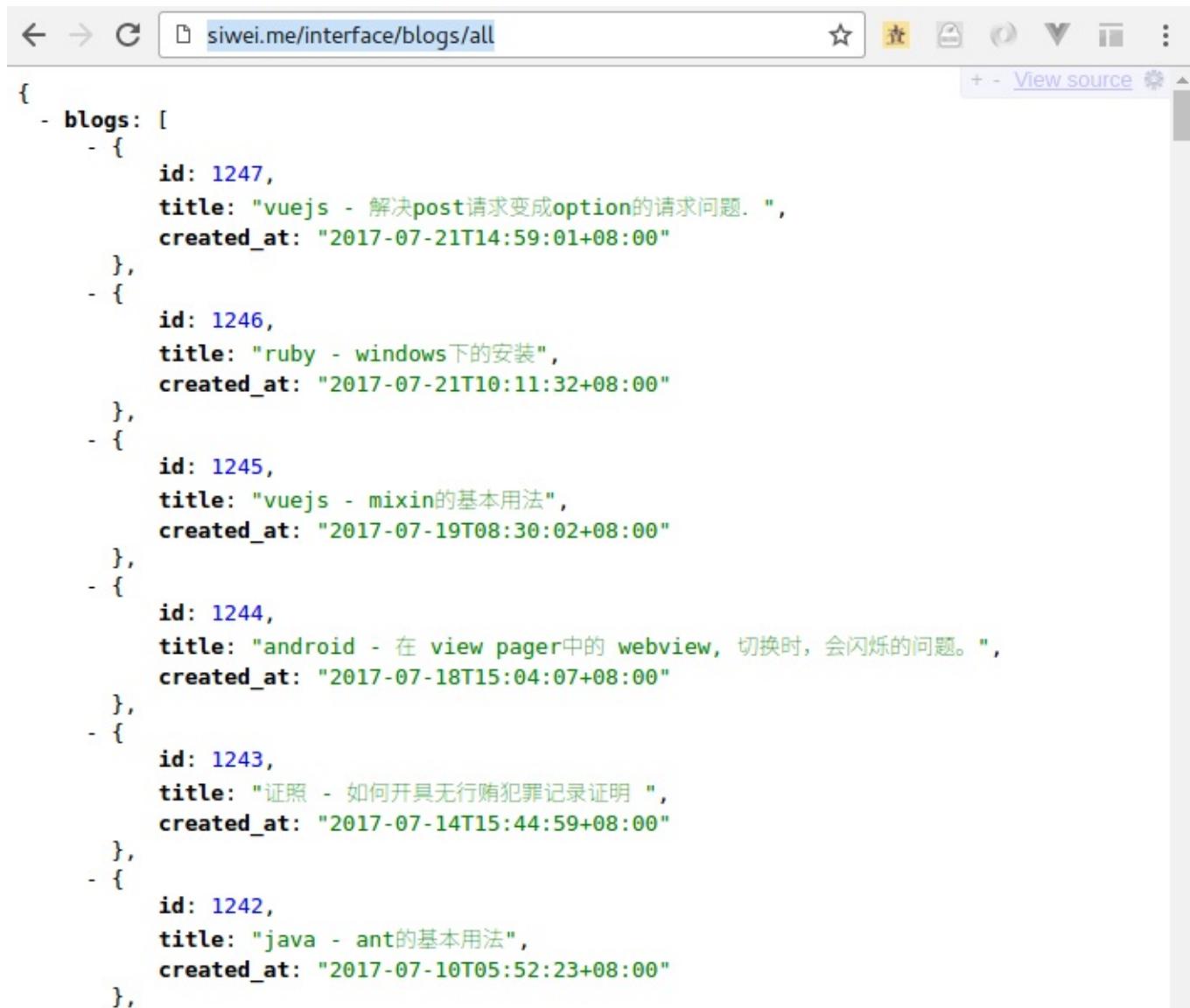
我们在部署之后，会发现Vue.js会遇到js的经典问题：远程服务器地址不对，或者跨域问题。

还是用我们本书中的例子为例。

我们的真正后台接口是：

<http://siwei.me/interface/blogs/all>

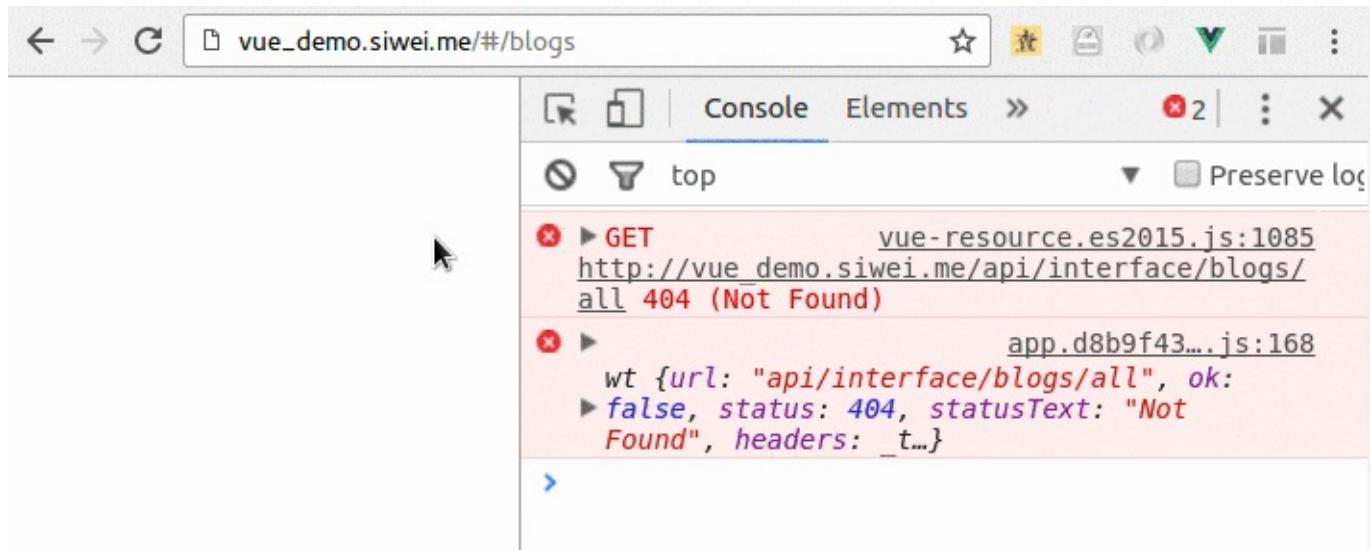
如下：



```
{  
  - blogs: [  
    - {  
      id: 1247,  
      title: "vuejs - 解决post请求变成option的请求问题.",  
      created_at: "2017-07-21T14:59:01+08:00"  
    },  
    - {  
      id: 1246,  
      title: "ruby - windows下的安装",  
      created_at: "2017-07-21T10:11:32+08:00"  
    },  
    - {  
      id: 1245,  
      title: "vuejs - mixin的基本用法",  
      created_at: "2017-07-19T08:30:02+08:00"  
    },  
    - {  
      id: 1244,  
      title: "android - 在view pager中的webview, 切换时, 会闪烁的问题。",  
      created_at: "2017-07-18T15:04:07+08:00"  
    },  
    - {  
      id: 1243,  
      title: "证照 - 如何开具无行贿犯罪记录证明",  
      created_at: "2017-07-14T15:44:59+08:00"  
    },  
    - {  
      id: 1242,  
      title: "java - ant的基本用法",  
      created_at: "2017-07-10T05:52:23+08:00"  
    },  
  ]  
}
```

域名404 问题

1. 使用浏览器打开页面：http://vue_demo.siwei.me/#/blogs， 页面出错。



2. 可以看到，出错的原因是 404， 打开

[“http://vue_demo.siwei.me/api/interface/blogs/all”](http://vue_demo.siwei.me/api/interface/blogs/all)



3. 这个问题是由于源代码中, 访问 `/interface/blogs/all` 这个接口引起的：

在文件 `src/components/BlogList.vue` 中, 第41行, 我们定义了远程访问的url:

```
1. this.$http.get('/api/interface/blogs/all')...
```

如下图所示：

```
31  methods: {
32    show_blog: function(blog_id) {
33      console.info("blog_id:" + blog_id)
34      this.$router.push({name: 'Blog', query: {id: blog_id}})
35    },
36    change_title: function(){
37      this.title = '好多文章啊(标题被代码修改过了)'
38    }
39  },
40  mounted() {
41    this.$http.get('/api/interface/blogs/all').then((response) => {
42      console.info(response.body)
43      this.blogs = response.body.blogs
44    }, (response) => {
45      console.error(response)
46    });
47  },
48  components: {
49    MyLogo
50  }
51
```

这是因为，在我们开发的时候，`vuejs` 会通过 `$npm run dev` 命令，跑起一个“开发服务器”，这个server中有一个代理，可以把所有的以 `'/api'` 开头的请求，如：

```
1. localhost:8080/api/interface/blogs/all
```

转发到：

```
1. siwei.me/interface/blogs/all
```

“开发服务器”的配置如下：

```
1. proxyTable: {
2.   '/api': {
3.     target: 'http://siwei.me',
4.     changeOrigin: true,
5.     pathRewrite: {
6.       '^/api': ''
7.     }
8.   }
9. },
```

所以，在开发环境下，一切正常。

但是在生产环境中，发起请求的时候，就不存在代理服务器，不存在开发服务器（dev server）了，所以会出错。

(这个问题的解决办法，我们等下再讲。)

跨域问题

这个问题，是js的经典问题。

比如，有的同学，在解决上面的问题的时候，会问：老师，我们直接把上图中 41 行的：

```
1. this.$http.get('/api/interface/blogs/all')
```

改成：

```
1. this.$http.get('http://siwei.me/interface/blogs/all')
```

不就可以了吗？

答案是不可以。 请动手试一下再说话。

一动手，我们就会发现， 如果 `vue_demo.siwei.me` 直接访问 `siwei.me` 域名下的资源，会报错。
因为他们是两个不同的域名。

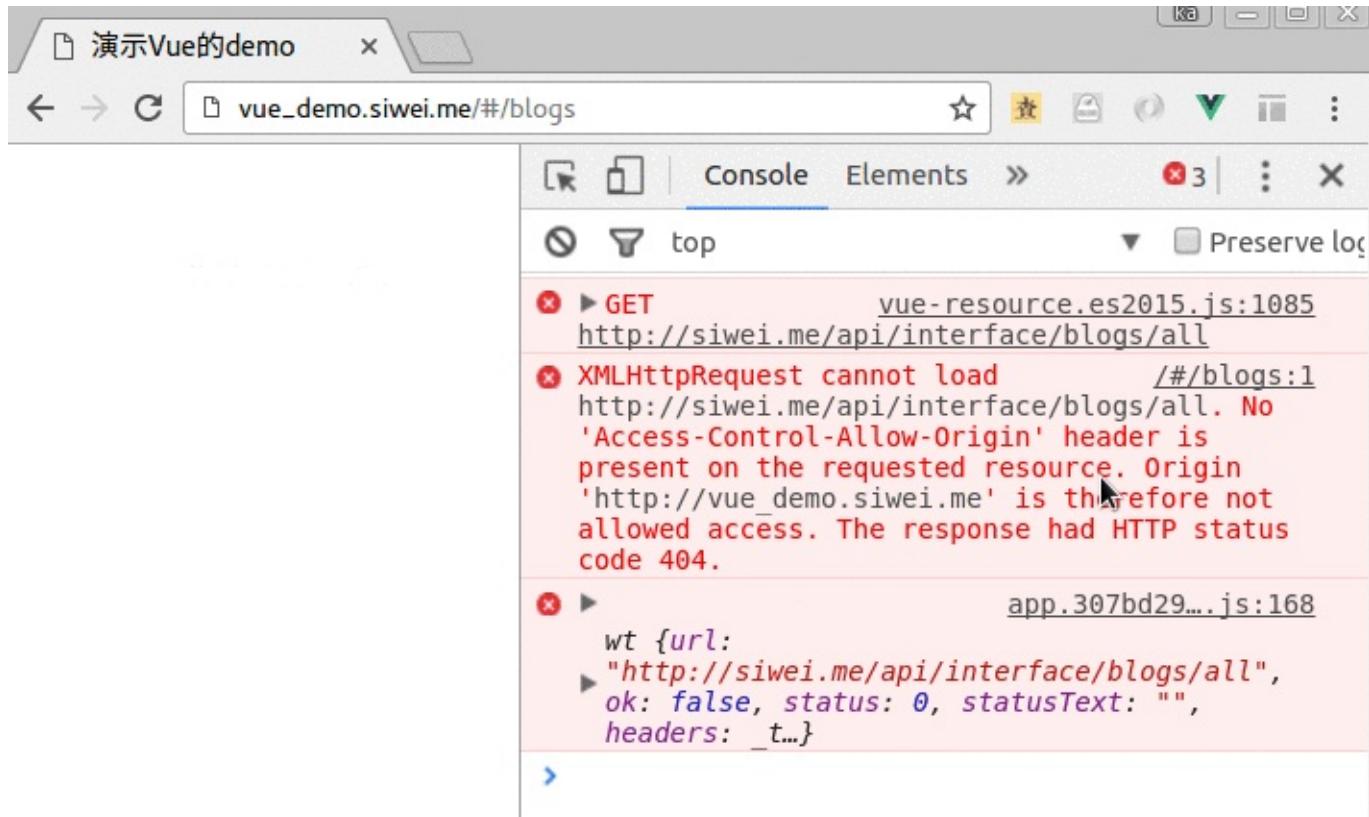
代码形如：

```
1. this.$http.get('http://siwei.me/api/interface/blogs/all')...
```

我们就会得到报错：

```
1. XMLHttpRequest cannot load http://siwei.me/api/interface/blogs/all.  
2. No 'Access-Control-Allow-Origin' header is present on the requested resource.  
3. Origin 'http://vue_demo.siwei.me' is therefore not allowed access.
```

如下图所示：



解决域名问题和跨域问题

其实，上面提到的两个问题，根源都是一个。所以解决办法都是一样的。

1. 在代码端，处理方式不变，访问 `/api` + 原接口url。（无变化）

```
1. this.$http.get('/api/interface/blogs/all')...
```

2. 在开发的时候，继续保持vuejs 的代理存在。配置代码如下：（无变化）

```
1. proxyTable: {
2.   '/api': {
3.     target: 'http://siwei.me',
4.     changeOrigin: true,
5.     pathRewrite: {
6.       '^/api': ''
7.     }
8.   }
9. },
```

3. 在nginx的配置文件中，加入代理：(详细说明见代码中的注释)（这个是新增的）

```
1. server {
```

```
2.     listen      80;
3.     server_name  vue_demo.siwei.me;
4.     client_max_body_size      500m;
5.     charset utf-8;
6.     root /opt/app/vue_demo;
7.
8.     # 第一步,把所有的 mysite.com/api/interface 转换成:    mysite.com/interface
9.     location /api {
10.        rewrite    ^(.*)\api(.*)$    $1$2;
11.    }
12.
13.    # 第二步,    把所有的 mysite.com/interface 的请求, 转发到 siwei.me/interface
14.    location /interface {
15.        proxy_pass          http://siwei.me;
16.    }
17. }
```

就可以了。

也就是说， 上面的配置，把

```
1. http://vue_demo.siwei.me/api/interface/blogs/all
```

在服务器端的nginx中做了个变换，相当于访问了：

```
1. http://siwei.me/interface/blogs/all
```

重启nginx ， 就会发现生效了。

如下所示：

The screenshot shows a browser window with the title "演示Vue的demo" and the URL "vue_demo.siwei.me/#/blogs". The main content area lists various blog posts with IDs and titles. To the right is a developer tools sidebar showing a call stack entry: "app.4b7a835...js:165 Object {blogs: Array[1230]}".

| ID | Title |
|------|---------------------------------------------------------------------|
| 1247 | vuejs - 解决post请求变成option的请求问题. |
| 1246 | ruby - windows下的安装 |
| 1245 | vuejs - mixin的基本用法 |
| 1244 | android - 在 view pager中的 webview, 切换时, 会闪烁的问题。 |
| 1243 | 证照 - 如何开具无行贿犯罪记录证明 |
| 1242 | java - ant的基本用法 |
| 1241 | java - eclipse的基本用法 |
| 1240 | java - eclipse 中, 启动一个项目之前, 要设置好 lib 的各种依赖 |
| 1239 | java - linux下启动tomcat |
| 1238 | rspec 不再输出 警告信息 : --deprecation-out temp |
| 1237 | android - android studio的最有用快捷键： 补全代码后直接跳到行末 : ctrl + shift + enter |
| 1235 | rails - 调用oracle存储过程 |
| 1236 | android - 使用tablayout + view pager 实现 底部tab (bottom tab) |
| 1234 | mysql 使用client 命令行的时候, 使用utf-8编码 |
| 1233 | rails - 调用mysql存储过程 |
| 1232 | mysql - 存储过程的入门 |

如何Debug

浏览器环境下的javascript，实际上有两个天生缺陷：

1. 不严谨。不同浏览器的js实现上会略有不同。这个问题在android, ios上也一样。
2. 不是严格意义上的计算编程语言。有语法漏洞。例如 “==”

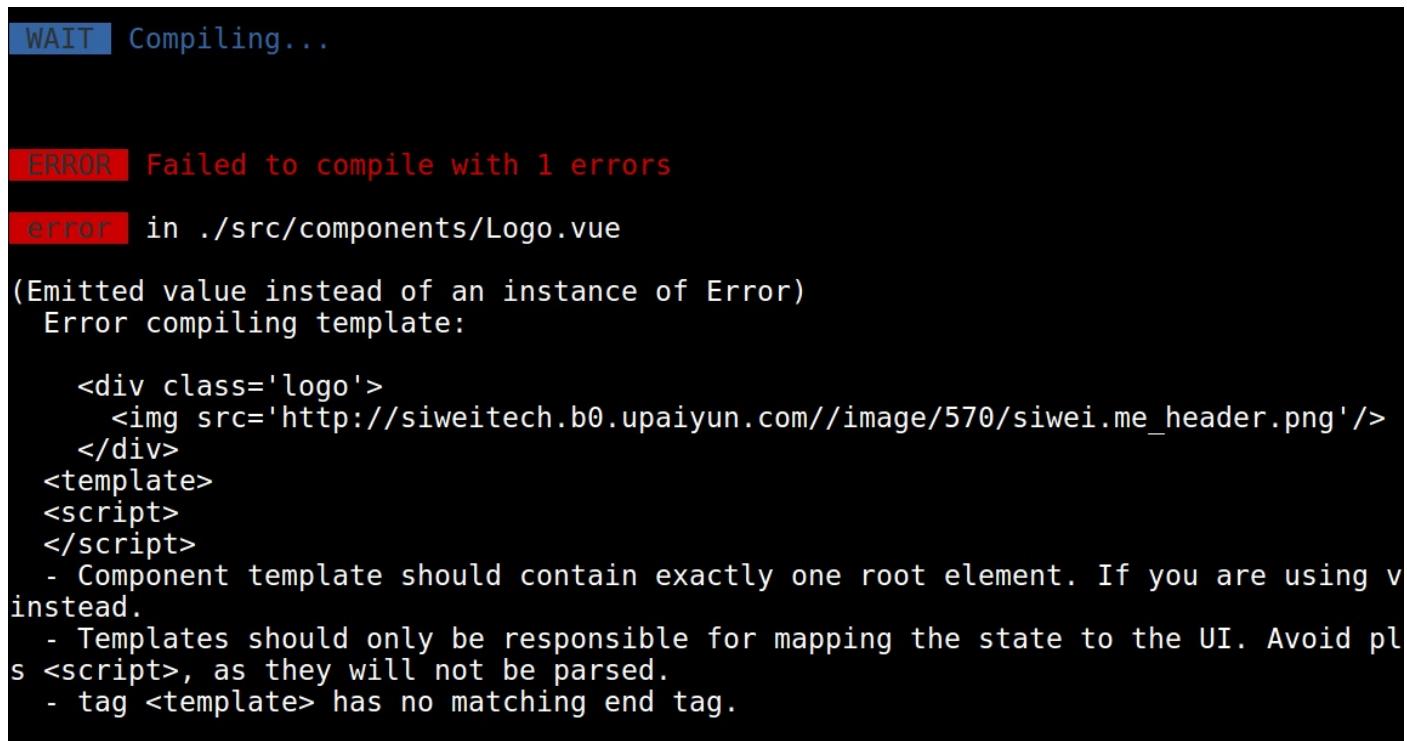
所以，我们要驾驭好JS语言，就要知道如何有效的Debug。

时刻留意 vue server

我们开发时的命令：

```
1. $ npm run dev
```

会开启一个“开发服务器”，这个开发服务器的后台，我们要时刻留意输出。有时候我们把代码写错了，导致Vue.js无法编译，前台就会一片空白，还没有任何出错提示。



```

WAIT Compiling...

ERROR Failed to compile with 1 errors
error in ./src/components/Logo.vue
(Emitted value instead of an instance of Error)
Error compiling template:

<div class='logo'>
  <img src='http://siweitech.b0.upaiyun.com//image/570/siwei.me_header.png' />
</div>
<template>
<script>
</script>
- Component template should contain exactly one root element. If you are using v
instead.
- Templates should only be responsible for mapping the state to the UI. Avoid pl
s <script>, as they will not be parsed.
- tag <template> has no matching end tag.

```

上面的错误提示很好理解，说“编译时出现错误”，也给出了错误的详细位置。

看developer tools 提出的日志

无论是 chrome, safari 还是firefox，以及 IE 7+，都带有这个工具。特别好用。任何时候

页面空白了，都要首先看它，而不是问别人：“页面怎么不动了？”

由于JS代码不是特别严谨，所以给出的错误提示也都很概括。我们可以做个对比：

- JSP 错误可以精确到某行
- PHP 错误可以精确到某行
- Rails 错误可以精确到某行
- Vue.js, Angular, Titanium 等JS框架：错误可以精确到“某个文件”。

这是由于，所有的JS框架的表现层，都是“框架怪胎”，是一种跟js语言环境妥协的代码。出了问题很难定位到最底层的根源。

而 JSP, PHP, Rails ERB，则是“正常框架”，出了问题可以直接找到最底层。

所以，我们要有一定的经验来Debug。来理解错误日志。

例如下图：

```

Console
top
Preserve log
Object {blogs: Array[1231]}          BlogList.vue?3b91:39
Object {result: Object}               Blog.vue?e337:26
[Vue warn]: Property or method "博客详情页" is not defined on the instance but referenced during render. Make sure to declare reactive data properties in the data option. vue.esm.js?65d7:434
found in
--> <Blog> at /workspace/test_vue_0613/src/components/Blog.vue
      <App> at /workspace/test_vue_0613/src/App.vue
      <Root>

```

```

vue.esm.js?65d7:434 [Vue warn]: Property or method "博客详情页" is not defined
1. on
2. the instance but referenced during render. Make sure to declare reactive data
3. properties in the data option.
4.
5. found in
6.
7. --> <Blog> at /workspace/test_vue_0613/src/components/Blog.vue
8.      <App> at /workspace/test_vue_0613/src/App.vue
9.      <Root>

```

- `vue.esm.js?65d7:434` 表示错误的来源。这个文件一般人不知道来自于哪里，我们暂且认为它来自于临时产生的文件，或者虚拟js机中。
- `Property or method "博客详情页" is not defined ...` 这句话提示了错误的原因。
- `found in ... <Blog> at ...` 这里则是调用栈，可以看出，文件是从下调用到最上面的。

问题出在最上面的文件. 但是没有给出错误的行数.

查看页面给出的错误提示(来自于dev server)

如下图:

```

ERROR in ./~/vue-loader/lib/template-compiler?{"id":"data-v-55ccd29a"}!./~/vue-loader/lib/selector.js?type=template&index=0!./src/comp
(Emitted value instead of an instance of Error)
Error compiling template:
1245<div class='logo'>用法
    <img src='http://siweitech.b0.upaiyun.com//image/570/siwei.me_header.png' />
1246</div>d - 在 view pager中的 webview, 切换时, 会闪烁的问题。
<template>
1247 Component template should contain exactly one root element. If you are using v-if on multiple elements, use v-else-if to chain them
- tag <template> has no matching end tag.
1248 java -ant的基本用法
@ ./src/components/Logo.vue 6:2-177
@ ./~/babel-loader/lib!./~/vue-loader/lib/selector.js?type=script&index=0!./src/components/BlogList.vue
@ ./src/components/BlogList.vue
@ ./src/router/index.js 一个项目之前, 要设置好 lib 的各种依赖
@ ./src/main.js
@ multi ./build/dev-client ./src/main.js

1238 rspec 不再输出警告信息: --deprecation-out temp

```

1. **Error compiling template:**
- 2.
3.

- 4. - 5. </div>
- 6. <template>
- 7. <script>
- 8. </script>
 - Component template should contain exactly one root element. If you are
- 9. using v-if
- 10. on multiple elements, use v-else-if to chain them instead.
- 11. - Templates should only be responsible for mapping the state to the UI. Avoid
- 12. placing tags with side-effects in your templates, such as <script>, as they
- 13. will not be parsed.
- 14. - tag <template> has no matching end tag.

这里 **Error compiling template:** 给出了提示, 错误是由于 模板在被编译时产生的.

下面给出的HTML代码则是出错的点.

1. @ ./src/components/Logo.vue 6:2-177
 - @ ./~/babel-loader/lib!./~/vue-loader/lib/selector.js?
2. type=script&index=0!./src/components/BlogList.vue
3. @ ./src/components/BlogList.vue
4. @ ./src/router/index.js
5. @ ./src/main.js

```
6. @ multi ./build/dev-client ./src/main.js
```

这里是调用栈. 可以看到, `@ ./src/components/Logo.vue 6:2-177`, 所以错误在于Logo.vue, 第六行第二列.

基本命令

这个命令都是 `vue-cli` 提供的。可以认为是“webpack + vuejs”的结合。

建立新项目

```
1. $ vue init webpack my_blog
```

这个命令会建立好一个文件夹。具体的说明，见《Webpack下的Vuejs项目文件结构》章节。

安装所有的第三方包

```
1. $ npm install
```

这个命令是根据“package.json”文件中定义的内容，来安装所有用到的第三方js库。所有的文件都会安装到“node_module”目录下。

着急的同学还可以输入 `--verbose` 命令，来查看运行细节：

```
1. $ npm install --verbose
```

运行结果如下：

```
1. $ npm install --verbose
2. npm info it worked if it ends with ok
3. npm verb cli [ 'D:\\nodejs\\node.exe',
4. npm verb cli   'D:\\nodejs\\node_modules\\npm\\bin\\npm-cli.js',
5. npm verb cli   'install',
6. npm verb cli   '--verbose' ]
7. npm info using npm@6.1.0
8. npm info using node@v10.5.0
9. npm verb npm-session d1e752145cbb60ba
10. npm info lifecycle test_vue_0613@1.0.0~preinstall: test_vue_0613@1.0.0
11. npm timing stage:loadCurrentTree Completed in 1609ms
12. npm timing stage:loadIdealTree:cloneCurrentTree Completed in 12ms
13. npm timing stage:loadIdealTree:loadShrinkwrap Completed in 681ms
14. ...
```

这样出现问题的时候，就很容易知道卡在哪里了。

在本地运行

使用下列命令：

```
1. $ npm run dev
```

默认就会在 localhost 的 8080 端口，启动一个小型的 web 服务器。 性能可以完全满足开发使用，还具备代理转发等功能。

源代码发生改变的时候，服务器也会自动重启。（偶尔需要手动重启）

运行的输入如下所示：

```
1. dashi@i5-16g MINGW64 /d/workspace/vue_js_lesson_demo (master)
2. $ npm run dev
3.
4. > test_vue_0613@1.0.0 dev D:\workspace\vue_js_lesson_demo
5. > node build/dev-server.js
6.
7. [HPM] Proxy created: /api -> http://siwei.me
8. [HPM] Proxy rewrite rule created: "^/api" ~> ""
9. > Starting dev server...
10. DONE Compiled successfully in 2373ms10:55:18
11.
12. > Listening at http://localhost:8080
13.
14. WAIT Compiling...11:02:14
15.
16. DONE Compiled successfully in 213ms11:02:14
17.
18. WAIT Compiling...11:06:09
19.
20. DONE Compiled successfully in 117ms11:06:09
21.
22. WAIT Compiling...11:06:26
23.
24. DONE Compiled successfully in 103ms11:06:26
25. ...
```

打包编译

```
1. $ npm run build
```

该命令用来把 “src” 目录下的所有文件，都打包成 “webpack”的标准文件，供部署使用。 具体的内容见 《打包和部署》 这一章节。

进阶

虽然是进阶知识，但是也是必学内容。

js的作用域 与 this

无论是 javascript, 还是 emscript, 变量的作用域都属于高级知识。 我们想考察一个js程序员的水平如何，可以直接用作用域来提问。

同时，我们在实际的开发中发现，很多js/emscript 的新人，对于作用域和 this 都很含混，所以这里要单独的提一下。

作用域

无论是 javascript, 还是 emscript, 对于作用域的使用基本是一样的。 后者更加严密一些。 我们看几个例子。

1. 全局变量，可以直接引用。

```
1. //全局变量 a:  
2. var a = 1;  
3. function one() {  
4.     console.info(a)  
5. }
```

打印结果是 1

2. 函数内的普通变量

```
1. function two(a ){  
2.     console.info('a is' + a)  
3. }
```

运行：

two(2) 打印结果是 : a is 2

3. 普通函数可以对全局变量做赋值。 如下图所示：

```
1. var a = 1;  
2. function four(){  
3.     console.info(' in four, before a=4: ' + a)  
4.     if(true) a = 4;  
5.     console.info(' in four, after a=4: ' + a)  
6. }
```

运行：

```
1. four(4)
```

结果：

```
1. in four, before a=4: 1      ( 这个是符合正常的scope逻辑的。 )
2. in four, after a=4: 4      ( 这个也是符合 )
```

再运行：`console.info(a)`，可以看到输出：4 说明 全局变量 a 在 four()函数中已经被发生了永久的变化。

4. 通过元编程定义的函数

```
1. var six = ( function(){
2.   var foo = 6;
3.   return function(){
4.     return foo;
5.   }
6. })
7. )();
```

在上述代码中，js解析器会先运行（忽略最后的`()`）：

```
1. var temp = function(){
2.   var foo = 6;
3.   return function(){
4.     return foo;
5.   }
6. }
```

然后再运行：`var six = (temp)()`，所以，six 就是：

```
1. function(){
2.   return foo;
3. }
```

上面的 `foo` 就是来自于方法最开始定义的 `var foo = 6`，而这个变量的定义，是在一个 `function()` 中的。所以它不是一个全局变量。

所以，如果我们在console中输入 `foo`，会看到报错消息：`Uncaught ReferenceError: foo`

```
is not defined
```

5. 通过元编程定义的函数中的变量，不会污染全局变量。

```
1. var foo = 1;
2.
3. var six = ( function(){
4.     var foo = 6;
5.     return function(){
6.         console.info("in six, foo is: " + foo);
7.     }
8. }
9. )();
```

在上面的代码中，我们先定义了一个 全局变量 `foo`，再定义了一个方法 `six`，里面定义了一个临时方法 `foo`。并且进行来了一些操作。

运行：

```
1. six() // 返回： in six, foo is: 6
2. foo    // 返回： 1
```

this

对于 `this` 的使用，在这里 <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this> 指出了对于javascript中this的详细用法。

在emscript中也基本是一样的。

简单的说，大家只要记得，`this` 指的是当前作用域的对象实例就好了。

```
1. var apple = {
2.   color: 'red',
3.   show_color: function() {
4.     return this.color
5.   }
6. }
```

我们输入 `apple.show_color` 就可以看到输出 `red`。这里的 `this` 指的就是 `apple` 变量。

实战经验

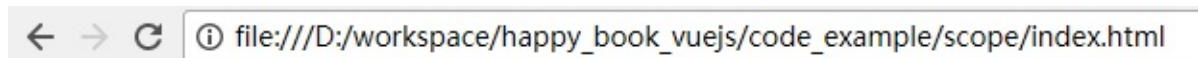
1. 在Vue的方法定义中容易用错。

当我们发现 代码看起来没问题， 但是console 总报错说 xx undefined 时， 十有八九就是 忘记加 this 了。

例如：

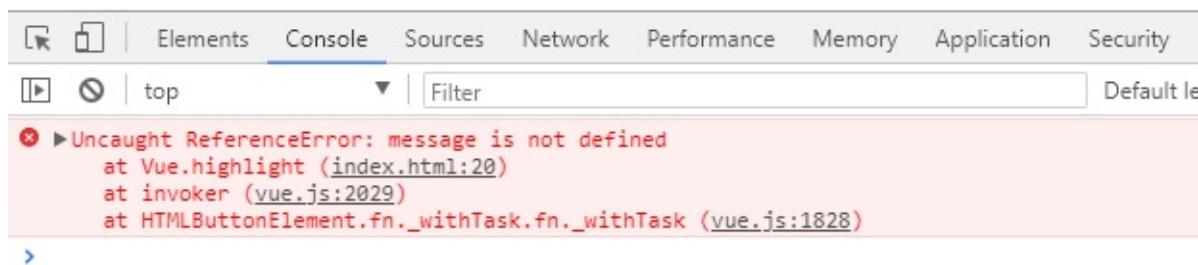
```
1. <html>
2.   <head>
3.     <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4.   </head>
5.   <body>
6.     <div id='app'>
7.       {{ message }}
8.       <br/>
9.       <button v-on:click='highlight' style='margin-top: 50px'>真的吗</button>
10.    </div>
11.
12.    <script>
13.      var app = new Vue({
14.        el: '#app',
15.        data: {
16.          message: 'this 是很重要的。 不要忘记它哟~'
17.        },
18.        methods: {
19.          highlight: function() {
20.            // 报错： message is not defined.
21.            message += '是的， 工资还会涨~!'
22.
23.            // 正确的代码应该是：
24.            // this.message += '是的， 工资还会涨~!'
25.          }
26.        }
27.
28.      })
29.    </script>
30.  </body>
31. </html>
```

使用浏览器加载上述代码，我们会发现报错了：



this是很重要的。不要忘记它哟~

真的吗



上面的代码中，`message += ...` 那一行中，`message` 是当前的vue的实例的一个“property”(属性)，而我们如果希望在 `methods` 中引用这个属性的话，就需要用 `this.message` 才对。这里的 `this` 对应的就是 `var app = new Vue()` 中定义的 `app`。

2. 在发起http请求时容易用错

我们看下面的例子，是一段代码片段：

```

1. new Vue({
2.   data: {
3.     cities: [...]
4.   },
5.   methods: {
6.     my_http_request: function(){
7.       let that = this
8.       axios.get('http://mysite.com/my_api.do')
9.       .then(function(response){
10.         // 这里不能使用 this.cities 来赋值
11.         that.cities = response.data.result
12.       })
13.     }
14.   }
15. })

```

在上面的代码中，我们定义了一个属性：`cities`，定义了一个方法：`my_http_request`。该方法会向远程发起一个请求，然后把返回的response中的值赋给 `cities`。

可以在上面代码中看到，需要先在 `axios.get` 之前，定义一个变量 `let that = this`。这个时候，`this` 和 `that` 都处于“Vue”的实例中。

但是在 `axios.get(..).then()` 函数中，就不能再使用 `this` 了。因为在 `then(...)` 中，这是个function callback，其中的 `this` 会代表这个 http request event. 这是个事件。

所以，只能用这样的方式。

（如果使用了 emscript 的 `=>` 的话，就可以避免上述问题）

3. 在event handler中容易用错

道理同上。

Mixin

Mixin是一种更好的复用代码的模式.

我们知道 java , Object C 中的 interface , implements , extends 等关键字的意义, 就是为了让代码可以复用、继承.

但是这几种方法, 都理解起来很不直观, 给人一种拐弯抹角的感觉. 特别是像我这样很不习惯“设计模式”的人。

在js, ruby等动态语言中, 我们如果要复用代码的话, 直接使用 mixin 就好了.

Mixin 的 概念

Mixin 实际上是利用语言的特性(关键字), 以更加简洁易懂的方式, 实现了“设计模式”中的“组合模式”。可以定义一个公共的类, 这个类就叫做“mixin”. 然后让其他的类, 通过“include”这样的语言特性, 来包含mixin, 直接具备了 mixin 所具备的各种方法。

我们下面看一下在 Vuejs 中如何使用mixin 这种强大的工具。

建立一个Mixin文件

可以在 `src/mixin` 目录下创建, 例如:

文件: `src/mixin/common_hi.js` :

```
1. export default {
2.   methods: {
3.     hi: function(name){
4.       return "你好, " + name;
5.     }
6.   }
7. }
```

使用

Mixin使用起来很简单, 在对应的 js文件, 或者 vue文件的 `<script>` 代码中引用即可.

例如, 新建一个vue文件:

`src/components/SayHiFromMixin.vue` , 内容如下:

```

1. <template>
2.   <div>
3.     {% raw %}{{% endraw %}hi("from view")}
4.   </div>
5. </template>
6.
7. <script>
8. import CommonHi from '@/mixins/common_hi.js'
9. export default {
10.   mixins: [CommonHi],
11.   mounted() {
12.     alert( this.hi('from script code'))
13.   }
14. }
15. </script>
```

注意:

- 使用的时候, `mixins: [CommonHi]` 这里的是中括号, 表示是数组.
- 在js代码中调用的话, 需要带有`this`关键字, 例如: `this.hi()`

路由如下:

```

1. import SayHiFromMixin from '@/components/SayHiFromMixin'
2.
3. export default new Router({
4.   routes: [
5.     {
6.       path: '/say_hi_from_mixin',
7.       name: 'SayHiFromMixin',
8.       component: SayHiFromMixin
9.     }
10.   ]
11. })
```

运行效果如下:



使用Computed properties(计算得到的属性)和watchers(监听器)

很多时候，我们在页面上想要显示某个变量的值时，都需要经过一些计算，例如：

```

1. <div id="example">
2.   {{some_string.split(',') .reverse() .join('-')}}
3. </div>
```

越是复杂，到后期越容易出错。

这个时候，我们就需要一种机制，可以方便的创建这样的通过计算得来的数据。

所以，Computed Properties 就是我们的解决方案。

典型例子

```

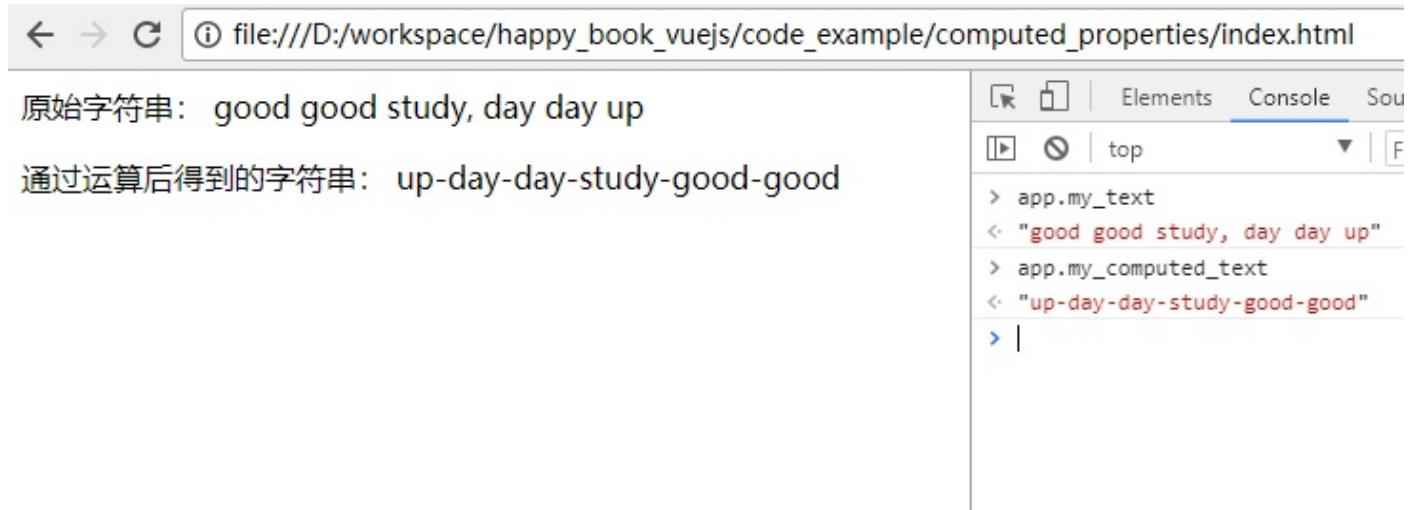
1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.     <p> 原始字符串： {{my_text}} </p>
8.     <p> 通过运算后得到的字符串： {{my_computed_text}} </p>
9.   </div>
10.  <script>
11.    var app = new Vue({
12.      el: '#app',
13.      data: {
14.        my_text: 'good good study, day day up'
15.      },
16.      computed: {
17.        my_computed_text: function(){
18.          // 先去掉逗号，然后按照空格分割成数组，然后翻转，并用'-'来连接
19.          return this.my_text.replace(',', ' ') .split(''
20.            ).reverse() .join('-')
21.        }
22.      }
23.    }
```

computed properties

```
23.     })
24.   </script>
25. </body>
26. </html>
```

可以看到，上面的关键代码是，在 Vue 的构造函数中，传入一个 `computed` 的段落。

使用浏览器运行后，可以看到结果如下图所示：



我们也打开 `console` 来查看。

输入

```
1. > app.my_text
```

会得到：“good good study, day day up”

输入

```
1. > app.my_computed_text
```

会得到转换后的：“up-day-day-study-good-good”

Computed Properties 与 普通方法的区别。

根据上面的例子，我们可以使用 普通方法来实现：

```
1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
```

computed properties

```
4.  </head>
5.  <body>
6.      <div id='app'>
7.          <p> 原始字符串： {{my_text}} </p>
8.          <p> 通过运算后得到的字符串： {{ raw }}{{ endraw }}{{ my_computed_text() }}</p>
9.      </div>
10.     <script>
11.         var app = new Vue({
12.             el: '#app',
13.             data: {
14.                 my_text: 'good good study, day day up'
15.             },
16.             methods: {
17.                 my_computed_text: function(){
18.                     return this.my_text.replace(',', '').split(' ')
19.                         .reverse().join('-') + ', 我来自于 function, 不是computed'
20.                 }
21.             }
22.         )
23.     </script>
24. </body>
25. </html>
```

上面的代码，运行后如下图所示：



可以发现，他们达到的效果是一样的。

他们的区别在于： 使用computed properties的方式，会把结果“缓存”起来。 每次调用对应的computed properties时，只要对应的依赖数据没有改动，那么就不会变化。

而使用“function”实现的版本，则不存在缓存问题。 每次都会重新计算对应的数值。

所以， 我们需要按照实际情况，来选择是使用“computed properties”， 还是使用普通

function的形式。

watched property

Vue.js 中的property(属性)， 是可以要么根据计算发生变化 (computed) ， 要么根据监听 (watch) 其他的变量的变化而发生变化

我们看一下，如何根据监听 (watch) 其他的变量而自身发生变化的例子，如下；

```
1.  <html>
2.  <head>
3.      <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4.  </head>
5.  <body>
6.      <div id='app'>
7.          <p>
8.              我所在的城市： <input v-model='city' /> (这是个watched property)
9.          </p>
10.         <p>
11.             我所在的街道： <input v-model='district' /> (这是个watched property)
12.         </p>
13.         <p> 我所在的详细一些的地址： {{full_address}} (每次其他两个发生变化，这里就会跟着变化) </p>
14.     </div>
15.     <script>
16.         var app = new Vue({
17.             el: '#app',
18.             data: {
19.                 city: '北京市',
20.                 district: '朝阳区',
21.                 full_address: "某市某区"
22.             },
23.             watch: {
24.                 city: function(city_name){
25.                     this.full_address = city_name + ' - ' + this.district
26.                 },
27.                 district: function(district_name){
28.                     this.full_address = this.city + ' - ' + district_name
29.                 }
30.             }
31.         })
32.     </script>
```

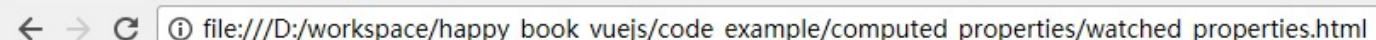
computed properties

```
33.      </script>
34.  </body>
35.  </html>
```

在上面的代码中，可以看到，`watch: { city: ..., district: ... }`，表示，`city` 和 `district` 都已经被监听了，这两个都是 `watched properties`。

只要 `city` 和 `district` 发生变化，`full_address` 就会跟着变化。

我们用浏览器打开上面的代码，如下图所示，此时由于 `city` 和 `district` 还没有发生变化，所以 `full_address` 的值还是“某市某区”：



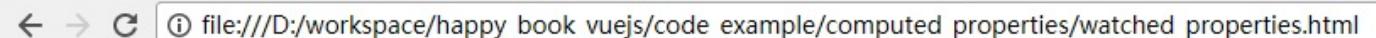
← → C ⓘ file:///D:/workspace/happy_book_vuejs/code_example/computed_properties/watched_properties.html

我所在的城市：北京市 (这是个watched property)

我所在的街道：朝阳区 (这是个watched property)

我所在的详细一些的地址：某市某区 (每次其他两个发生变化，这里就会跟着变化)

当我在“街道”的输入框，后面加上“望京街道”几个字后，可以看到，下面的“详细地址”，就发生了变化。如下图所示：



← → C ⓘ file:///D:/workspace/happy_book_vuejs/code_example/computed_properties/watched_properties.html

我所在的城市：北京市 (这是个watched property)

我所在的街道：朝阳区 望京街道 (这是个watched property)

我所在的详细一些的地址：北京市-朝阳区 望京街道 (每次其他两个发生变化，这里就会跟着变化)

使用computed 会比watch 更加简洁

上面的例子，我们可以使用 `computed` 来改写，如下图所示：

```
1.  <html>
2.  <head>
```

```

3.      <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4.  </head>
5.  <body>
6.      <div id='app'>
7.          <p>
8.              我所在的城市： <input v-model='city' />
9.          </p>
10.         <p>
11.             我所在的街道： <input v-model='district' />
12.         </p>
13.         <p> 我所在的详细一些的地址： {{full_address}} （这是使用computed 实现的版本）
14.     </p>
15. 
16.     </div>
17.     <script>
18.         var app = new Vue({
19.             el: '#app',
20.             data: {
21.                 city: '北京市',
22.                 district: '朝阳区',
23.             },
24.             computed: {
25.                 full_address: function(){
26.                     return this.city + this.district;
27.                 }
28.             }
29.         })
30.     </script>
31. </body>
32. </html>

```

可以看到， 方法少了一个， `data` 中定义的属性也少了一个， 简洁了不少。 代码简洁， 维护起来就容易（代码量越少， 程序越好理解）

为 computed property 的setter（赋值函数）

原则上来说， `computed property` 是根据其他的值， 经过计算得来的。 是不应该被修改的。

不过在开发中， 确实有一些情况， 需要对 “`computed property`” 做修改， 同时影响某些对应的属性。 （过程跟上面是相反的）。

我们看下面的代码：

```

1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.     <p>
8.       我所在的城市： <input v-model='city' />
9.     </p>
10.    <p>
11.      我所在的街道： <input v-model='district' />
12.    </p>
13.    <p> 我所在的详细一些的地址： <input v-model='full_address' /> </p>
14.
15.  </div>
16.  <script>
17.    var app = new Vue({
18.      el: '#app',
19.      data: {
20.        city: '北京市',
21.        district: '朝阳区',
22.      },
23.      computed: {
24.        full_address: {
25.          get: function(){
26.            return this.city + " - " + this.district;
27.          },
28.          set: function(new_value){
29.            this.city = new_value.split('-')[0]
30.            this.district = new_value.split('-')[1]
31.          }
32.        }
33.      }
34.    })
35.  </script>
36. </body>
37. </html>

```

可以看出，上面代码中，有这样一段：

```

1. computed: {
2.   full_address: {

```

```
3.     get: function(){
4.         return this.city + " - " + this.district;
5.     },
6.     set: function(new_value){
7.         this.city = new_value.split('-')[0]
8.         this.district = new_value.split('-')[1]
9.     }
10.    }
11. }
```

可以看出，上面的 `get` 代码段，就是原来的代码内容。而 `set` 端中，则定义了，如果 `computed property` (也就是 `full_address`) 发生变化的时候，`city` 和 `district` 的值应该如何变化。

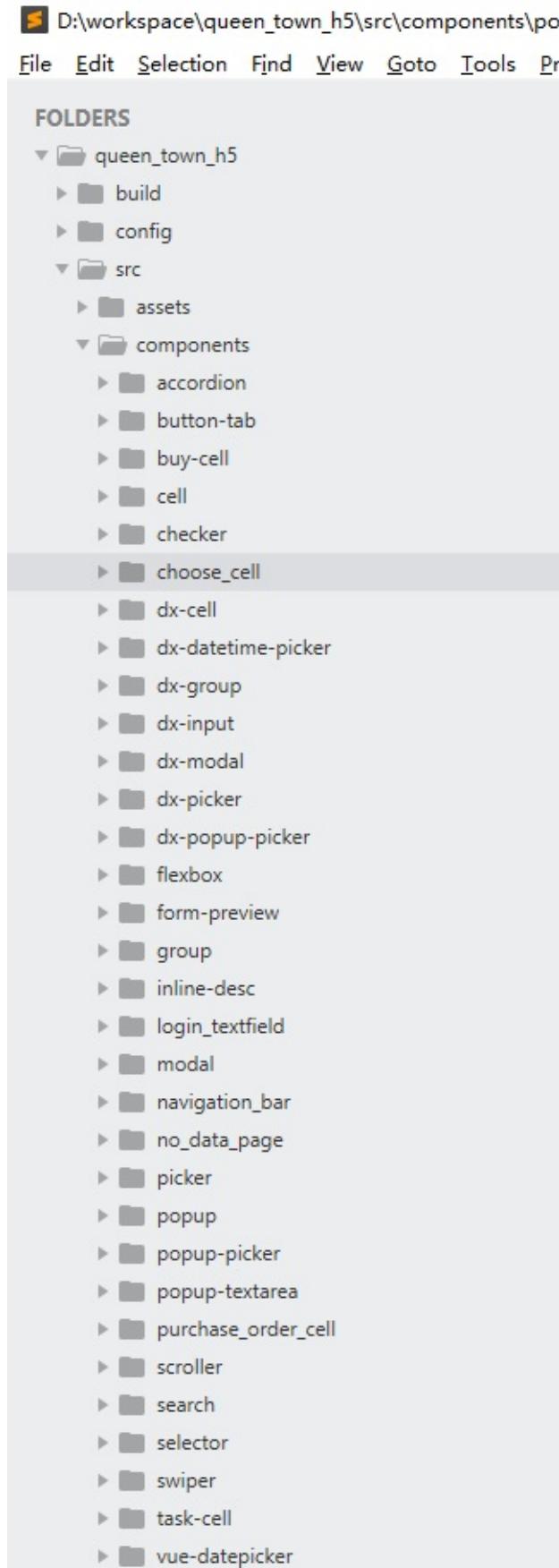
用浏览器打开后，我们在“最下方的输入框”中，后面输入一些字，可以看到，对应的“街道”发生了变化：



Component (组件) 进阶

Component 是非常常见的，在我们的Web开发中，只要是生产环境的项目，就一定会有Component.

下面就是我们的一个实际项目中的例子，这个项目我们只做了两个月，里面就发展到了32个component. 如下图所示：



很多时候，我们甚至会看到一个 component 中嵌套着另一个，这个 component 再嵌套另外5个....

例如：

popup-picker 这个 component 中，看起来是这样的：

```

1. <template>
2.   <div>
3.     <popup
4.       class="vux-popup-picker"
5.       :id="`vux-popup-picker-${uuid}`"
6.       @on-hide="onPopupHide"
7.       @on-show="onPopupShow">
8.
9.       <picker
10.         v-model="tempValue"
11.         @on-change="onPickerChange"
12.         :columns="columns"
13.         :fixed-columns="fixedColumns"
14.         :container="'#vux-popup-picker-' + uuid"
15.         :column-width="columnWidth">
16.       </picker>
17.     </popup>
18.   </div>
19. </template>
20. <script>
21. import Picker from '../picker'
22. import Popup from '../popup'
23. ...
24. </script>

```

可以看到，这个component中，还包含了另外两个，一个是 `popup`，一个是 `picker`。

这个时候，新人往往会觉得眼花缭乱。如果看到 `this.$emit`，就更晕了。

所以，要做好实际项目，同学们一定要学好本章。

Component 命名规则

每个component 的命名，官方建议使用 小写字母 + 横线的形式，例如：

```
1. Vue.component('my-component-name', { /* ... */ })
```

这个是符合W3C的规范的。

也可以定义为：

```
1. Vue.component('MyComponentName', { /* ... */ })
```

这个时候，可以使用 `<MyComponentName/>` 来调用，也可以使用 `<my-component-name/>` 来调用。

Prop 命名规则

同 component，建议使用 小写字母 + ‘-’ 连接。

Prop 可以有多种类型。

下面是一个例子，可以看出，一个component的 prop 可以有多种类型，包括：字符串，数字，bool，数组，和 Object.

```
1. props: {
2.   title: "Triple Body",
3.   likes: 38444,
4.   isPublished: true,
5.   commentIds: [30, 54, 118, 76],
6.   author: {
7.     name: "Liu Cixin",
8.     sex: "male"
9.   }
10. }
```

可以动态为 prop 赋值

例如，这是个静态的赋值：

```
1. <blog-post title="Vue.js的学习笔记"></blog-post>
```

这是个动态的赋值：

```
1. // 1. 在script中定义
2. post = {
3.   title: 'Triple body',
4.   author: {
5.     name: "Big Liu",
6.     sex: 'male'
7.   }
8. }
```

```

9.
10. // 2. 在模板中使用。
11. <blog-post v-bind:title="post.title + 'by' + post.author.name"></blog-post>

```

赋值的时候，只要是符合标准的类型，都可以传入（包括String, bool, array 等）。

使用Object来为Prop赋值

假设，我们定义有：

```

1. post = {
2.   author: {
3.     name: "Big Liu",
4.     sex: 'male'
5.   }
6. }

```

那么，下面的代码：

```
1. <blog-post v-bind:author></blog-post>
```

等价于：

```
1. <blog-post v-bind:name="author.name" v-bind:sex="author.sex"></blog-post>
```

单向的数据流

当“父页面”引用一个“子组件”时，如果父页面中的变量发生了变化，那么对应的“子组件”也会发生页面的更新。

反之则不行。

Prop的验证

Vue.js 的组件的Prop，是可以被验证的。如果验证不匹配，浏览器的 console就会弹出警告(warning)。这个对于我们的开发非常有利。

我们下面的代码：

```
1. Vue.component('my-component', {
```

```

2.   props: {
3.     name: String,
4.     sexandheight: [String, Number],
5.     weight: Number,
6.     sex: {
7.       type: String,
8.       default: 'male'
9.     }
10.   }
11. })

```

可以看得出，

name：必须是字符串
sexandheight：必须是个数组。第一个元素是String，第二个元素是Number
weight：必须是Number
sex：是个String，默认值是‘male’

支持的类型有：

- String
- Number
- Boolean
- Array
- Object
- Date
- Function
- Symbol

Non Prop (非Prop) 的属性

很多时候，component的作者无法预见到应该用哪些属性，所以Vue.js在设计的时候，就支持让component接受一些没有预先定义的 prop. 例如：

```

1. Vue.component('my-component', {
2.   props: ['title']
3. })
4.
5. <my-component title='三体' second-title='第二册：黑暗森林'></my-component>

```

上面的 `title` 就是预先定义的“Prop”，`second-title` 就是“非Prop”

我们想传递一个 non-prop，非常简单，prop 怎么传，non-prop 就怎么传。

对于Attribute的合并和替换

如果component中定义了一个 attribute，例如：

```
1. <template>
2.   <div color="red">我的最终颜色是蓝色</div>
3. </template>
```

如果在引用了这个“子组件”的“父页面”中，也定义了同样的attribute，例如：

```
1. <div>
2.   <my-component color="blue"></my-component>
3. </div>
```

那么，父页面传递进来的 `color="blue"` 就会替换子组件中的 `color="red"`

但是，对于 `class` 和 `style` 是例外的。对于上面的例子，如果attribute换成 `class`，那么最终component中的class的值，就是 “red blue” (发生了合并)

避免 子组件的attribute 被父页面 所影响

根据上面的小结，我们知道了“父页面”的值总会“替换”“子组件”中的同名attribute。

如果不希望有这样的情况的话，我们就可以在定义 component 的时候，这样做：

```
1. Vue.component('my-component', {
2.   inheritAttrs: false,
3.   // ...
4. })
```

Slot

作为对Component的补充，Vue.js 增加了 Slot 这个功能。

普通的Slot

我们从具体的例子来说明。

```
1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.     <study-process>
8.       我学习到了Slot 这个章节。
9.     </study-process>
10.    </div>
11.    <script>
12.      Vue.component('study-process', {
13.        data: function () {
14.          return {
15.            count: 0
16.          }
17.        },
18.        template: '<div><slot></slot></div>'
19.      })
20.      var app = new Vue({
21.        el: '#app',
22.        data: {
23.        }
24.      })
25.    </script>
26.  </body>
27. </html>
```

从上面的代码从，我们可以看到，我们先是定义了一个 component：

```
1. Vue.component('study-process', {
2.   data: function () {
```

```

3.     return {
4.       count: 0
5.     },
6.   },
7.   template: '<div><slot></slot></div>'
8. })

```

在这个component的template中，是这样的：

```
1. template: '<div><slot></slot></div>'
```

这里就使我们定义的 slot.

然后，我们在调用这个 component的时候，这样：

```

1. <study-process>
2.   我学习到了Slot 这个章节。
3. </study-process>

```

所以，“我学习到了Slot 这个章节。”就好像一个参数那样传入到了 component中， component发现自身已经定义了 slot，就会把这个字符串放到slot的位置，然后显示出来。

如下图所示：



named slot

也就是带有名字的slot .

很多时候我们可能需要多个slot. 看下面的例子：

```

1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.     <study-process>
8.       <p slot='slot_top'>
9.         Vue.js 比起别的框架真的简洁好多！
10.      </p>

```

```

11.
12.          我学习到了 Slot 这个章节。
13.
14.          <h5 slot='slot_bottom'>
15.              再也不怕H5 项目了~
16.          </h5>
17.          </study-process>
18.      </div>
19.      <script>
20.          Vue.component('study-process', {
21.              template: '<div>' +
22.                  '<slot name="slot_top"></slot>' +
23.                  '<slot></slot>' +
24.                  '<slot name="slot_bottom"></slot>' +
25.                  '</div>'
26.          })
27.          var app = new Vue({
28.              el: '#app',
29.              data: {
30.                  }
31.          })
32.      </script>
33.  </body>
34. </html>

```

上面的代码中， 我们定义了这样的 component：

```

1.  Vue.component('study-process', {
2.      template: '<div>' +
3.          '<slot name="slot_top"></slot>' +
4.          '<slot></slot>' +
5.          '<slot name="slot_bottom"></slot>' +
6.          '</div>'
7.  })

```

其中的 `<slot name="slot_top"></slot>` 就是一个named slot (具备名字的slot) ,这样，在后面对于 component的调用中：

```

1.  <p slot='slot_top'>
2.      Vuejs 比起别的框架真的简洁好多！
3.  </p>

```

就会渲染在对应的位置了。

slot 的默认值

我们可以为 slot 加上默认值。这样当“父页面”没有指定某个slot的时候，就会显示这个默认值了。

例如：

```
1. <slot name="slot_top">这里 top slot的默认值 </slot>
```

Vuex

Vuex 是 状态管理工具. 跟React中的Redux相似，但是更加简洁直观。

简单的说，Vuex 帮我们管理 “全局变量”，供任何页面在任何时刻使用.

跟其他语言中的“全局变量”相比， 使用Vuex 的优点是：

1. Vuex中的变量的状态是响应式的。当某个组件读取这个变量时，只要Vuex中的变量发生变化，那么对应的组件就会发生变化（类似于双向绑定）
2. 用户或者程序无法直接改变Vuex中的变量。必须通过Vuex提供的接口来操作. 这个接口就是通过 “commit mutation”来实现的。

Vuex 非常重要. 不管是大项目还是小项目, 都有用到它的时候. 我们必须会用.

完整官方文档: <https://vuex.vuejs.org/zh-cn/getting-started.html>

Vuex 的内容很庞大，用到了比较烧脑的设计模式（这是由于javascript语言本身不够严谨和成熟决定的），所以我不打算带同学们把源代码和实现机理详细的学一遍。大家只要可以娴熟的使用就行了。

下面，我们以一个例子来说明如何使用。

正常使用的顺序

假设, 我们有两个页面：“页面1” 和“页面2” , 共同使用一个变量: counter. 页面1对“counter” + 1 后，页面2的值也会发生变化.

1. 修改 `package.json`

增加 `vuex` 的依赖声明，如下：

```
1. "dependencies": {
2.   "vuex": "^2.3.1"
3. },
```

如果不确定你的 vuex 用什么版本, 就先手动安装一下：

```
1. $ npm install vuex --verbose
```

然后看安装过来的版本号就可以了.

2. 新建store文件

文件名：`src/vuex/store.js`

这个文件的作用，是在整个Vue.js项目中声明：我们要使用Vuex进行状态管理了。

它的内容如下：

```

1. import Vue from 'vue'
2. import Vuex from 'vuex'
3.
4. // 这个就是我们后续会用到的counter 状态.
5. import counter from '@/vuex/modules/counter'
6.
7. Vue.use(Vuex)
8.
9. const debug = process.env.NODE_ENV !== 'production'
10. export default new Vuex.Store({
11.   modules: {
12.     counter // 所有要管理的module，都要列在这里.
13.   },
14.   strict: debug,
15.   middlewares: []
16. })

```

在上面代码中，大部分是鸡肋代码。有用的代码只有：

```

1. import counter from '@/vuex/modules/counter'
2. ...
3.   modules: {
4.     counter
5.   }
6. ...

```

这里定义了所有的 vuex module。

3. 新建vuex/module文件

文件名：`src/vuex/modules/counter.js`

内容如下：

```
1. import { INCREASE } from '@/vuex/mutation_types'
```

```

2.
3. const state = {
4.   points: 10
5. }
6.
7. const getters = {
8.   get_points: state => {
9.     return state.points
10.  }
11. }
12.
13. const mutations = {
14.   [INCREASE](state, data){
15.     state.points = data
16.   }
17.
18. }
19.
20. export default {
21.   state,
22.   mutations,
23.   getters
24. }

```

上面是一个最典型的 vuex module，它的作用就是计数。

- state: 表示状态。可以认为state是一个数据库，保存了各种数据。我们无法直接访问里面的数据。
- mutations: 变化。可以认为所有的state都是由mutation来驱动变化的。也可以认为它是个setter。
- getter: 取值的方法。就是getter(跟setter相对)

我们如果希望“拿到”某个数据，就需要调用 vuex module的 `getter` 方法。我们如果希望“更改”某个数据，就需要调用 vuex module的 `mutation` 方法。

4. 新增文件：`src/vuex/mutation_types.js`

```
1. export const INCREASE = 'INCREASE'
```

大家做项目的时候，要统一把 mutation type 定义在这里。它类似于方法列表。

这个步骤不能省略。Vue.js官方也建议这样写。好处是维护的同学可以看到 某个mutation有多少

种状态。

5. 新增路由： `src/routers/index.js`

```

1. import ShowCounter1 from '@/components>ShowCounter1'
2. import ShowCounter2 from '@/components>ShowCounter2'
3.
4. export default new Router({
5.   routes: [
6.     {
7.       path: '/show_counter_1',
8.       name: 'ShowCounter1',
9.       component: ShowCounter1
10.    },
11.    {
12.      path: '/show_counter_2',
13.      name: 'ShowCounter2',
14.      component: ShowCounter2
15.    }
16.  ]
17. })

```

6. 新增两个页面： `src/components>ShowCounter1.vue` 和
`src/components>ShowCounter2.vue`

这两个页面基本一样。

```

1. <template>
2.   <div>
3.     <h1> 这个页面是 1号页面 </h1>
4.     {{points}} <br/>
5.     <input type='button' @click='increase' value='点击增加1' /><br/>
6.     <router-link :to="{name: 'ShowCounter2'}">
7.       计数页面2
8.     </router-link>
9.   </div>
10.  </template>
11.
12.  <script>
13.  import store from '@/vuex/store'
14.  import { INCREASE } from '@/vuex/mutation_types'
15.  export default {

```

```

16.     computed: {
17.       points() {
18.         return store.getters.get_points
19.       }
20.     },
21.     store,
22.     methods: {
23.       increase() {
24.         store.commit(INCREASE, store.getters.get_points + 1)
25.       }
26.     }
27.   }
28. </script>

```

可以看出， 我们可以在 `<script>` 中调用 vuex的module的方法。 例如：

```

1. increase() {
2.   store.commit(INCREASE, store.getters.get_points + 1)
3. }

```

这里， `store.getters.get_points` 就是通过 `getter` 获取到 状态“points”的方法。

`store.commit(INCREASE, ...)` 则是 通过 `INCREASE` 这个 `action` 来改变 “points”的值。

Computed属性

Computed 代表的是某个组件(component)的属性， 这个属性是算出来的。 每当计算因子发生变化时，这个结果也要及时的重新计算。

上面的代码中：

```

1. <script>
2. export default {
3.   computed: {
4.     points() {
5.       return store.getters.get_points
6.     }
7.   },
8. </script>

```

就是定义了一个叫做 ‘points’ 的 ‘computed’属性。 然后，我们在页面中显示这个“计算属性”：

```
1. <template>
2.   <div>
3.     {{points}}
4.   </div>
5. </template>
```

就可以把 state中的数据显示出来， 并且会自动的更新了。

重启服务器(`$ npm run dev`)，之后运行，可以看到如下图所示：



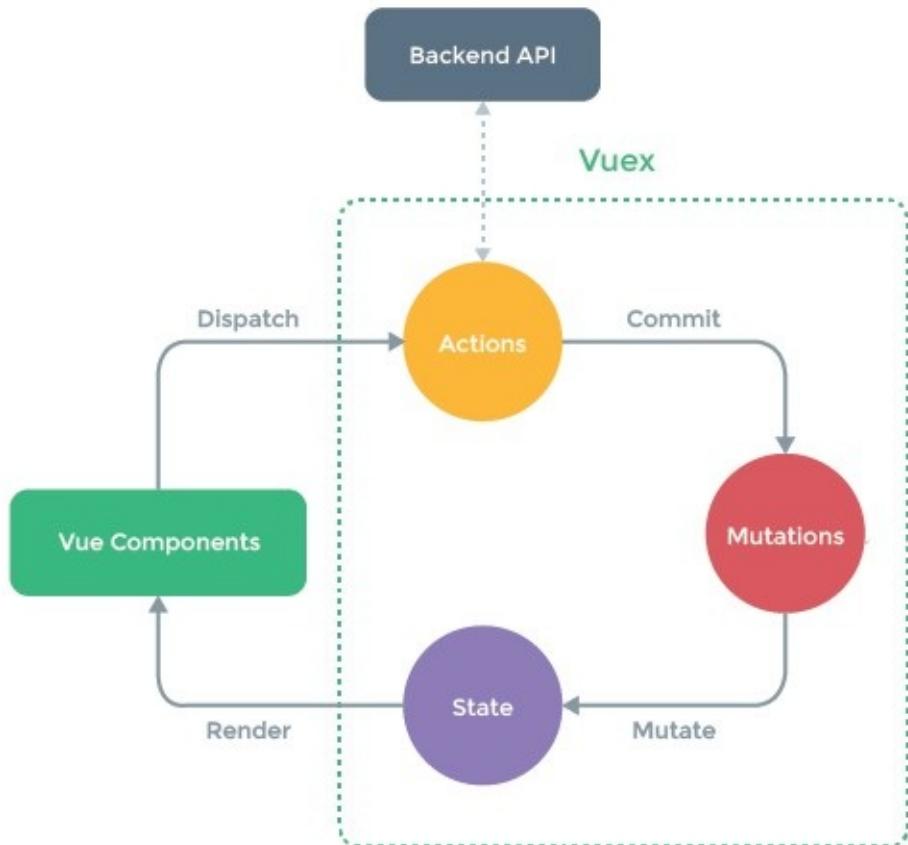
这个页面是 1号页面



Vuex 原理图

为了学会本章内容，大家务必亲手敲一敲代码。

下面是Vuex的原理图 .

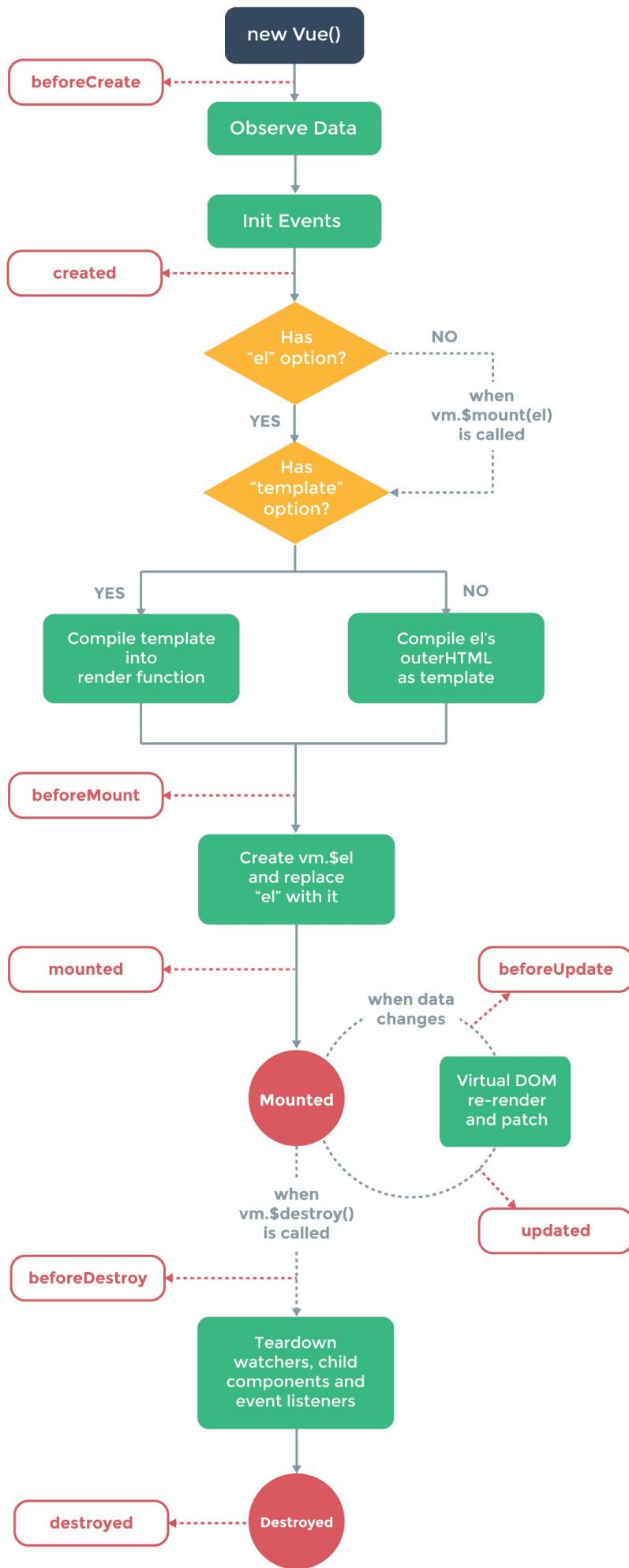


可以看到：

1. 总体分成：Action, Mutation, State 三个概念. State由Mutation来变化
2. Vuex 通过 Action 与后端API进行交互
3. Vuex 通过 State 来渲染前端页面。
4. 前端页面通过触发 Vuex的 Action, 来提交mutation, 达到改变 “state”的目的。

Vuejs的生命周期

每个 Vuejs 的实例，都会经历下图的生命周期。



可以看出，基本周期是：

1. created (创建好DOM)
2. mounted (页面基本准备好了。)
3. updated (update 可以理解成入肉手动操作触发)
4. destroyed (销毁)

上面步骤中的 1, 3, 4都是自动触发。 每个步骤都有对应的 beforeXyz方法

所以， 我们一般使用 `mounted` 作为页面初始化时执行的方法

最佳实践

适当的使用vuex

能不用就不用。 能用就用。

不要为了使用而使用，例如一个小方法就可以搞定的事情，搞出五个设计模式来实现。

不要过度使用CSS框架

因为CSS框架一般会大幅度增加文件体积。 例如 bootstrap, ele.me前端框架。 这个在低端安卓机上影响显著。 特别是使用Android中的 Webview 来加载H5页面时，基本上1k大小的CSS就会消耗1ms。

而CSS框架动辄几百K，每次手机端都要等好久，才会打开对应的页面。

使用CDN来存放图片文件

例如， upyun 就是个不错的选择。 阿里的oss也很好。

js, css 尽量使用压缩

我们在nginx中可以设置这一项。 让js, css 都以zip的形式来发送和接收，一般都会有效减少30%-60%的体积和传送时间。 具体请参考nginx文档。

灵活使用第三方Vue 插件

例如： 轮播图， 表单验证等等。这些轮子都是现成的。

好的程序员不一定算法好，但一定是一个对各种第三方组件见多识广的人。

前端逻辑务必简单

能在后台处理的，绝对不要放在前端处理。 因为Vue.js 擅长的不是处理数据结构。

例如，前端需要展示一个列表的话，后端的接口就应该给出JSON中的数组， 而不是给出一个字符串，然后由前端去解析。

不用写行末分号。

Vue.js 源代码中没有一行有“行末分号”。会有es 预处理器帮我们填上的。

灵活使用CSS，HTML预处理工具

我们知道，

- JADE, HAML可以生成HTML,
- SASS, SCSS, LESS可以生成CSS

如果是公司的人数比较多，有UI，有前端，那么建议大家使用的话要慎重。建议直接使用原生的HTML, CSS。因为UI设计师同学不一定看得懂SCSS, JADE

如果是一个人独立负责整个项目，那么用JADE, SCSS也没问题。

Event Handler 事件处理

Vuejs中的事件处理非常强大， 也非常重要。 我们一定要学好它。

Event Handler 之所以会被Vuejs放到很高的地位， 是基于这样的考虑：

1. 把跟事件相关的代码独立的写出来， 非常容易定位各种逻辑， 维护起来方便。
2. event handler 被独立出来之后， 页面的DOM元素看起来就会很简单。 容易理解。
3. 当一个页面被关掉时， 对应的ViewModel也会被回收。 那么该页面定义的各种 event handler 也会被一并垃圾回收。 不会造成内存溢出。

支持的Event

我们在前面曾经看到过 `v-on:click`， 那么， 都有哪些事件可以被 `v-on` 所支持呢？

只要是标准的HTML定义的Event， 都是被Vuejs支持的。

- `focus` (元素获得焦点)
- `blur` (元素失去焦点)
- `click` (单击 鼠标左键)
- `dblclick` (双击鼠标左键)
- `contextmenu` (单机鼠标右键)
- `mouseover` (指针移到有事件监听的元素或者它的子元素内)
- `mouseout` (指针移出元素， 或者移到它的子元素上)
- `keydown` (键盘动作： 按下任意键)
- `keyup` (键盘动作： 释放任意键)

可以来这里查看 [所有HTML标准事件](#)

总共定义了 162个标准事件， 和 几十个非标准事件， 以及 Mozilla的特定事件。 如下图所示：

标准事件

这些事件在官方Web规范中定义，并且应在各个浏览器中通用。每个事件都和代表事件接收方的对象（由此您可以查到每个事件提供的数据），定义这个事件的标准或标准链接会一起列出。

| 事件名称 | 事件类型 | 规范 | 触发时机... |
|---------------------------------|------------------------------------|-----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| <code>abort</code> | <code>UIEvent</code> | <input checked="" type="checkbox"/> DOM L3 | 资源载入已被中止 |
| <code>abort</code> | <code>ProgressEvent</code> | <input checked="" type="checkbox"/> Progress and <input checked="" type="checkbox"/> XMLHttpRequest | Progress 被终止(不是error造成的) |
| <code>abort</code> | <code>Event</code> | <input checked="" type="checkbox"/> IndexedDB | 事务已被中止 |
| <code>afterprint</code> | <code>Event</code> | <input checked="" type="checkbox"/> HTML5 | 相关文档已开始打印或打印预览已被关闭 |
| <code>animationend</code> | <code>AnimationEvent</code> | <input checked="" type="checkbox"/> CSS Animations | 完成一个CSS 动画 |
| <code>animationiteration</code> | <code>AnimationEvent</code> | <input checked="" type="checkbox"/> CSS Animations | 重复播放一个CSS 动画 |
| <code>animationstart</code> | <code>AnimationEvent</code> | <input checked="" type="checkbox"/> CSS Animations | 一个CSS 动画已开始 |
| <code>audioprocess</code> | <code>AudioProcessingEvent</code> | <input checked="" type="checkbox"/> Web Audio API
audioprocess | 一个 <code>ScriptProcessorNode</code> 的输入缓冲区可处理 |
| <code>audioend</code> | <code>Event</code> | <input checked="" type="checkbox"/> Web Speech API | 用户代理捕捉到用以语音识别的音频 |
| <code>audiostart</code> | <code>Event</code> | <input checked="" type="checkbox"/> Web Speech API | 用户代理开始捕捉用以语音识别的音频 |
| <code>beforeprint</code> | <code>Event</code> | <input checked="" type="checkbox"/> HTML5 | 相关文档将要开始打印或准备打印预览 |
| <code>beforeunload</code> | <code>BeforeUnloadEvent</code> | <input checked="" type="checkbox"/> HTML5 | 即将卸载 window, document 及其资源 |
| <code>beginEvent</code> | <code>TimeEvent</code> | <input checked="" type="checkbox"/> SVG | A SMIL animation element begins. |
| <code>blocked_indexedDB</code> | <code>IDBVersionChangeEvent</code> | <input checked="" type="checkbox"/> IndexedDB | An open connection to a database is blocking a <code>versionchange</code> transaction on the same database. |

我们不用全部都记住，通常在日常开发中，只有不到20个是最常见的event.

使用 v-on 进行事件的绑定

我们可以认为，几乎所有的事件，都是由 `v-on` 这个 directive 来驱动的。所以，本节会对 `v-on` 有更加详尽的说明。

1. 在 v-on 中使用变量

如下面代码所示，可以在 `v-on` 中引用变量：

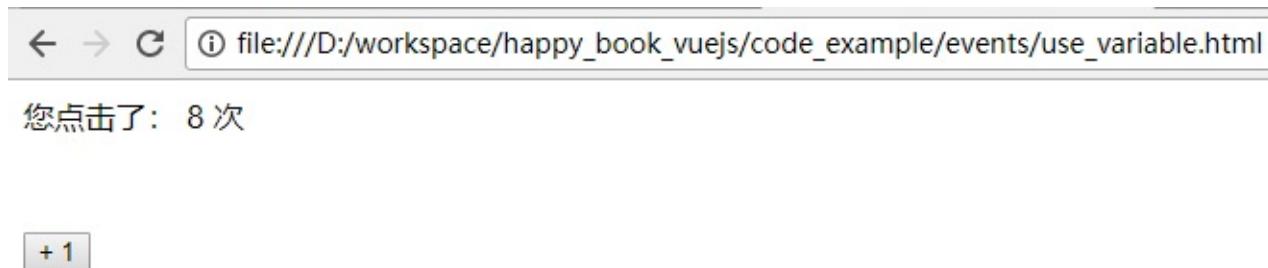
```
1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
```

```

6.   <div id='app'>
7.     您点击了： {{ count }} 次
8.     <br/>
9.     <button v-on:click='count += 1' style='margin-top: 50px'> + 1</button>
10.    </div>
11.
12.    <script>
13.      var app = new Vue({
14.        el: '#app',
15.        data: {
16.          count: 0
17.        }
18.      })
19.    </script>
20.  </body>
21. </html>

```

上面的代码，用浏览器打开后， 点击 按钮， 就可以看到 `count` 这个变量会随之 +1。如下图所示：



2. 在 `v-on` 中使用方法名

上面的例子，也可以按照下面的写法来实现：

```

1.  <html>
2.  <head>
3.    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4.  </head>
5.  <body>
6.    <div id='app'>
7.      您点击了：{{ count }} 次
8.      <br/>

```

```

    <button v-on:click='increase_count' style='margin-top: 50px'> + 1
9. </button>
10. </div>
11.
12. <script>
13.     var app = new Vue({
14.         el: '#app',
15.         data: {
16.             count: 0
17.         },
18.         methods: {
19.             increase_count: function(){
20.                 this.count += 1
21.             }
22.         }
23.     })
24. </script>
25. </body>
26. </html>

```

可以看到，在 `v-on:click='increase_count'` 中，`increase_count` 就是一个方法名。

3. 在v-on 中使用方法名 + 参数

我们也可以直接使用 `v-on:click='some_function("your_parameter")'` 这样的写法，如 下面的例子所示：

```

1. <html>
2. <head>
3.     <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.     <div id='app'>
7.         {{ message }}
8.         <br/>
9.         <button v-on:click='say_hi("明日的Vuejs大神")' style='margin-top: 50px'>
9.             跟我打个招呼~ </button>
10.        </div>
11.
12.        <script>
13.            var app = new Vue({
14.                el: '#app',

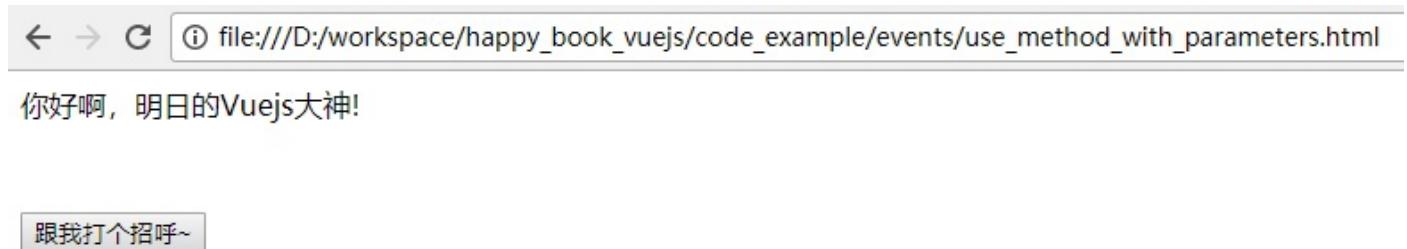
```

```

15.     data: {
16.         message: "这是个 在click中调用 方法 + 参数的例子"
17.     },
18.     methods: {
19.         say_hi: function(name){
20.             this.message = "你好啊, " + name + "!"
21.         }
22.     }
23. )
24. </script>
25. </body>
26. </html>

```

使用浏览器打开后，点击按钮，就可以看到下图所示：



4. 重新设计按钮的逻辑

我们在实际开发中，往往遇到这样的情况： 点击某个按钮，或者触发某个事件后，希望概念 按钮的默认状态。

最典型的例子： 提交表单(

)的时候，我们希望先对该表单进行验证。 如果验证不通过，该表单就不要提交。

这个时候，如果希望表单不要提交，我们就要让 这个 submit 按钮，不要有下一步的动作。 在所有的开发语言当中，都会有一个对应的方法，叫做：“`preventDefault`”(停止默认动作)

我们看这个例子：

```

1. <html>
2. <head>
3.     <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>

```

```

4.  </head>
5.  <body>
6.      <div id='app'>
7.
8.          请输入您想打开的网址,      <br/>
9.          判断规则是 :           <br/>
10.         1. 务必以 "http://"开头    <br/>
11.         2. 不能是空字符串        <br/>
12.         <input v-model="url" placeholder="请输入 http:// 开头的字符串, 否则不会跳转"
13.     <br/>
14.     <br/>
15.     <a v-bind:href="this.url" v-on:click='validate($event)'> 点我确定 </a>
16.   </div>
17.
18.   <script>
19.       var app = new Vue({
20.           el: '#app',
21.           data: {
22.               url: ''
23.           },
24.           methods: {
25.               validate: function(event){
26.                   if(this.url.length == 0 || this.url.indexOf('http://') != 0){
27.                       alert("您输入的网址不符合规则。 无法跳转")
28.                       if(event){
29.                           alert("event is: " + event)
30.                           event.preventDefault()
31.                       }
32.                   }
33.               }
34.           })
35.       </script>
36.   </body>
37. </html>

```

上面的代码中，可以看到，我们定义了一个变量：`url`。并且通过代码：

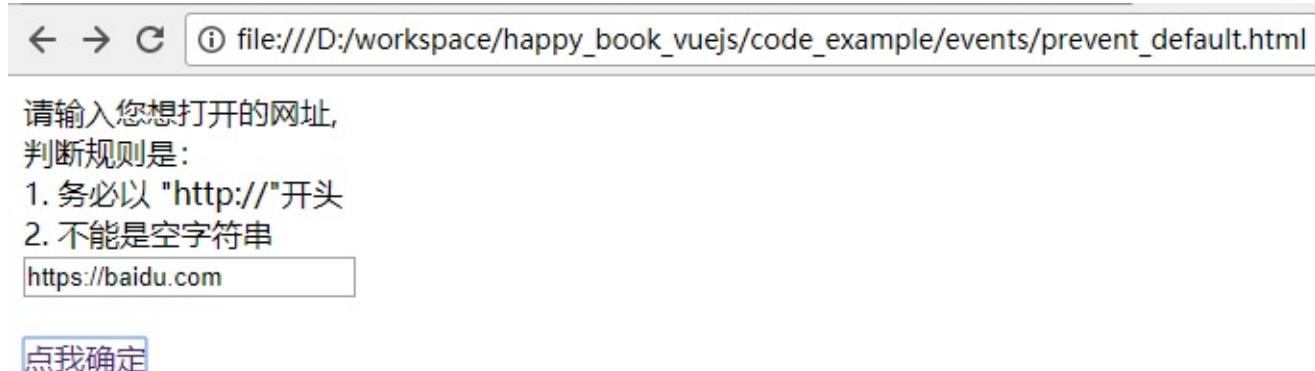
`<a v-bind:href="this.url" v-on:click='validate($event)'> 点我确定 ` 做了两件事情：

1. 把 `url` 绑定到了该元素上。
2. 该元素 在触发 `click` 事件时，会调用 `validate` 方法。 该方法传递了一个特殊的参数：

`$event` . 该参数是当前 事件的一个实例。 (MouseEvent)

在 `validate` 方法中，我们是这样定义的： 先验证是否符合规则。 如果符合，放行，会继续触发 `<a/>` 元素的默认动作（让浏览器发生跳转） 。 否则的话，会弹出一个 “alert” 提示框。

用浏览器打开这段代码，可以看到下图所示：



我们先输入一个合法的地址：`http://baidu.com`，可以看到，点击后，页面发生了跳转。 跳转到了百度。

我们再输入一个“不合法”的地址：`https://baidu.com` 注意： 该地址不是以 “http://” 开头，所以我们的vuejs 代码不会让它放行。

如下图所示：



进一步观察，页面也不会跳转（很好的解释了 这个时候 `<a/>` 标签点了也不起作用）

5. Event Modifiers 事件修饰语

我们很多时候，希望把代码写的优雅一些。 使用传统的方式，可能会把代码写的很臃肿。 如果某个元素在不同的event下有不同的表现，那么代码看起来就会有很多个 `if ... else ...` 这样的分支。

所以， Vue.js 提供了 “Event Modifiers”。

例如，我们可以把上面的例子略加修改：

```
1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.
8.     请输入您想打开的网址,          <br/>
9.     判断规则是：                  <br/>
10.    1. 务必以 "http://"开头      <br/>
11.    2. 不能是空字符串          <br/>
12.    <input v-model="url" placeholder="请输入 http:// 开头的字符串，否则不会跳转"
13.    /> <br/>
14.    <br/>
15.    <a v-bind:href="this.url" v-on:click='validate($event)' v-
16.      on:click.prevent='show_message'> 点我确定 </a>
17.  </div>
18.
19.  <script>
20.    var app = new Vue({
21.      el: '#app',
22.      data: {
23.        url: ''
24.      },
25.      methods: {
26.        validate: function(event){
27.          if(this.url.length == 0 || this.url.indexOf('http://') !=
28.            0){
29.            if(event){
30.              event.preventDefault()
31.            }
32.            show_message: function(){
33.              alert("您输入的网址不符合规则。 无法跳转")
34.            }
35.          })
36.        }
37.      }
38.    }
39.  )
40. </script>
```

```
37. </body>
38. </html>
```

可以看出，上面的代码的核心是：

```
<a v-bind:href="this.url" v-on:click='validate($event)' v-
1. on:click.prevent='show_message'> 点我确定 </a>
2.
3. methods: {
4.     validate: function(event){
5.         if(this.url.length == 0 || this.url.indexOf('http://') != 0){
6.             if(event){
7.                 event.preventDefault()
8.             }
9.         }
10.    },
11.    show_message: function(){
12.        alert("您输入的网址不符合规则。 无法跳转")
13.    }
14. }
```

先是在 `<a/>` 中定义了两个 `click` 事件，一个是 `click`，一个是 `click.prevent`。后者表示，如果该元素的 `click` 事件被 阻止了的话， 应该触发什么动作。

然后，在 `methods` 代码段中，专门定义了 `show_message`，用来给 `click.prevent` 所使用。

上面的代码运行起来，跟前一个例子是一模一样的。 只是抽象分类的程度更高了一些。 在复杂的项目中有用处。

这样的“event modifier”，有这些：

- `stop propagation` 被停止后（也就是调用了 `event.stopPropagation()`方法后），被触发
- `prevent` 调用了 `event.preventDefault()` 后被触发。
- `capture` 子元素中的事件可以在该元素中 被触发。
- `self` 事件的 `event.target` 就是本元素时，被触发。
- `once` 该事件最多被触发一次。
- `passive` 为移动设备使用。（在`addEventListener` 定义时，增加`passive`选项。）

以上的“event modifier”也可以连接起来使用。 例如： `v-on:click.prevent.self`

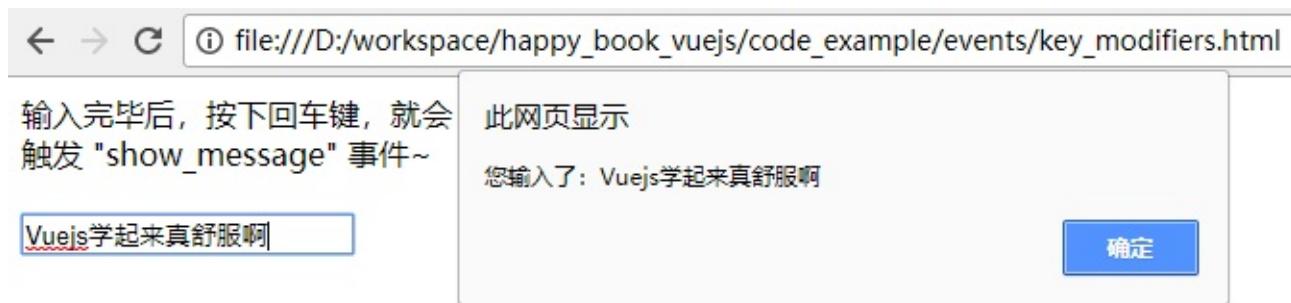
6. Key Modifiers 按键修饰语

Vue.js 也很贴心的提供了 Key Modifiers，也就是一种支持键盘事件的快捷方法。 我们看下面的例子：

```
1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.     输入完毕后，按下回车键，就会<br/>
8.     触发 "show_message" 事件~ <br/><br/>
9.
10.    <input v-on:keyup.enter="show_message" v-model="message" />
11.  </div>
12.
13.  <script>
14.    var app = new Vue({
15.      el: '#app',
16.      data: {
17.        message: ''
18.      },
19.      methods: {
20.        show_message: function(){
21.          alert("您输入了：" + this.message)
22.        }
23.      }
24.    })
25.  </script>
26. </body>
27. </html>
```

可以看到，在上面的代码中，`v-on:keyup.enter="show_message"` 为 `<a/>` 元素定义了事件，该事件对应了“回车键”。（严格的说，是回车键被按下后，松开弹起来的那一刻）

我们用浏览器打开上面的代码对应的文件，输入一段文字，按回车，就可以看到事件已经被触发了。如下图所示：



Vuejs 总共支持下面这些 Key modifiers:

- enter 回车键
- tab tab 键
- delete 同时对应了 backspace 和 del 键
- esc ESC 键
- space 空格
- up 向上键
- down 向下键
- left 向左键
- right 向右键

随着 Vuejs 版本的不断迭代和更新，越来越多的 Key modifiers 被添加了进来，例如 `page` `down`，`ctrl`。对于这些键的用法，大家可以查阅官方文档。

与CSS预处理器结合使用

20年前的《程序员修炼之道》这本书，就提到了程序员的一个职业习惯： DIY。 (Don't Repeat Yourself)， 不要做重复的事儿。

目前的编程语言，几乎都具备了消灭重复代码的能力。

除了CSS

CSS 是唯一不具备 支持变量的 编程语言。 因为CSS 本身只是一个DSL (Domain Specific Language领域特定的语言)， 它不是一个“编程语言”。

这样就决定了它的特点： 上手极快， 可以很好的表现HTML中某个元素的外观。 缺点就是： 无法通过常见的重构手法 (Extract Method, Extract Variable..) 来精简代码。

所以， SCSS, SASS, LESS 等一系列的 “CSS 预处理器” (precompiler) 应运而生。

SCSS

全名叫 Sassy CSS， (时髦的CSS) 是 SASS 3 引入新的语法，其语法完全兼容 CSS3，并且继承了 SASS 的强大功能。

也就是说，任何标准的 CSS3 样式表都是具有相同语义的有效的 SCSS 文件。

官方网站同 SASS。

由于 SCSS 是 CSS 的扩展，因此，所有在 CSS 中正常工作的代码也能在 SCSS 中正常工作。 也就是说，对于一个 SASS 用户，只需要理解 SASS 扩展部分如何工作的，就能完全理解 SCSS。

大部分的用法都跟SASS相同。 唯一不同的是，SCSS 需要使用分号和花括号而不是换行和缩进。

SCSS可以说是全面取代了 SASS。

我们看下面的例子：

```
1. $font-stack: Helvetica, sans-serif;
2. $primary-color: #333;
3.
4. body {
5.   font: 100% $font-stack;
6.   color: $primary-color;
7. }
```

在上面的代码中， 定义了两个变量： `$font-stack` 和 `$primary-color` . 编译后的CSS如下：

```
1. body {
2.   font: 100% Helvetica, sans-serif;
3.   color: #333;
4. }
```

更多内容，可以到官方网站来学习： <https://sass-lang.com/guide> 非常简单，有一点儿CSS功底的人，往往可以瞬间上手。

LESS

LESS 也是一种 CSS 预处理器，它的自我介绍是： 只是多了“一丢丢”内容的CSS. (It's CSS, with just a little more).

官方网址： <http://lesscss.org/> , Github: <https://github.com/less/less.js> , 截止到 2018-6-25 , github关注数是 15591

它的作用跟SCSS一样，也是为了让代码更加精简，去掉无意义的重复。 我们看下面的例子：

```
1. // Variables
2. @link-color:          #428bca; // sea blue
3. @link-color-hover:   darken(@link-color, 10%);
4.
5. // Usage
6. a,
7. .link {
8.   color: @link-color;
9. }
10. a:hover {
11.   color: @link-color-hover;
12. }
13. .widget {
14.   color: #fff;
15.   background: @link-color;
16. }
```

可以看到，上面的例子定义了 两个变量： `@link-color` 和 `@link-color-hover` . 并且在下方进行了引用。 同时， 还有使用了换算功能 `darken(@link-color, 10%)` 。

上面的代码会被编译成下面的CSS：

```

1. a,
2. .link {
3.   color: #428bca;
4. }
5. a:hover {
6.   color: #3071a9;
7. }
8. .widget {
9.   color: #fff;
10.  background: #428bca;
11. }

```

可以看到， less的功能非常强大。

SASS

提到了SCSS, LESS, 就不得不提及SASS

官方网站: <https://sass-lang.com/> , github: <https://github.com/sass/sass> , 截止到 2016-6-25, github关注数是 11376

特点是去掉了 花括号，分号，看起来特别简单。 使用空格来标记不同的段落层次。 跟HAML基本是一样的。

我们看下面的例子：

```

1. $font-stack:    Helvetica, sans-serif
2. $primary-color: #333
3.
4. body
5.   font: 100% $font-stack
6.   color: $primary-color

```

在上面的代码中， 定义了两个变量： `$font-stack` 和 `$primary-color` 。 并且在下面对它们进行了引用。

我们看编译后的结果：

```

1. body {
2.   font: 100% Helvetica, sans-serif;
3.   color: #333;
4. }

```

不过在实际应用当中，却很少使用这个语言。 因为实际应用中，程序员最喜欢的事情，就是把“UI”或者“美工”， 或者“前端工程师”给过来的CSS文件直接使用。 如果使用SASS的话就很尴尬，还要再动作做一遍转换。 浪费时间。 而且“美工”同学可以看得懂CSS，却无法看懂SASS。

所以这个技术比较落没。 慢慢的被SCSS（SASS 3.0）所取代。

所以，同学在这个技术的故事上要学习到一个道理： 程序员认为好的技术，在团队配合的过程和实践中是不一定适用的。 SASS 团队也做的非常不做，立刻就在SASS 3.0的时候推出了SCSS。

在 Vue.js 中使用CSS预编译器

使用的前提，是我们以 Webpack的形式使用Vue.js。

使用方法非常简单。 我们以SASS为例：

1. 安装依赖：“sass-loader” 和 “node-sass”，运行下面命令：

```
1. $ npm i sass-loader node-sass -D
```

2. 在“webpack.base.conf.js”中添加相关配置：

```
1. {
2.   test: /\.s[a|c]ss$/,
3.   loader: 'style!css!sass'
4. }
```

3. 在对应的 “.vue” 文件中，我们就可以这样的定义某个样式：

```
1. <style lang='sass'>
2. td {
3.   border-bottom: 1px solid grey;
4. }
5. </style>
```

上面的代码，会在运行的时候，被webpack编译成对应的CSS文件。

自定义 Directive

Vue.js除了自身提供的 `v-if` , `v-model` 等标准的Directive之外，还提供了非常强大的自定义功能。使用这个功能，我们就可以定义属于我们自己的Directive.

例子

我们看下面例子：

```

1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
    下面是使用了自定义的Directive的input，可以自动聚焦（调用 focus() 方法）：
7.   <br/>
8.     <br/>
9.     <input v-myinput/>
10.  </div>
11.  <script>
12.    var app = new Vue({
13.      el: '#app',
14.      directives: {
15.        "myinput": {
16.          inserted: function(element){
17.            element.focus()
18.          }
19.        }
20.      }
21.    })
22.  </script>
23. </body>
24. </html>

```

上面的代码中，先是在 Vue 中，定义了一个 `directives` 代码段：

```

1. directives: {
2.   myinput: {
3.     inserted: function(element){

```

```

4.           element.focus()
5.       }
6.   }
7. }
```

- `myinput` 就是自定义Directive的名字。 使用的时候就是 `v-myinput` .
- `inserted` : 这是一个定义好的方法（钩子方法），表示在页面被Vue.js渲染的过程中，在该DOM被“`insert`”（插入）到页面中的时候，被触发。触发的内容是 `element.focus()`

用浏览器打开后，可以看到 `<input/>` 标签是会自动聚焦的。这个时候用户就可以直接输入内容了。如下图所示：



← → C file:///D/workspace/happy_book_vuejs/code_example/custom_directive/index.html

下面是使用了自定义的Directive的input，可以自动聚焦（调用 `focus()` 方法）：



如果在“webpack”等可以修改“全局Vue实例”的时候，也可以使用这样的方法：

```

1. Vue.directive('myinput', {
2.   inserted: function (element) {
3.     element.focus()
4.   }
5. })
```

自定义Directive的命名方法

如果您希望把 `v-myinput` 的调用，写成 `v-my-input`，那么在定义的时候，就应该：

```

1. directives: {
2.   // 注意下面的写法。使用双引号括起来
3.   "my-input": {
4.     inserted: function(element){
5.       element.focus()
6.     }
7.   }
8. }
```

那么我们就可以在View中这样使用了：

```
1. <input v-my-input />
```

钩子方法 (Hook Functions)

我们在上面的例子中，知道了 `inserted` 是一个钩子方法。下面是一个完整的列表：

- `bind` 只运行一次。当该元素首次被渲染的时候。（绑定到页面的时候）
- `inserted` 该元素被插入到父节点的时候（也可以认为是该元素被Vue渲染的时候）
- `update` 该元素被更新的时候。
- `componentUpdated` 包含的component被更新的时候。
- `unbind` 只会运行一次。当该元素被Vue从页面解除绑定的时候。

自定义Directive可以接收到的参数

Vue.js 为自定义Directive 实现了强大的功能，可以接收很多个参数。下面是个例子：

```
1. <html>
2. <head>
3.   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
4. </head>
5. <body>
6.   <div id='app'>
7.     下面是一个非常全面的自定义Directive的例子：<br/>
8.     <br/>
9.     <input v-my-input:foo.click="say_hi" />
10.    </div>
11.    <script>
12.      var a
13.      var app = new Vue({
14.        el: '#app',
15.        data: {
16.          say_hi: '你好啊，我是个value'
17.        },
18.        directives: {
19.          "my-input": {
20.            inserted: function(element, binding, vnode) {
21.              element.focus()
22.              console.info("binding.name: " + binding.name)
23.              console.info("binding.value: " + binding.value)

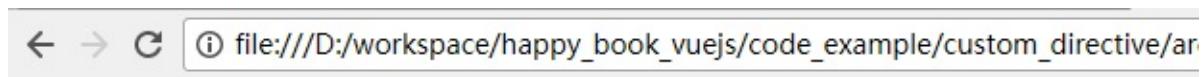
```

```

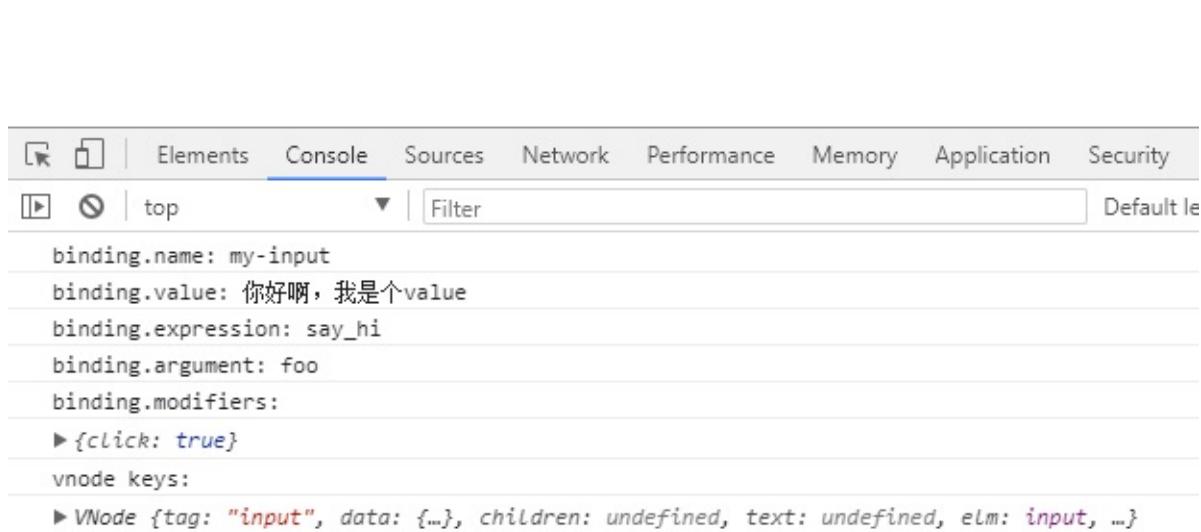
        console.info("binding.expression: " +
24. binding.expression)
25.                 console.info("binding.argument: " + binding.arg)
26.                 console.info("binding.modifiers: ")
27.                 console.info(binding.modifiers)
28.                 console.info("vnode keys:")
29.                 console.info(vnode)
30.             }
31.         }
32.     }
33. )
34. </script>
35. </body>
36. </html>

```

先看运行结果：



下面是一个非常全面的自定义Directive的例子：



在上图中，可以看到，自定义Directive 在声明的时候，接收了3个参数：`function(element, binding, vnode)`

通过这三个参数，就可以看到很多对应的内容，包括 `binding.name`，`binding.value`，`binding.expression`。他们的含义都是字面的意思。

借助这些内容，我们可以实现我们自己想要的Directive.

实战经验

1. 优先考虑使用Component .

考虑到维护成本。 它的作用跟 JSP 中的自定义标签是一样的。 与其使用 Directive，不如使用 Component .

2. 如果一定要用，把它实现的尽量简单。

如果接手的新手水平不如你，那他很可能读不懂这块代码。

实战周边

本章的几个问题，都是曾经耗费我们几天到几周的问题。 我们一并列举出来。前人踩坑，后人绕路。希望我们的经验对同学们有用。

微信支付

微信支付，按照微信的官方文档来看不算难。特别是我们的项目是“传统的WEB项目”的话。

但是，对于SPA（单页应用）来说，就很坑了。几乎没有解释，文档也很烂。

优先使用IOS来调试

微信支付带有一个选项，是可以打印出支付过程中的调试信息的。

但是我们在使用中发现，Android的微信支付错误是不可读的。也就是说，开启debug选项是不可用的。

但是对于苹果的设备支持的就很好。所以大家在开发的时候，要先把苹果设备走通。

微信的支付授权目录问题

在生产环境下，微信要求对于支付路径，在微信的管理后台进行配置，如下图所示：



大家要注意，安卓和IOS的配置是不一样的。

安卓： 取支付页面的URL
IOS： 取根路径URL

例如： 根路径是：<http://yoursite.com>， 支付路径是：
<http://yoursite.com/#/books/pay?id=3>

那么，在设置“支付授权目录”的时候，需要设置两个目录：

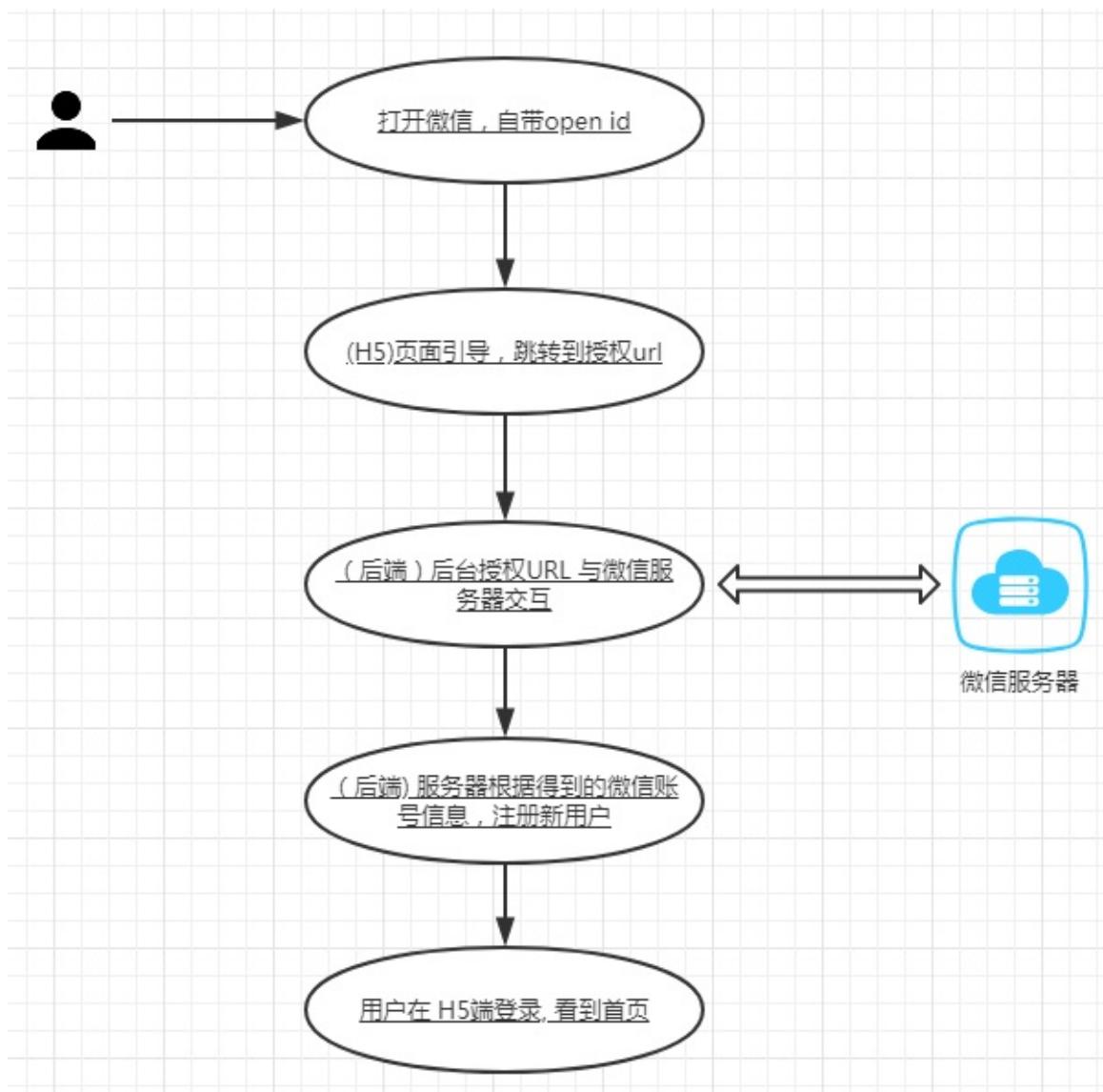
一个是 <http://yoursite.com/#/> (给IOS)一个 是 <http://yoursite.com/#/books/> (给Android)

用户的注册和微信授权

为了追求快速上线，项目组决定，去掉传统项目中的“用户注册”，“用户登录”，直接使用微信的账户来核对。

1. 用户的微信浏览器带着当前微信用户的open_id，跳转到“后台服务器”。
2. “后台服务器” 给“微信服务器” 发送请求。 获得当前微信用户的信息
3. “后台服务器” 为该用户生成一个用户文件
4. “后台服务器” 告知“H5端” 已经成功注册该用户。
5. “H5端” 为该用户展示对应的页面。

如下图所示：



可以看出，主要代码都是在 服务器端。

1. 让微信用户打开首页后，直接跳转到后台服务器

1.1 修改对应的路由文件(src/router/index.js)：

```
1. Vue.use(Router)
2.
3. export default new Router({
4.   routes: [
5.     {
6.       path: '/wait_to_shouquan',
7.       name: 'wait_to_shouquan',
8.       component: require('../views/wait_to_shouquan.vue')
9.     },
10.   ]
11. })
```

1.2 增加对应的vue(src/views/wait_to_shouquan.vue)：

```
1. <template>
2.   <div style="padding: 50px;">
3.     <h3>正在跳转到授权界面...</h3>
4.   </div>
5. </template>
6.
7. <script>
8.   export default {
9.     created () {
10.       window.location.href = this.$store.state.web_share + "/auth/wechat"
11.     },
12.     components: {
13.     }
14.   }
15. </script>
```

可以看到，上面的代码中，使用到了 Vuex 来保存系统变量（后台服务器的地址）。

1.3 增加核心模板文件(src/main.vue)：

```
1. <template>
2.   <div id="app">
3.     <router-view></router-view>
4.   </div>
5. </template>
```

```
6.
7. <script>
8. import store from './vuex/store'
9. import { SET_BASEINFO, GET_BASEINFO } from './vuex/mutation_types'
10. export default {
11.   store,
12.   name: 'app',
13.   data () {
14.     return {
15.       user_info: {
16.         open_id: this.$route.query.open_id
17.       }
18.     }
19.   },
20.   mounted () {
21.
22.     // TODO 开发环境下使用， 生产环境下注释掉
23.     // store.dispatch(SET_BASEINFO, {open_id: 'opFELv6YkJkMaH-xFkokTWCs5AlQ'})
24.
25.     if (this.user_info.open_id) {
26.       store.dispatch(SET_BASEINFO, this.user_info)
27.     } else {
28.       store.dispatch(SET_BASEINFO)
29.       if (store.state.userInfo.open_id === undefined) {
30.         console.info('用户id和open_id不存在， 跳转到授权等待页面')
31.         this.$router.push({name: 'wait_to_shouquan'})
32.       } else {
33.         console.info('已经有了BASEINFO')
34.       }
35.     }
36.   },
37.   watch: {
38.     '$route' (val) {
39.     }
40.   },
41.   methods: {
42.   },
43.   components:{}
44.   }
45. }
46. </script>
47. <!-- 下方的 CSS 略过 -->
```

可以看到，上面代码的 `mounted()` 方法中，会对当前用户的 `open_id` 进行判断。如果存在，调到首页。如果不存在，表示该用户是新用户。需要跳转到授权等待页面。对应的代码如下所示：

```

1.     if (this.userInfo.open_id) {
2.         store.dispatch(SET_BASEINFO, this.userInfo)
3.     } else {
4.         store.dispatch(SET_BASEINFO)
5.         if (store.state.userInfo.open_id === undefined) {
6.             console.info('用户id和open_id不存在，跳转到授权等待页面')
7.             this.$router.push({name: 'wait_to_shouquan'})
8.         } else {
9.             console.info('已经有了BASEINFO')
10.        }
11.    }

```

1.4 增加对应的Vuex代码

目前来看，Vuex需要保存2个信息：

- 用户的 `open id`
- 远程服务器的地址，端口等常量。

1.4.1 增加 `src/vuex/store.js`。这个是最最核心的文件。完整如下所示：

```

1. import Vue from 'vue'
2. import Vuex from 'vuex'
3. import userInfo from './modules/user_info'
4. import tabbar from './modules/tabbar'
5. import toast from './modules/toast'
6. import countdown from './modules/countdown'
7. import products from './modules/products'
8. import shopping_car from './modules/shopping_car'
9.
10. import * as actions from './actions'
11. import * as getters from './getters'
12.
13. Vue.use(Vuex)
14. Vue.config.debug = true
15.
16. const debug = process.env.NODE_ENV !== 'production'
17.
18. export default new Vuex.Store({
19.     state: {

```

```

20.     web_share: 'http://shopweb.siwei.me',
21.     h5_share: 'http://shoph5.siwei.me/?#'
22.   },
23.   actions,
24.   getters,
25.   modules: {
26.     products,
27.     shopping_car,
28.     userInfo,
29.     tabbar,
30.     toast,
31.     countdown
32.   },
33.   strict: debug,
34.   middlewares: debug ? [] : []
35. })

```

上面的代码中，部分代码是在后面会陆续用到的。我们不用过多考虑。只需要关注下面几行：

```

1. export default new Vuex.Store({
2.   // 这里定义了若干系统常量
3.   state: {
4.     web_share: 'http://shopweb.siwei.me',
5.     h5_share: 'http://shoph5.siwei.me/?#'
6.   },
7.
8.   modules: {
9.     // 这里定义了 当前用户的各种信息，我们把它封装成为一个js对象
10.    userInfo,
11.  },
12.
13. })

```

1.4.2 增加 `vuex/modules/user_info.js` 这个文件：

```

1. import {
2.   SET_BASEINFO,
3.   CLEAR_BASEINFO,
4.   GET_BASEINFO,
5.   COMMEN_ROLE,
6.   GET_BGCOLOR,
7.   GET_FONTCOLOR,

```

```
8.     GET_BORDERCOLOR,
9.     GET_ACTIVECOLOR,
10.    EXCHANGE_ROLE
11. } from './mutation_types'
12.
13. const state = {
14.   id: undefined, //用户id
15.   open_id: undefined, // 用户open_id
16.   role: undefined
17. }
18.
19. const mutations = {
20.   //设置用户个人信息
21.   [SET_BASEINFO] (state, data) {
22.     try {
23.       state.id = data.id
24.       state.open_id = data.open_id
25.       state.role = data.role
26.     } catch (err) {
27.       console.log(err)
28.     }
29.   },
30.   //注销用户操作
31.   [CLEAR_BASEINFO] (state) {
32.     console.info('清理缓存')
33.     window.localStorage.clear()
34.   },
35. }
36.
37. const getters = {
38.   [GET_BASEINFO]: state => {
39.     console.info('进入到了getter中了')
40.     let localStorage = window.localStorage
41.     let user_info
42.     if (localStorage.getItem('SLLG_BASEINFO')) {
43.       console.info('有数据')
44.       user_info = JSON.parse(localStorage.getItem('SLLG_BASEINFO'))
45.     } else {
46.       console.info('没有数据')
47.     }
48.     return user_info
49.   },

```

```

50. [COMMEN_ROLE]: state => {
51.   if (state.role === 'yonghu') {
52.     return true
53.   } else {
54.     return false
55.   }
56. },
57. }
58.
59.
60.
61. const actions = {
62.   [SET_BASEINFO] ({ commit, state }, data) {
63.     //保存信息
64.     if (data !== undefined) {
65.       let localStorage = window.localStorage
66.       localStorage.setItem('BASEINFO', JSON.stringify(data))
67.       commit(SET_BASEINFO, data)
68.     } else {
69.       if (localStorage.getItem('BASEINFO')) {
70.         data = JSON.parse(localStorage.getItem('BASEINFO'))
71.         commit(SET_BASEINFO, data)
72.       } else {
73.       }
74.     }
75.   }
76. }
77.
78. export default {
79.   state,
80.   mutations,
81.   actions,
82.   getters
83. }

```

该文件定义了用户的信息的各种属性。

1.4.3 增加 `src/vuex/mutation_types.js`

```

1. export const SET_BASEINFO = 'SET_BASEINFO'
2. export const GET_BASEINFO = 'GET_BASEINFO'

```

上面内容定义了该对象的两个操作。

1.4.4 增加 `src/vuex/modules/actions.js`

```
1. import * as types from './mutation_types'
```

可以看到上面的内容对于 `mutation_types` 进行了引用。

1.4.5 增加 `src/vuex/modules/getters.js`

```
1. // 先放成空内容好了
```

这个文件先这样存在，目前阶段不需要有任何内容。

1.5 与后台的对接

后台的同学，为我们提供了一个链接入口：

<http://shopweb.siwei.me/auth/wechat>

我们让H5页面直接跳转过去就可以。这里不需要加任何参数，直接由后端的同学搞定接下来的事情。

查看效果

在微信开发者工具中，打开我们的H5 页面，就会看到页面会自动跳转。

观察仔细的同学可以看到有两次跳转：

- 第一次跳转到 <http://shopweb.siwei.me/auth/wechat>
- 第二次跳转到 <https://open.weixin.qq.com/connect/oauth2/authorize>

如下图所示：



明创软件

网页由该公众号开发，请确认授权以下信息

- 获得你的公开信息（昵称、头像等）

确认登录

点击“确认登录”按钮，进行授权后，就会进入到H5的首页

总结

本节中，为了做一件事：让用户跳转到微信授权，并注册，我们做了如下的程序层面的内容：

- 使用Vuex 记录系统常量(远程服务器的地址)
- 使用Vuex 记录用户的信息（新增了一个对象：user_info）
- 使用了一个独立的页面（等待微信授权页面）
- 每次打开首页之前，都要判断该用户是否登录。
- （后台任务）让该用户在微信端授权，并且在本地生成一个新的用户，然后把相关数据返回给前端

Vuex 是Vue.js 最复杂最不好理解的地方。同学们不要怕。之所以这么麻烦，可以认为是

javascript的语言特性决定的。 就好像java, C语言中大量用到的设计模式， 在现代编程语言(Ruby , Python, Perl)中就用不到， 可以有更加简单的办法， 例如Mixin)

另外，本节对于后端的同学是个挑战， 不但要在微信端做修改，还需要对微信返回的数据结构很熟悉。 由于本书内容所限，不再赘述。

Hybrid App：混合式App

目前App几乎是每个互联网公司的标配。但是不是每个团队都具备开发原生app的能力。

于是出现了以 phonegap, titanium, xamarin, react native 等一系列的 混合式App.

我们可以大概了解一下：

- phonegap：出现的最早，使用了很多H5的技术来实现原生app的功能，例如拍照等。很有新意。不过完全没有实用价值。响应速度特别慢。卡顿非常明显。
- titanium：出现的比较早。性能不错。跟小程序很类似，使用js,css的类似技术，在不同平台上只要稍加修改代码，就可以跨平台媲美原生App. 缺点是有学习曲线，特别是module很难写。公司被收购了。
- xamarin：使用 .NET 来实现，跟titanium 很接近。也有不小的使用群体。
- react native：使用js 黑科技，直接生成 Android/IOS，原理与titanium, xamarin 一样。但是是目前走的最远的 混合式开发方式。性能也很高。

共同的缺点

无论是Titanium, 还是 Xamarin, React native, 包括 Weex , 都属于 使用 js/.net , 然后把代码改造成 可以被 Android/IOS 的 javascript virtual machine所能接受的情况。

也就是说，对原生的Android/IOS平台做了封装。

这样的好处，为两个不同的编程语言增加了一个统一的编程入口。

缺点也非常明显：做一些普通的事情（展示页面，点击按钮）没问题，但是一旦需要用到第三方应用（定制化的地图，定制化的身份证识别，人脸识别等等），就需要开发人员具备写“native module”的能力。这就要求开发人员 同时精通 Android / IOS （写这种 native module的要求比单纯使用的要求要高得多）而这背离了 这些技术的初衷（初衷是让不太懂 app 编程的人可以快速上手开发）。

原生的壳儿 + Webview的开发方式，只适用于IOS.

还有一种，就是原生的壳儿 + Webview的开发方式。

- 原生的壳儿，指的是：外壳部分完全使用100%的 native app.
- Webview：所有的页面，都是放到了Webview中来展示。

这个情况，经过我们的实践，对于 IOS 设备是可行的。 安卓是不行的。

IOS： 机器硬件性能好， 软件使用Object C 开发，所以用起来效果非常棒，跟native App的体验是一模一样的。 每个页面都可以瞬间打开，而且页面滑动非常流畅。但是在开发层面上，几乎不用考虑页面的适配。 这个解决了很大的问题。

Android： 机器硬件性能稍差于苹果， 软件使用 java 开发，性能比 Object C差。 而且在 Android中， Webview的性能体验比 IOS的差太多。 每个页面打开速度是3-10秒。而且卡顿严重。

我们曾经试着在优化的路子上走了一段时间，发现对于Android + Webview(Vuejs) 的方式，前期开发成本低于原生， 后期维护成本远高于原生，而且一些问题在混合的架构下， 无解。

所以，我们的结论：

开发App， IOS可以使用混合式开发， Android 务必使用原生开发。

发送http请求

TODO：需要加上 http resource，在 main.js。

只要有js的地方，就要有接口。特别是我们这样前后端分离的SPA，几乎每个页面都要发起http请求。从后台接口读取数据，并且显示在前台页面。

这就需要用到http请求了。

1. 调用http请求

vuejs 内置了对发送http请求的支持。只需要在对应页面的script 标签内加上对应的代码就好。例如：

我们新增一个页面，叫“博客列表页”：[src/components/BlogList.vue](#)，它的作用是从我的个人网站(<http://siwei.me>)上，读取文章的标题，并且显示出来。

代码如下：

```
1. <template>
2.   <div>
3.     <table>
4.       <tr v-for="blog in blogs">
5.         <td>{{blog.title }}</td>
6.       </tr>
7.     </table>
8.   </div>
9. </template>
10.
11. <script>
12. export default {
13.   data () {
14.     return {
15.       title: '博客列表页',
16.       blogs: [
17.     ]
18.   }
19. },
20. mounted() {
21.   this.$http.get('api/interface/blogs/all').then((response) => {
22.     console.info(response.body)
```

```

23.     this.blogs = response.body.blogs
24.   }, (response) => {
25.     console.error(response)
26.   });
27. }
28. }
29. </script>
30.
31. <style >
32.
33. td {
34.   border-bottom: 1px solid grey;
35. }
36. </style>

```

上面的代码中， 我们先看 `<script/>` 代码段，

```

1. export default {
2.   data () {
3.     return {
4.       title: '博客列表页',
5.       blogs: [
6.         ]
7.       }
8.     },
9.     mounted() {
10.       this.$http.get('api/interface/blogs/all').then((response) => {
11.         console.info(response.body)
12.         this.blogs = response.body.blogs
13.       }, (response) => {
14.         console.error(response)
15.       });
16.     }
17.   }

```

上面代码中，先是定义了两个变量：`title`，`blogs`，然后定义了一个 `mounted` 方法。该方法表示当页面加载完毕后应该做哪些事情。是一个钩子方法。

```

1. this.$http.get('api/interface/blogs/all').then((response) => {
2.   console.info(response.body)
3.   this.blogs = response.body.blogs
4. }, (response) => {

```

```

5.     console.error(response)
6. });

```

上面代码，是发起http请求的核心代码。 访问的接口地址是 `api/interface/blogs/all`，然后使用 `then` 方法做下一步的事情，`then` 方法接受两个函数作为参数，第一个是成功后干嘛，第二个是失败后干嘛。

成功后的代码如下：

```
1. this.blogs = response.body.blogs
```

然后，在对应的视图部分显示：

```

1. <tr v-for="blog in blogs">
2.   <td>{{blog.title }}</td>
3. </tr>

```

2. 远程接口的格式

在我的服务器上，读取个人博客标题的接口我已经提前做好了，是：

<http://siwei.me/interface/blogs/all>

内容如下；

```

1. {
2.   blogs: [
3.     {
4.       id: 1516,
5.       title: "网络安全资源",
6.       created_at: "2018-06-24T09:36:20+08:00"
7.     },
8.     {
9.       id: 1515,
10.      title: "github - 邀请伙伴后，需要修改权限",
11.      created_at: "2018-06-20T15:03:33+08:00"
12.    },
13.    {
14.      id: 1514,
15.      title: "ruby/rails - 根据浏览器的语言，来自动识别",
16.      created_at: "2018-06-19T08:28:44+08:00"

```

```

17.     },
18.     {
19.         id: 1513,
20.         title: "google cloud - 申请VM的经验",
21.         created_at: "2018-06-09T16:42:08+08:00"
22.     },
23.     {
24.         id: 1512,
25.         title: "验证码 - 使用geetest 或者网易云盾提供的动态二维码",
26.         created_at: "2018-06-07T09:28:14+08:00"
27.     }
28.     // 更多内容...
29. ]
30. }
```

在浏览器中打开后，如下图所示（使用了 jsonview 插件做了json 的代码格式化）：



```

{
  - blogs: [
    - {
      id: 1516,
      title: "网络安全资源",
      created_at: "2018-06-24T09:36:20+08:00"
    },
    - {
      id: 1515,
      title: "github - 邀请伙伴后，需要修改权限",
      created_at: "2018-06-20T15:03:33+08:00"
    },
    - {
      id: 1514,
      title: "ruby/rails - 根据浏览器的语言，来自动识别",
      created_at: "2018-06-19T08:28:44+08:00"
    },
    - {
      id: 1513,
      title: "google cloud - 申请VM的经验",
      created_at: "2018-06-09T16:42:08+08:00"
    },
    - {
      id: 1512,
      title: "验证码 - 使用geetest 或者网易云盾提供的动态二维码",
      created_at: "2018-06-07T09:28:14+08:00"
    }
  ]
}
```

3. 设置 Vuejs 开发服务器的代理

正常来说， javascript在浏览器中是无法发送跨域请求的，所以我们需要在vuejs的 "开发服务器" 上做个转发配置。

修改：`config/index.js` 文件，增加下列内容：

```

1. module.exports = {
2.   dev: {
3.     proxyTable: {
4.       '/api': { // 1. 对于所有以 "/api" 开头的url 做处理.
5.         target: 'http://siwei.me', // 3. 转发到 siwei.me 上.
6.         changeOrigin: true,
7.         pathRewrite: {
8.           '^/api': '' // 2. 把url中的 "/api" 去掉.
9.         }
10.      }
11.    },
12.  }

```

上面的代码做了三件事：

1. 对于所有以 “/api” 开头的url 做处理.
2. 把url中的 “/api” 去掉.
3. 把新的url 请求打到 siwei.me 上.

例如：

- 原请求： <http://localhost:8080/api/interface/blogs/all>
- 新请求： <http://siwei.me/interface/blogs/all>

注意： 以上的代理服务器内容，只能在 “开发模式” 下才能使用。在生产模式下，只能靠服务器的nginx的特性来解决js跨域问题。

修改后的 `config/index.js` 文件的完整内容如下：

```

1. var path = require('path')
2.
3. module.exports = {
4.   build: {
5.     env: require('./prod.env'),
6.     index: path.resolve(__dirname, '../dist/index.html'),
7.     assetsRoot: path.resolve(__dirname, '../dist'),
8.     assetsSubDirectory: 'static',
9.     assetsPublicPath: '/',
10.    productionSourceMap: true,
11.    productionGzip: false,
12.    productionGzipExtensions: ['js', 'css'],

```

```

13.     bundleAnalyzerReport: process.env.npm_config_report
14.   },
15.   dev: {
16.     env: require('./dev.env'),
17.     port: 8080,
18.     autoOpenBrowser: true,
19.     assetsSubDirectory: 'static',
20.     assetsPublicPath: '/',
21.
22.     proxyTable: {
23.       '/api': {
24.         target: 'http://siwei.me',
25.         changeOrigin: true,
26.         pathRewrite: {
27.           '^/api': ''
28.         }
29.       }
30.     },
31.
32.     cssSourceMap: false
33.   }
34. }
```

重启服务器，可以看到我们的转发设置已经生效：

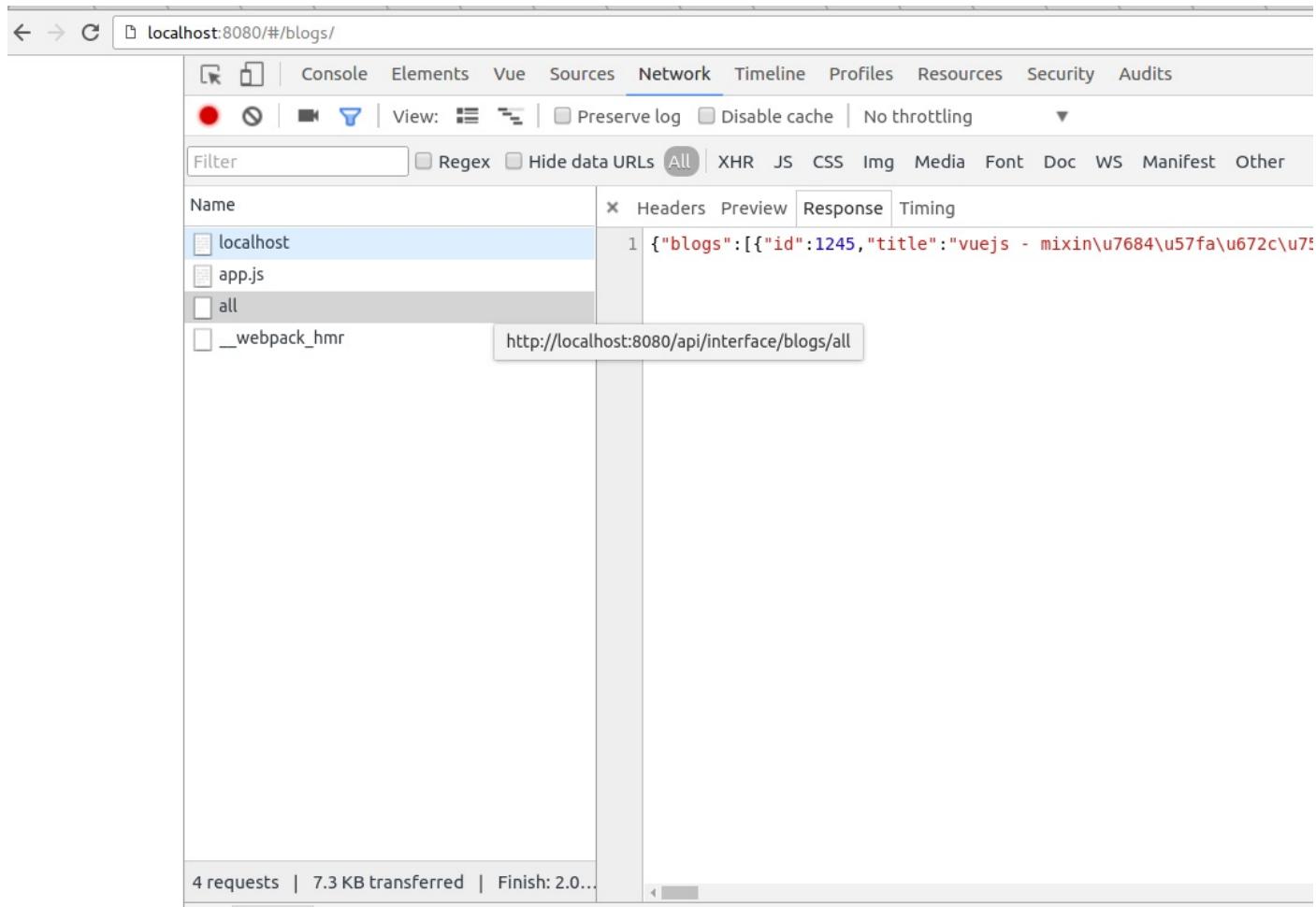
```

1. $ npm run dev
2. ...
3. [HPM] Proxy created: /api  ->  http://siwei.me
4. [HPM] Proxy rewrite rule created: "^/api" ~> ""
5. > Starting dev server...
6. ...
```

4. 打开页面，查看http请求

我们接下来，访问 <http://localhost:8080/#/blogs/>

打开chrome developer tools，就可以看到，“Network”中，已经有请求发出去了，截图显示了结果：



另外，我们也可以直接在浏览器中，输入要打开的链接，看到结果。（该浏览器使用了 json view插件）



5. 把结果渲染到页面中。

我们发现，在export代码段中，有两个部分：

1. `<script>`
2. `export default {`

```

3.   data () { },
4.   mounted() { }
5. }
6. </script>

```

实际上，上面代码中，

- `data` 方法，是用于 " 声明页面会出现的变量 "，并且赋予初识值。(非常重要，切记这一点)
- `mounted` 表示页面被vue渲染好之后的钩子方法，会立刻执行。

所以，我们要把发送http的请求，写到mounted方法中。（钩子方法还有created，我们可以暂且认为mounted方法与created方法基本一样，一般我们在Vue 2.0中都使用mounted。后续会说到区别。）

```

1.   mounted() {
2.     this.$http.get('api/interface/blogs/all').then((response) => {
3.       this.blogs = response.body.blogs
4.     }, (response) => {
5.       console.error(response)
6.     });
7.   }

```

上面代码中：

- `this.$http` 中的
- `this` 表示当前的vue组件（也即 `BookList.vue`）
- `$http` 所有以 `$` 开头的变量，都是vue的特殊变量，往往是vue框架自带。这里的`$http`就是可以发起http请求的对象。
- `$http.get` 是一个方法，可以发起get 请求。只有一个参数就是目标url，
- `then()` 方法，来自于promise，可以把异步的请求写成普通的非异步形式。第1个参数是成功后的callback，第2个参数是失败后的callback。
- `this.blogs = response.body.blogs` 中，是把远程返回的结果（json），赋予到本地。由于javascript的语言特性，能直接支持json，所以才可以这样写。

然后，我们通过这个代码进行渲染：

```

1. <tr v-for="blog in blogs">
2.   <td>{{blog.title }}</td>
3. </tr>

```

在上面的代码中：

- `v-for` 是一个循环语法，可以把这个元素进行循环。注意：这个叫directive，指令，需要跟标签一起使用。
- `blog in blogs` : 前面的 `blog` 是一个临时变量，用于遍历使用。

后面的 `blogs` 是http 请求成功后，`this.blogs = ...` 这个变量。

同时，这个 `this.blogs` 是声明于 `data` 钩子方法中。

- `{{blog.title}}` 用来显示每个`blog.title`的值

如何发起post请求？

跟get特别类似，就是第二个参数是 请求的body。

在 vue的配置文件中（例如 webpack项目的 `src/main.js` 中）增加下面一句：

```

1. import VueResource from 'vue-resource';
2. Vue.use(VueResource);
3. ....
4.
5. //增加下面这句：
6. Vue.http.options.emulateJSON = true;

```

上面这句的目的，是为了能够让发出的post请求不会被浏览器转换成option 请求。

然后就可以按照下面的代码发送请求了：

```

1. this.$http.post('api/interface/blogs/all', {title: '', blog_body: ''})
2.   .then((response) => {
3.     ...
4.   }, (response) => {
5.     ...
6. });

```

在本书的《表单的提交》章节中，会对 http POST 的发送有个实际的例子，看起来会更加明白。

关于发送http请求的更多内容，请看： 官方文档：<https://github.com/pagekit/vue-resource>

在vue中操作什么属性，在接口中就要存在该属性的key .

例如，vue页面，会在初始化时，请求一个接口：

```
{ products: [ { name: 'a' comment: 'lalala' }, { name: 'b' comment: 'kakaka' } ] }
```

如果在接口中，这个comment不存在，那么我们在vue中怎么操作，都不会把‘comment’放到products中去。

发送层次很深的json 格式的参数

vuejs - 发送大量的请求时，使用 post + JSON.stringify(xx)的形式

2017-11-26 14:13访问量： 1

分类： 技术如题，我们从vuejs向接口发起请求时， 这样：

```
1.   this.$http.post(this.$configs.api + 'dish_orders/native_add_dish_order', {
2.     dishes: JSON.stringify(this.chosen_dishes),
3.   }).then((response) => {
```

服务器端会看到这样的日志：

```
{"socket_ips": "[{\"socketIPAddress\": \"192.168.1.181\", \"port\": 9100, \"printType\": 1, \"ticketTitle\": \"皇后镇餐厅总单\", \"categoryArray\": [1, 2, 3, 4, 5, 6, 11, 12, 13], \"categoryHashArray\": [{\"id\": 1, \"name\": \"特色菜\"}, {"id": 2, "name": "凉菜"}, {"id": 3, "name": "热菜"}, {"id": 4, "name": "烧烤"}, {"id": 5, "name": "面点、主食"}, {"id": 6, "name": "酒水饮料"}, {"id": 11, "name": "早点"}, {"id": 12, "name": "铁锅炖"}, {"id": 13, "name": "餐具"}]}, {"socketIPAddress": "192.168.1.182", "port": 9100, "printType": 0, "ticketTitle": "热菜、凉菜、特色菜、烧烤"}, ...]
```

然后，我们在rails端，这样解析：

```
dishes = ActiveSupport::JSON.decode(params[:dishes])
```

实战Vuejs

通过前面的学习，大家会对Vuejs有个非常全面的了解了。

接下来，我们用一个真实的项目，来使用Vuejs。

假设我们在一家互联网电子商务公司就职。该公司的业务，是帮助大山里的穷苦农民，把自家的一些特产放到网上售卖。

而我们要做的产品，就是要帮助这些农民，把商品卖出去。

解决的问题有三个：

1. 让农民把大山中的东西卖掉。（通过快递等途径）
2. 让都市中的人享受到纯原生态的绿色食品，以更便宜的价格。
3. 去掉中间商。保证农民的收入更多，消费者消费的价格更低。

通过这样的公益项目，公司也可以解决自身的生存问题。

需求文档

顺利需求是项目的重中之重，把老板的“一句话需求”，梳理成一系列的符合逻辑的文字，再进一步的整理成“可视化”的原型图。

所以，接下来，就是文字模块。

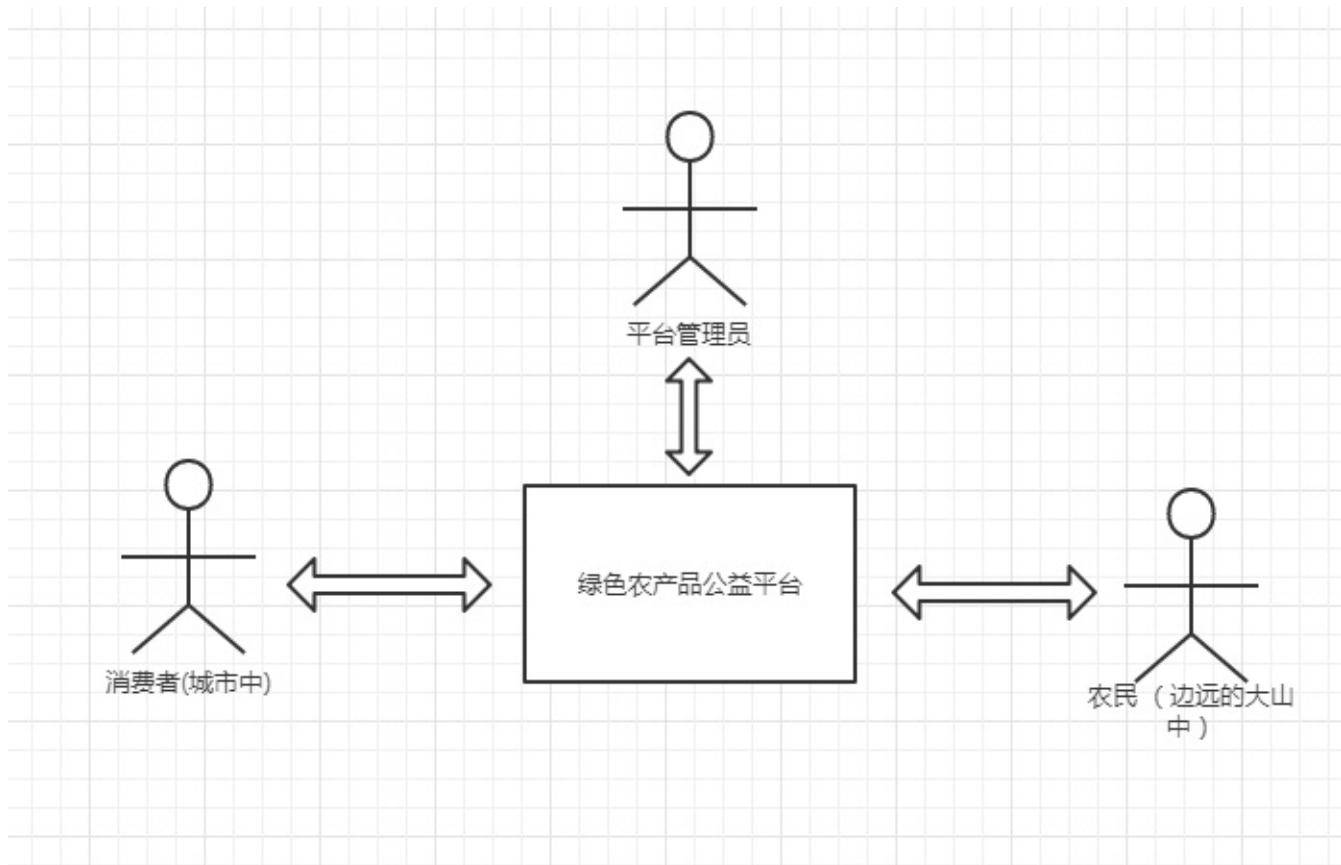
首先，我们先明确三种参与角色。（同学们记得，分析需求的时候，一定要图文并茂。站在一张大白板前面，手里一支笔。）

参与的角色

总共是三个角色参与。

边远大山中的农民，提供农产品城市中的消费者，来购买农产品平台的管理员，来对平台进行日常的运作

如下图所示：

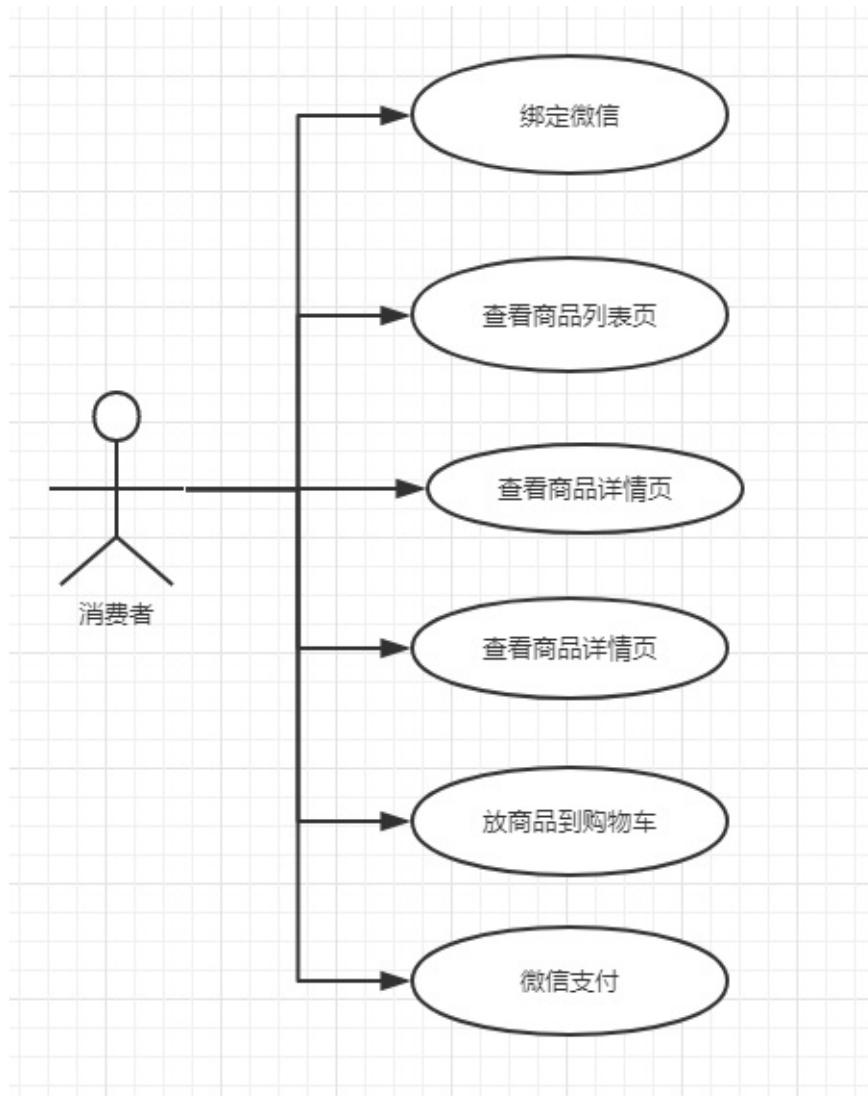


消费者

- 可以注册

- 可以微信授权
- 可以查看商品列表页
- 可以查看商品详情页
- 可以查看购物车
- 可以支付商品

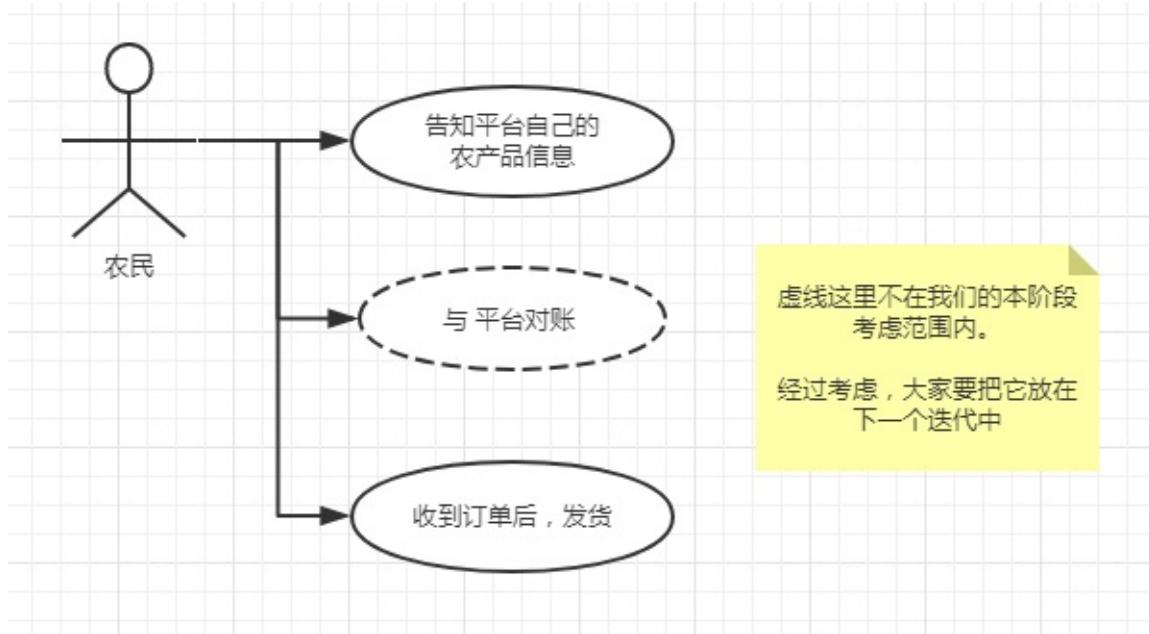
用户的用例图如下图所示：



农民

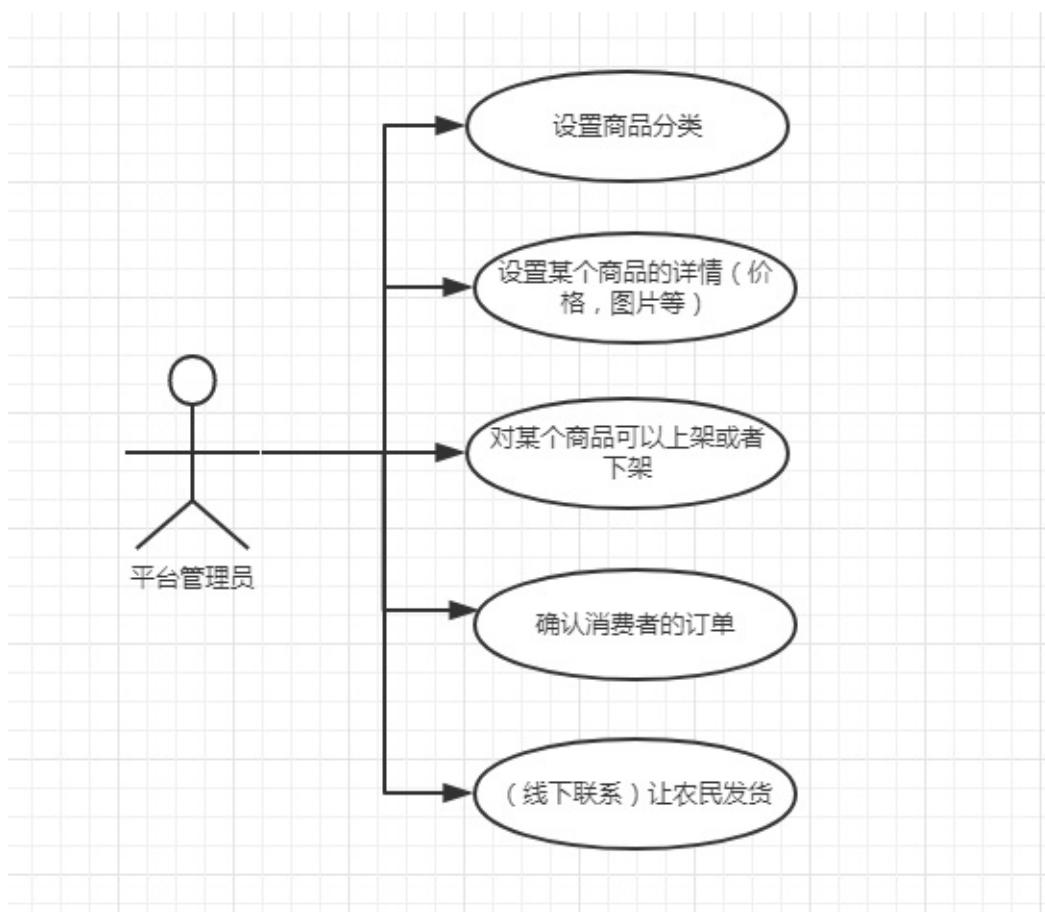
直接与公司联系, 告知可以出售的特产, 价格等信息

用例图如下图所示：



平台管理员

是公司的人员。可以管理商品分类可以管理商品的上架下架可以处理订单订单的付款确认后，发送快递。



原型图

UI 永远是程序员和产品经理沟通的最主要方式。

程序员可以同时看见UI和底层的东西。但是产品经理，以及我们的用户，只会看到UI。所以，任何一个程序员，在开始一个新项目的时候，都不要贸然的根据某段文字需求就开工。而是需要把“文字需求”转换成原型图。

虽然这个事情往往是由产品经理来做，但是我们作为有追求的程序员，一定要亲自具备这个能力。
(实际上程序员比产品经理更容易把原型图画出来)

先确定需要多少个页面

根据前面小节，我们知道了每个角色的主要任务（也叫use case）。也知道了只有消费者会使用微信端：

- 可以做微信绑定（有微信授权页面，这里不需要注册页面了。注册的过程是在后台直接做的）
- 可以看到首页（有首页）
- 可以看到商品列表页
- 可以看到商品详情页
- 可以看到购物车（有购物车页面）
- 可以看到个人页面
- 有支付页面

以上页面中，微信的授权和支付页面，可以根据直接调用微信SDK获得。其他页面，则需要我们来依次实现。

原型图

原型图，就是简笔画。

不要把它看得很难。画这个东西不需要门槛，一个人只要会用手机电脑，就应该会画。

建议新手直接动笔画，找一支笔，一张大白纸，心中想着自己要做的APP的样子，一个页面一个页面画出来便是。根据个人经验，基本一个不太复杂的APP，1-2个小时足够画出来了。不要怕丑陋难看。原型图没有人在意这个。

一旦有了动笔画的经验，下一步就可以用鼠标来画。市面上的原型图设计工具中，我最喜欢的还是Mockplus。简单好用。没有门槛。

下面就是若干原型图：

首页

用户打开连接后，直接进入到首页，如下图所示：

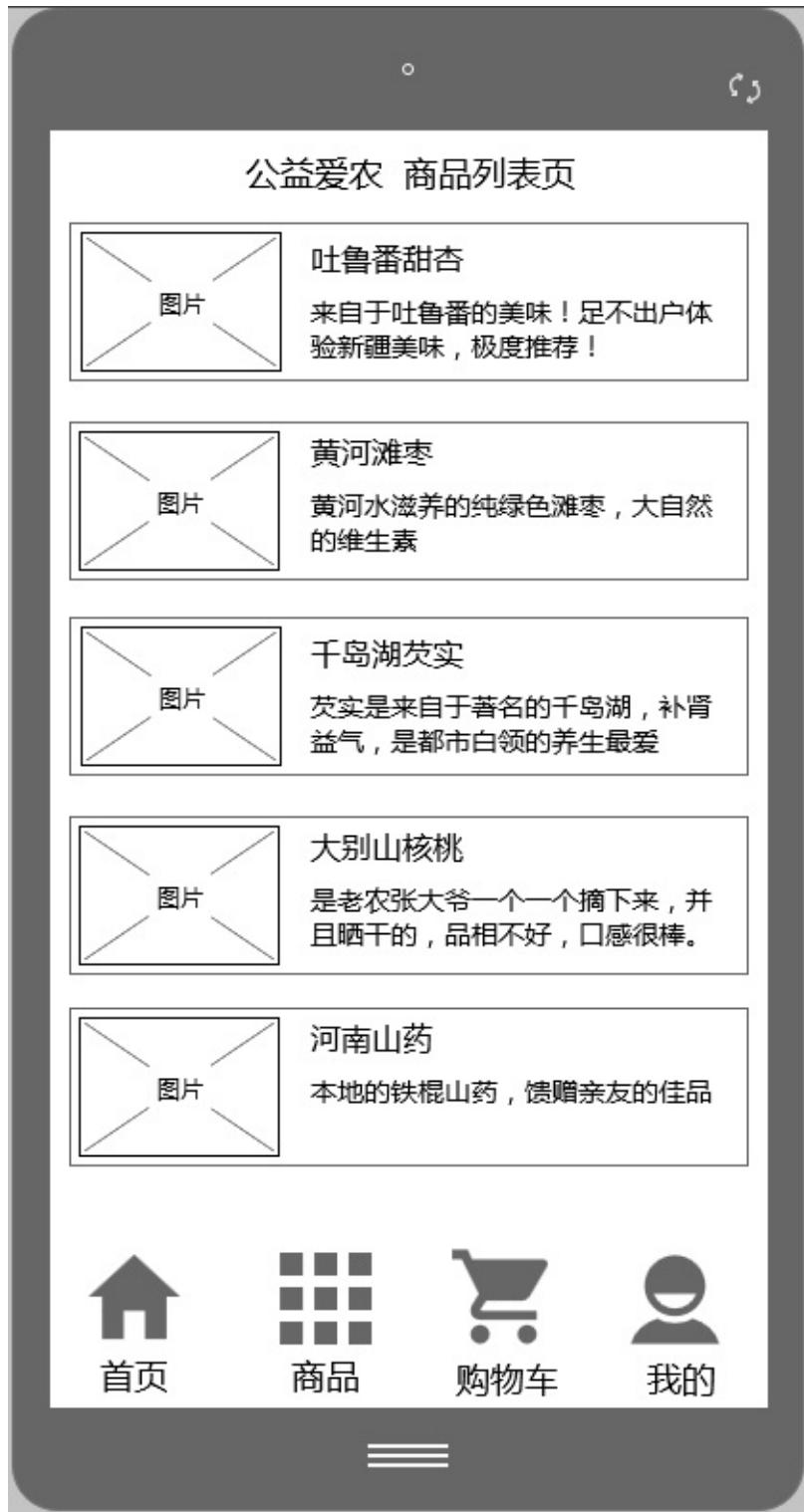


在首页中，

- 上部分是轮播中，
- 中间部分是 商品分类，
- 下方是商品列表
- 最下面是4个标签页：首页， 分类，购物车，我的。

商品列表页

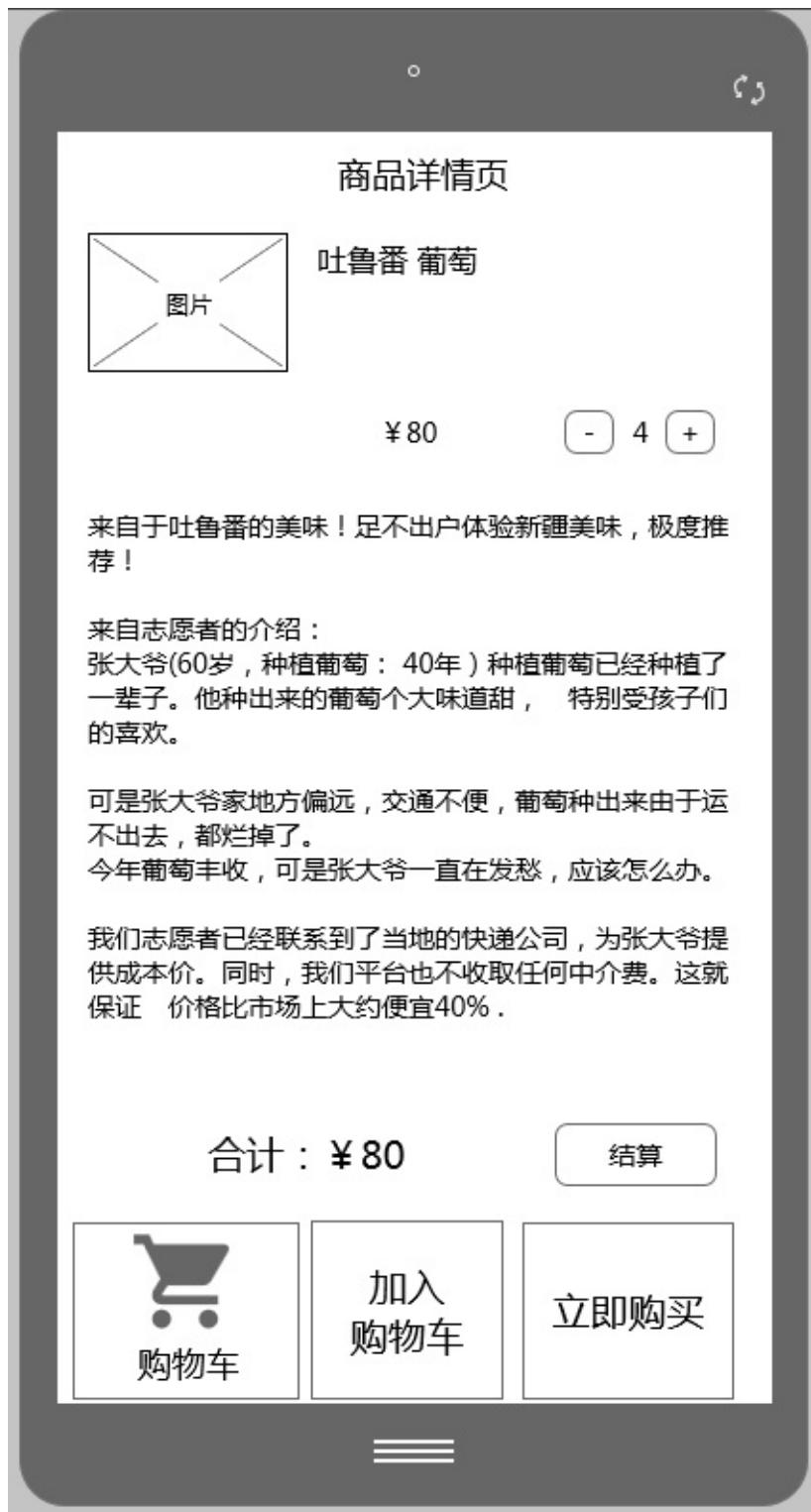
用户在首页点击分类，或者点击下面的第二个TAB后，进入到商品列表页， 如下图所示：



在本页中，会列出对应分类下的所有商品。

商品详情页

用户在商品列表页中点击某个商品后，会进入到商品详情页面。 如下图所示：



在该页面，可以看到商品的图文介绍，可以修改购买的数量，可以直接下单付款

购车页面

用户可以在查看商品的时候，把商品放到购物车中。然后在后续统一结算。如下图所示：



支付页面

用户可以在购物车中进行支付，也可以在商品购买页中进行支付，如下图所示：



即将支付页面，需要显示商品的各种信息，待付金额，用户的收货地址等。

确定全部信息无误后，就进入到微信的支付页面。（后续的页面由微信提供）

支付成功和失败页面略。

我的页面

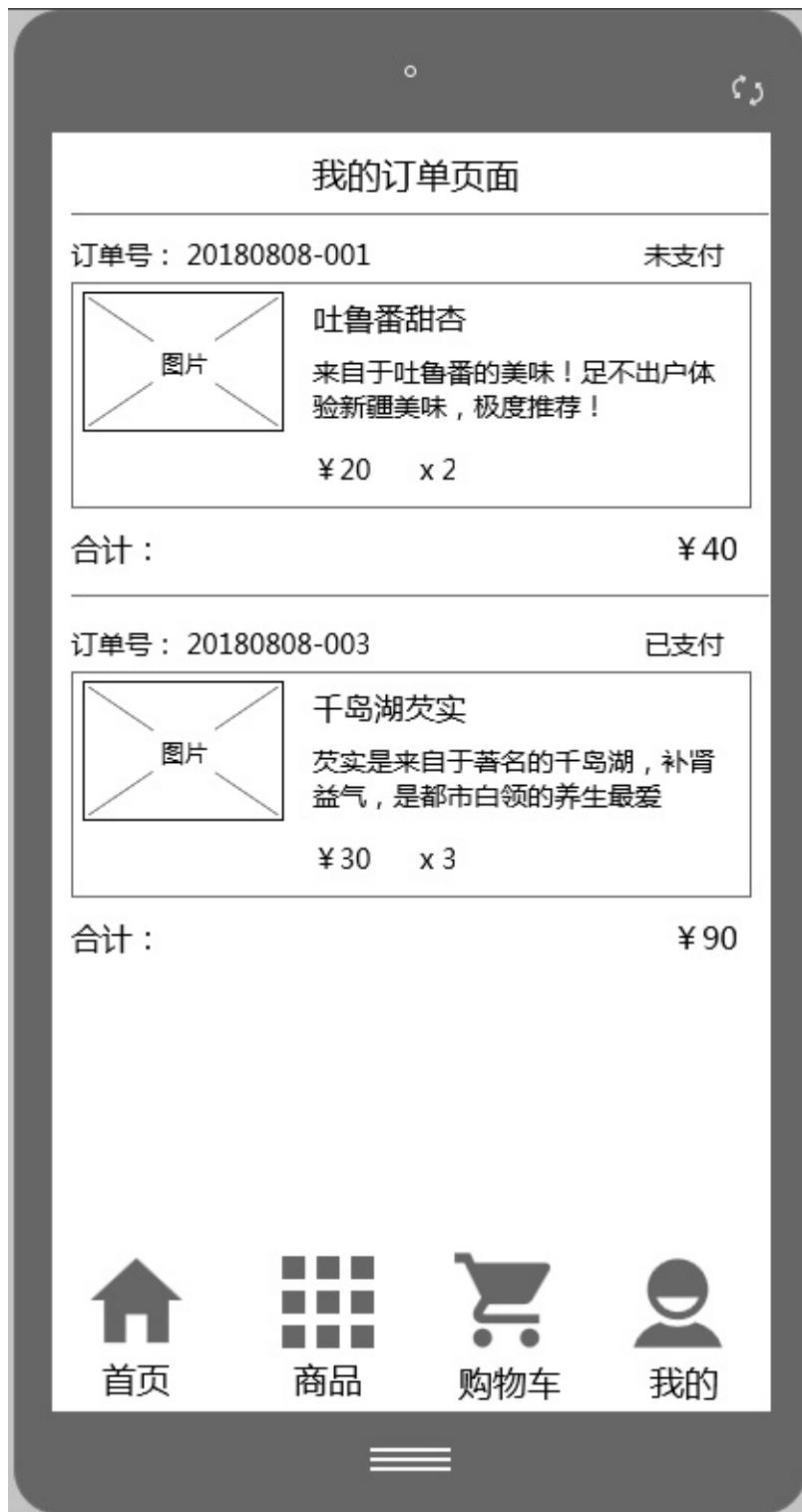
用户点击首页的第四个Tab，就可以进入到我的页面，也叫个人中心。如下图所示：



可以看到自己的头像， 昵称， 以及历史下单记录。

我的订单列表页面

用户在我的页面中， 点击订单列表， 就可以看到该页面。 如下图所示：



可以看到历史的订单， 每个订单的编号， 内容， 支付状态等信息。

总结

上面七个原型图，直接勾勒出我们要做的事情来。 其实还有很多功能都是一个商城应该必须的， 例如： 注册， 忘记密码， 管理地址， 客服等等。

但是为了抢时间，项目经理把需求做了排序， 保留了“不做项目就会死”的需求。 其他的需求可以放到下一周进行迭代。

注册微信的相关账号并下载开发者工具

三个平台账号

微信的H5页面会涉及到一些功能（登录，分享等）， 所以我们需要事先了解一下微信的产品家族。

- 微信公众平台： 包括服务号， 订阅号。 网址：<https://mp.weixin.qq.com>
- 微信开放平台： 给“手机App”提供登录分享等操作。 网址：<https://open.weixin.qq.com/>
- 微信商户平台： 提供微信支付的功能。 <https://pay.weixin.qq.com>

几乎无论是新手，还是老手，都会在这里发蒙。 所以大家务必做好必要的笔记。 另外，每申请一个账号， 就要把用户名和密码记录下来。 上面三个平台的账号都是独立的。 并且每个账号中几乎都有自己的appid, appkey, app secret 等各种机密的秘钥。 而且每个都不同。 大家一定要妥善保管好。 不能混淆。 否则会为调试带来巨大的困扰。

对于公司来说，需要准备好相关的证照， 并且在对应的时间内使用公司账号打款给微信。 由于过程比较繁琐， 所以申请的步骤略。

各位同学不要打退堂鼓，我们在真实的项目中，几乎每个互联网公司，都是一定会用到微信公众号的功能的。

下面的内容，我们假设已经成功的申请到了微信的相关账户， 公众平台上的是“服务号”。 （具有支付功能）

微信开发者工具

由于微信自带浏览器的特殊性（一定会自带weixin openid等微信独有的信息）， 这就导致我们平时使用的普通浏览器，在开发微信相关功能的时候（授权， 分享， 支付等）是无法使用的。 所以我们需要下载微信的开发者工具。 地址：

<https://developers.weixin.qq.com/miniprogram/dev/devtools/download.html>

如果上述地址有变化，我们也可以自行百度搜索：“微信开发者工具”。

下载页面如下图所示：

最新版本下载地址 (1.02.1808300)

[Windows 64位](#) / [Windows 32位](#) / [Mac OS](#)

Windows 仅支持 Windows 7 及以上版本。

2018.08.30

1. A 新增 版本管理 [详情](#)
2. A 新增 体验评分 [详情](#)
3. A 新增 npm 支持 [详情](#)
4. A 新增 英文版支持
5. A 新增 小程序分包预加载调试
6. A 新增 小程序独立分包调试
7. A 新增 plugin.json 中新增 page 会自动生成页面
8. A 新增 代码片段管理反选功能
9. A 新增 wxml 标签属性支持数字
10. A 新增 编辑器折叠所有文件夹功能
11. A 新增 项目配置支持 `debugOptions` 选项，可以隐藏指定源文件避免调试器不响应 [详情](#)

下载后，双击开始安装， 安装后会出现登录页面， 如下图所示：



微信开发者工具

v1.02.1808300



欢迎使用微信开发者工具

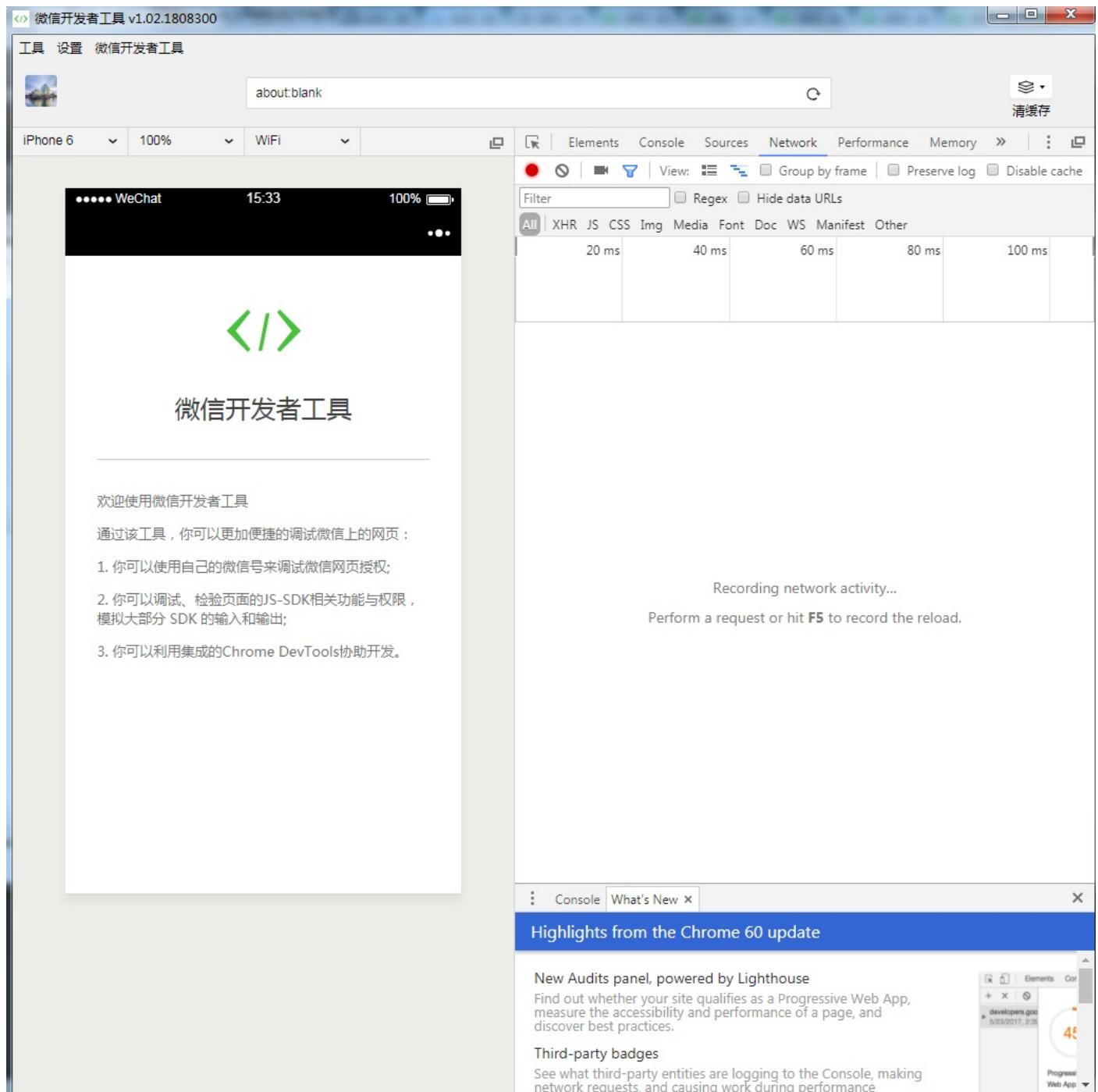
扫描后, 可以看到有两个入口, 一个是微信小程序, 另一个是公众号网页项目, 如下图所示:



打开后, 可以看到, 几乎跟chrome developer tool是一模一样的。除了:

- 左上角提供了wifi的信号选择.
- 右上角提供了“清缓存”这个功能。
- 左上角可以看到当前登录的微信用户的图标

如下图所示:



目前虽然网上有一些Linux的开源组件，但是基本没有windows和mac上的好用。所以建议Linux的开发同学，暂时回到windows的开发环境。

团队架构和其他准备

经过考虑,这个任务需要一周内完成第一版, 负责该项目的小组成立了,
包括: 前端工程师(也就是我们), 后端工程师, 项目经理(梳理需求) , 和一位UI设计师同学.
大家的任务如下:

项目经理:

1. 负责梳理老板的需求, 形成文字
2. 负责把文字版的需求, 画成原型图, 给前后端工程师参考.
3. 每周的任务进度控制.
4. 承担测试人员的工作. 对已经完成的功能进行测试.

前端工程师

1. 负责呈现给用户使用的界面. 包括 H5端.
2. 负责与后端同学沟通API方面的问题

后端工程师

1. 负责后端管理员的操作界面
2. 负责给前端实现API接口
3. 负责数据库的设计, 实现等

UI设计师

完成对于前端操作界面的设计.

要有个bug跟踪系统， 有个wiki

一定要有, 但不要过分依赖。 我只用它做两件事:

- bug跟踪: 记录每天要做的事
- wiki: 专门记录 前端 和 后端 所用到的API的格式.

开源的redmine 很棒。 两者都满足。 如下图所示:

← → C ① bug.sweetysoft.com/projects/si_lu_le_gou/issues

主页 我的工作台 项目 管理 帮助

丝路乐购

概述 工作简报 问题 新建问题 Wiki 配置

问题

过滤器

状态 打开 ▾

选项

应用 清除 保存

| 序号 | # | 跟踪 | 状态 | 优先级 | 主题 |
|----|----------------------|-----|-----------|-----|----------------------------|
| 1 | 4149 | Bug | 问题已分发 | 正常 | 完成订单跟踪模块的内容。 使用 阿里云的 查询接口。 |
| 2 | 4148 | Bug | 问题已分发 | 正常 | 完成购物车模块的所有内容 |
| 3 | 4147 | Bug | 问题已分发 | 正常 | 完善商城 订单模块的内容。 完成后台订单模块的逻辑。 |
| 4 | 4146 | Bug | 问题已解决,待确认 | 正常 | 完成支付宝支付模块的所有内容。 |
| 5 | 3881 | Bug | 问题已分发 | 正常 | web - 有新订单时,发送短信到商城负责人. |
| 6 | 3880 | Bug | 问题已解决,待确认 | 正常 | h5 - 商品轮播图 |
| 7 | 3879 | Bug | 问题已解决,待确认 | 正常 | h5 - 我的订单 |
| 8 | 3878 | Bug | 问题已解决,待确认 | 正常 | h5 - 首页轮播图 , 做成活的 . |

要有个代码仓库

国外的github， 国内的 coding.net都很不错。

对于初学者，最容易翻的错误就是不用版本控制。 同学们，一定记住，任何程序，都要用git保管起来。

对于目前的项目，需要两个，一个是h5端(shop_h5)， 一个后台(shop_web)。

整体项目的搭建

为新项目生成骨架

创建一个基于 webpack 模板的新项目：

```
1. $ vue init webpack shop_h5
```

安装依赖：

```
1. $ cd shop_h5  
2. $ cnpm install
```

在本地，以默认端口来运行：

```
1. $ npm run dev
```

然后就可以看到 在本地已经跑起来了。

在这个阶段， 我们只需要修改它的标题就可以：

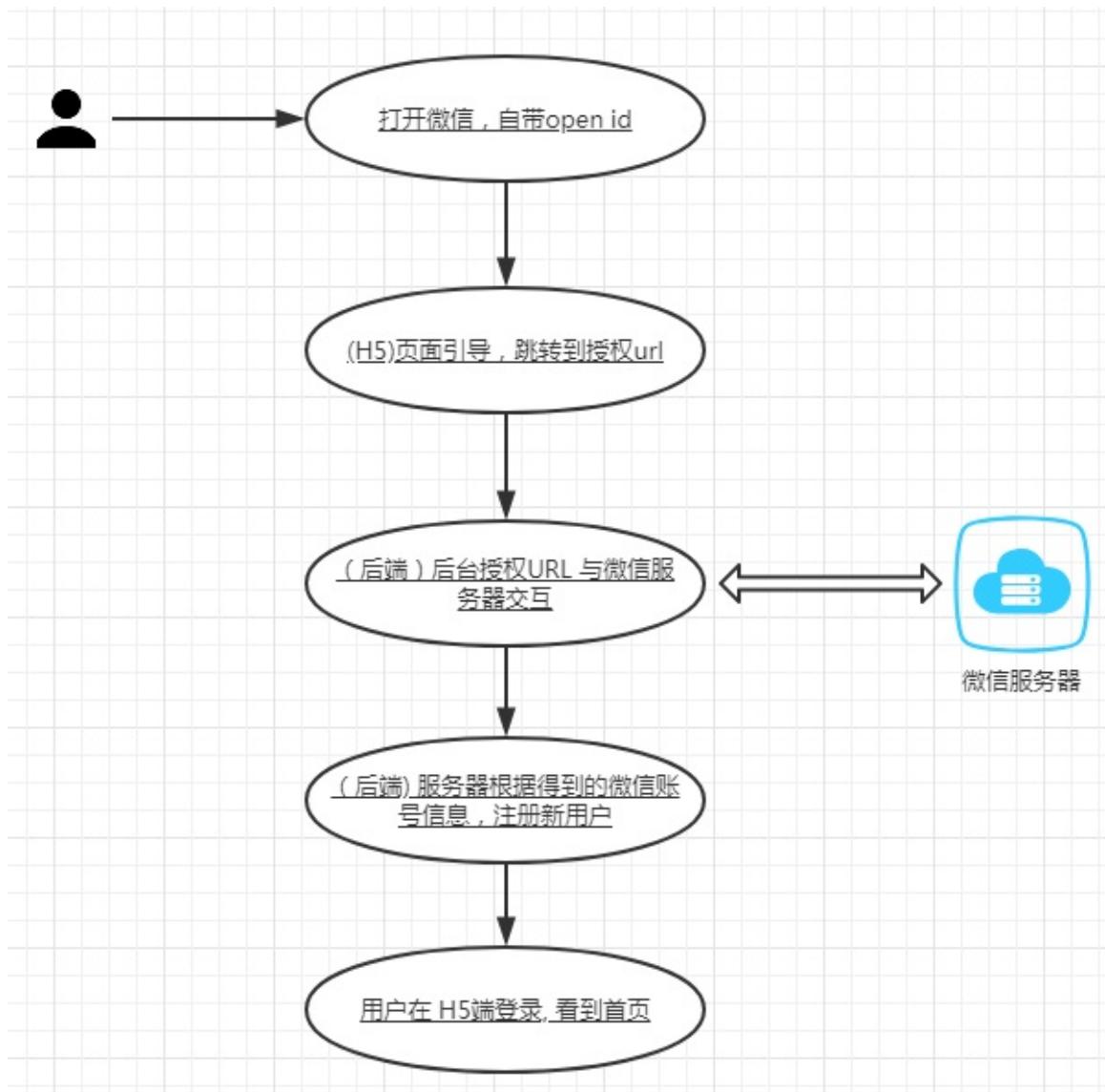
打开 根目录下的 index.html ， 修改

用户的注册和微信授权

为了追求快速上线，项目组决定，去掉传统项目中的“用户注册”，“用户登录”，直接使用微信的账户来核对。

1. 用户的微信浏览器带着当前微信用户的open_id，跳转到“后台服务器”。
2. “后台服务器” 给“微信服务器” 发送请求。 获得当前微信用户的信息
3. “后台服务器” 为该用户生成一个用户文件
4. “后台服务器” 告知“H5端” 已经成功注册该用户。
5. “H5端” 为该用户展示对应的页面。

如下图所示：



可以看出，主要代码都是在 服务器端。

1. 让微信用户打开首页后，直接跳转到后台服务器

1.1 修改对应的路由文件(src/router/index.js)：

```
1. Vue.use(Router)
2.
3. export default new Router({
4.   routes: [
5.     {
6.       path: '/wait_to_shouquan',
7.       name: 'wait_to_shouquan',
8.       component: require('../views/wait_to_shouquan.vue')
9.     },
10.   ]
11. })
```

1.2 增加对应的vue(src/views/wait_to_shouquan.vue)：

```
1. <template>
2.   <div style="padding: 50px;">
3.     <h3>正在跳转到授权界面...</h3>
4.   </div>
5. </template>
6.
7. <script>
8.   export default {
9.     created () {
10.       window.location.href = this.$store.state.web_share + "/auth/wechat"
11.     },
12.     components: {
13.     }
14.   }
15. </script>
```

可以看到，上面的代码中，使用到了 Vuex 来保存系统变量（后台服务器的地址）。

1.3 增加核心模板文件(src/main.vue)：

```
1. <template>
2.   <div id="app">
3.     <router-view></router-view>
4.   </div>
5. </template>
```

```
6.
7. <script>
8. import store from './vuex/store'
9. import { SET_BASEINFO, GET_BASEINFO } from './vuex/mutation_types'
10. export default {
11.   store,
12.   name: 'app',
13.   data () {
14.     return {
15.       user_info: {
16.         open_id: this.$route.query.open_id
17.       }
18.     }
19.   },
20.   mounted () {
21.
22.     // TODO 开发环境下使用， 生产环境下注释掉
23.     // store.dispatch(SET_BASEINFO, {open_id: 'opFELv6YkJkMaH-xFkokTWCs5AlQ'})
24.
25.     if (this.user_info.open_id) {
26.       store.dispatch(SET_BASEINFO, this.user_info)
27.     } else {
28.       store.dispatch(SET_BASEINFO)
29.       if (store.state.userInfo.open_id === undefined) {
30.         console.info('用户id和open_id不存在， 跳转到授权等待页面')
31.         this.$router.push({name: 'wait_to_shouquan'})
32.       } else {
33.         console.info('已经有了BASEINFO')
34.       }
35.     }
36.   },
37.   watch: {
38.     '$route' (val) {
39.     }
40.   },
41.   methods: {
42.   },
43.   components:{}
44.   }
45. }
46. </script>
47. <!-- 下方的 CSS 略过 -->
```

可以看到，上面代码的 `mounted()` 方法中，会对当前用户的 `open_id` 进行判断。如果存在，调到首页。如果不存在，表示该用户是新用户。需要跳转到授权等待页面。对应的代码如下所示：

```

1.     if (this.userInfo.open_id) {
2.         store.dispatch(SET_BASEINFO, this.userInfo)
3.     } else {
4.         store.dispatch(SET_BASEINFO)
5.         if (store.state.userInfo.open_id === undefined) {
6.             console.info('用户id和open_id不存在，跳转到授权等待页面')
7.             this.$router.push({name: 'wait_to_shouquan'})
8.         } else {
9.             console.info('已经有了BASEINFO')
10.        }
11.    }

```

1.4 增加对应的Vuex代码

目前来看，Vuex需要保存2个信息：

- 用户的 `open id`
- 远程服务器的地址，端口等常量。

1.4.1 增加 `src/vuex/store.js` . 这个是最最核心的文件。完整如下所示：

```

1. import Vue from 'vue'
2. import Vuex from 'vuex'
3. import userInfo from './modules/user_info'
4. import tabbar from './modules/tabbar'
5. import toast from './modules/toast'
6. import countdown from './modules/countdown'
7. import products from './modules/products'
8. import shopping_car from './modules/shopping_car'
9.
10. import * as actions from './actions'
11. import * as getters from './getters'
12.
13. Vue.use(Vuex)
14. Vue.config.debug = true
15.
16. const debug = process.env.NODE_ENV !== 'production'
17.
18. export default new Vuex.Store({
19.     state: {

```

```

20.     web_share: 'http://shopweb.siwei.me',
21.     h5_share: 'http://shoph5.siwei.me/?#'
22.   },
23.   actions,
24.   getters,
25.   modules: {
26.     products,
27.     shopping_car,
28.     userInfo,
29.     tabbar,
30.     toast,
31.     countdown
32.   },
33.   strict: debug,
34.   middlewares: debug ? [] : []
35. })

```

上面的代码中，部分代码是在后面会陆续用到的。我们不用过多考虑。只需要关注下面几行：

```

1. export default new Vuex.Store({
2.   // 这里定义了若干系统常量
3.   state: {
4.     web_share: 'http://shopweb.siwei.me',
5.     h5_share: 'http://shoph5.siwei.me/?#'
6.   },
7.
8.   modules: {
9.     // 这里定义了 当前用户的各种信息，我们把它封装成为一个js对象
10.    userInfo,
11.  },
12.
13. })

```

1.4.2 增加 `vuex/modules/user_info.js` 这个文件：

```

1. import {
2.   SET_BASEINFO,
3.   CLEAR_BASEINFO,
4.   GET_BASEINFO,
5.   COMMEN_ROLE,
6.   GET_BGCOLOR,
7.   GET_FONTCOLOR,

```

```
8.     GET_BORDERCOLOR,
9.     GET_ACTIVECOLOR,
10.    EXCHANGE_ROLE
11. } from './mutation_types'
12.
13. const state = {
14.   id: undefined, //用户id
15.   open_id: undefined, // 用户open_id
16.   role: undefined
17. }
18.
19. const mutations = {
20.   //设置用户个人信息
21.   [SET_BASEINFO] (state, data) {
22.     try {
23.       state.id = data.id
24.       state.open_id = data.open_id
25.       state.role = data.role
26.     } catch (err) {
27.       console.log(err)
28.     }
29.   },
30.   //注销用户操作
31.   [CLEAR_BASEINFO] (state) {
32.     console.info('清理缓存')
33.     window.localStorage.clear()
34.   },
35. }
36.
37. const getters = {
38.   [GET_BASEINFO]: state => {
39.     console.info('进入到了getter中了')
40.     let localStorage = window.localStorage
41.     let user_info
42.     if (localStorage.getItem('SLLG_BASEINFO')) {
43.       console.info('有数据')
44.       user_info = JSON.parse(localStorage.getItem('SLLG_BASEINFO'))
45.     } else {
46.       console.info('没有数据')
47.     }
48.     return user_info
49.   },

```

```

50. [COMMEN_ROLE]: state => {
51.     if (state.role === 'yonghu') {
52.         return true
53.     } else {
54.         return false
55.     }
56. },
57. }
58.
59.
60.
61. const actions = {
62.     [SET_BASEINFO] ({ commit, state }, data) {
63.         //保存信息
64.         if (data !== undefined) {
65.             let localStorage = window.localStorage
66.             localStorage.setItem('BASEINFO', JSON.stringify(data))
67.             commit(SET_BASEINFO, data)
68.         } else {
69.             if (localStorage.getItem('BASEINFO')) {
70.                 data = JSON.parse(localStorage.getItem('BASEINFO'))
71.                 commit(SET_BASEINFO, data)
72.             } else {
73.             }
74.         }
75.     }
76. }
77.
78. export default {
79.     state,
80.     mutations,
81.     actions,
82.     getters
83. }

```

该文件定义了用户的信息的各种属性。

1.4.3 增加 `src/vuex/mutation_types.js`

```

1. export const SET_BASEINFO = 'SET_BASEINFO'
2. export const GET_BASEINFO = 'GET_BASEINFO'

```

上面内容定义了该对象的两个操作。

1.4.4 增加 `src/vuex/modules/actions.js`

```
1. import * as types from './mutation_types'
```

可以看到上面的内容对于 `mutation_types` 进行了引用。

1.4.5 增加 `src/vuex/modules/getters.js`

```
1. // 先放成空内容好了
```

这个文件先这样存在，目前阶段不需要有任何内容。

1.5 与后台的对接

后台的同学，为我们提供了一个链接入口：

<http://shopweb.siwei.me/auth/wechat>

我们让H5页面直接跳转过去就可以。这里不需要加任何参数，直接由后端的同学搞定接下来的事情。

查看效果

在微信开发者工具中，打开我们的H5 页面，就会看到页面会自动跳转。

观察仔细的同学可以看到有两次跳转：

第一次跳转到 <http://shopweb.siwei.me/auth/wechat> 第二次跳转到

<https://open.weixin.qq.com/connect/oauth2/authorize...>

如下图所示：



明创软件

网页由该公众号开发，请确认授权以下信息

- 获得你的公开信息（昵称、头像等）

确认登录

点击“确认登录”按钮，进行授权后，就会进入到H5的首页

总结

本节中，为了做一件事：让用户跳转到微信授权，并注册，我们做了如下的程序层面的内容：

- 使用Vuex 记录系统常量(远程服务器的地址)
- 使用Vuex 记录用户的信息（新增了一个对象：user_info）
- 使用了一个独立的页面（等待微信授权页面）
- 每次打开首页之前，都要判断该用户是否登录。
- （后台任务）让该用户在微信端授权，并且在本地生成一个新的用户，然后把相关数据返回给前端

Vuex 是Vue.js 最复杂最不好理解的地方。同学们不要怕。之所以这么麻烦，可以认为是

javascript的语言特性决定的。 就好像java, C语言中大量用到的设计模式， 在现代编程语言(Ruby , Python, Perl)中就用不到， 可以有更加简单的办法， 例如Mixin)

另外，本节对于后端的同学是个挑战， 不但要在微信端做修改，还需要对微信返回的数据结构很熟悉。 由于本书内容所限，不再赘述。

登陆状态的保持

如何判断当前用户是否登录呢？

对于经典的web开发（后台开发），都是使用当前用户的session。把登录信息保存进去，每次需要的时候读取就可以了。

对于前端，从理论上讲有两种方式：

1. 读取cookie（适合所有h5框架）
2. 读取vuex（适合Vuejs）

下面我们分别来说一下。

简单版： 使用cookie

cookie是明文存储。可以直接调用 `document.cookie` 来实现。例如，打开浏览器的 console，

输入：

```
document.cookie
```

返回：

```
"TY_SESSION_ID=fad74371-40d1-444b-9d1c-5dd33c086b20; uuid_tt_dd=10_37220323210-1535781572649-914811; dc_session_id=10_1535781572649.670168; Hm_lvt_6bcd52f51e9b3dce32bec4a3997715ac=1535781569,1535873915; dc_tos=pef3wv; 1. Hm_lpvt_6bcd52f51e9b3dce32bec4a3997715ac=1535873935"
```

里面有哪些内容一目了然。

所以，从安全性的角度讲，用户的登录信息容易被人弄走。

使用Vuex

Vuex 虽然不好上手，但是一旦熟悉了它的写法，还是很容易的。工作量两者差不多。

另外，Vuex的存储虽然是使用了Cookie，但是会对很多信息进行封装和作用域的判断。就算被人拿到，也不会轻易的泄露信息。

保存信息：

```
1. store.dispatch(SET_BASEINFO, this.userInfo)
```

读取信息：

```
1. store.state.userInfo
```

对于Vuex的代码细节， 请看前一节 .

移动端的H5要保存哪些信息？

1. 保存的越少越好。 一个唯一的用户标识就好了。
2. 不要泄露数据库的情况。

差的例子：

'xiao_wang_user_id'， 这个不用猜， 被人一读， 就会大概估计到， 是小王的用户id'140'， �恩。 好的。 当前用户在数据库中的id是140 。 下一个用户肯定是141号了。

好的例子：

'a1b2c3d4e5f6'， 微信的open_id 就是这样的。 不给黑客任何机会。

轮播图

用户登录到首页之后，第一个看到的内容，就是轮播图。（见前面的原型图）

所以，我们接下来，要为首页增加它。

1. 增加路由

路由文件(src/router/index.js)，对应位置，增加：

```
1. import Index from '@/views/shops/index'  
2.  
3. Vue.use(Router)  
4.  
5. export default new Router({  
6.   routes: [  
7.     {  
8.       path: '/',
9.       name: 'Home',
10.      component: Index
11.    }
12.  ]
```

2. 增加对应页面

src/views/shops/index.vue

```
1. <template>
2.   <div class="background">
3.     <div class="home">
4.       <div class="m_layout">
5.         <!-- 轮播图-->
6.         <HomeBannerView></HomeBannerView>
7.         <!--导航-->
8.         <HomeNavView></HomeNavView>
9.
10.        <span class="divider" style="height: 4px;"></span>
11.        <div class="product_top">
12.          <div class="product_left">
```

```
13.          <div>商品列表</div>
14.      </div>
15.      </div>
16.      <span class="divider" style="height: 2px;"></span>
17.      <!-- 特产商品 -->
18.      <SpecialMarket :id="good.id" :name="good.name"
   :description="good.description" :image_url="good.image_url" v-for="good in
18. goods"></SpecialMarket>
19.      </div>
20.      </div>
21.      <NavBottomView :is_shops_index="is_shops_index"></NavBottomView>
22.      </div>
23.  </template>
24. <script>
25. // 轮播图需要的js文件
26. import {bindEvent, scrollPic} from '../../libs/index.js'
27. // 轮播图需要的前台文件
28. import HomeBannerView from '../../components/HomeBanner.vue';
29. // 商品分类
30. import HomeNavView from '../../components/HomeNav.vue';
31.
32. // 特产, 商品的列表
33. import SpecialMarket from '../../components/SpecialMarket.vue';
34.
35. // 底部4个TAB, 下一节会讲到
36. import NavBottomView from '../../components/NavBottom.vue';
37.
38. export default{
39.     data () {
40.         return {
41.             goods: [],
42.             is_shops_index: true,
43.         }
44.     },
45.     components:{
46.         HomeHeaderView,
47.         HomeBannerView,
48.         HomeNavView,
49.         HomeMainView,
50.         SpecialMarket,
51.         NavBottomView
52.     },
53.     mounted () {
```

```

54.         //bindEvent();
55.         scrollPic();
56.         this.loadPage ();
57.     },
58.     computed: {
59.     },
60.     methods: {
61.         loadPage () {
62.             this.$http.get(this.$configs.api +
63. 'goods/get_goods').then((response)=>{
64.                 console.info(response.body)
65.                 this.goods= response.body.goods
66.             },(error) => {
67.                 console.error(error)
68.             });
69.         }
70.     }
71. </script>

```

上面的代码中，

- 先是读取了一个API
- 渲染数据
- 实现首页的轮播图

3. 增加轮播图

我们做开发的核心方法论： 不要造轮子。

一定要造轮子的话， 先搜索一下是否已经有了现成的轮子。

轮播图是最最常见的组件， 所以一定会有别人写好的第三方包， 我们拿过来用就好。

经过一番考察， 我们发现这个项目不错：

所以， 直接拿过来用。

3.1 轮播图的组件

我们需要在src/views/shops/index.vue中，增加：

```
1. import {bindEvent,scrollPic} from '../../libs/index.js'
```

然后， 增加对应的文件 (libs/index.js)

```
1. 
2. function $id(id) {
3.     return document.getElementById(id);
4. }
5.
6. function bindEvent() {
7.     var sea = $id("my_search");
8.     /*banner对象*/
9.     var banner = $id("my_banner");
10.    /*高度*/
11.    var height = banner.offsetHeight;
12.    window.onscroll = function() {
13.        var top = document.body.scrollTop;
14.        /*当滚动高度大于banner的高度时候颜色不变*/
15.        if (top > height) {
16.            sea.style.background = "rgba(201,21,35,0.85)";
17.        } else {
18.            var op = top / height * 0.85;
19.            sea.style.background = "rgba(201,21,35," + op + ")";
20.        }
21.    };
22. }
23.
24. function scrollPic() {
25.     var imgBox = document.getElementsByClassName("banner_box")[0];
26.     var width = $id("my_banner").offsetWidth;
27.     var pointBox = document.getElementsByClassName("point_box")[0];
28.     var ols = pointBox.children;
29.     var indexx = 1;
30.     var timer = null;
31.     var moveX = 0;
32.     var endX = 0;
33.     var startX = 0;
34.     var square = 0;
35.
36.     function addTransition() {
37.         imgBox.style.transition = "all .3s ease 0s";
38.         imgBox.style.webkitTransition = "all .3s ease 0s";
39.     }
40. }
```

```
41.     function removeTransition() {
42.         imgBox.style.transition = "none";
43.         imgBox.style.webkitTransition = "none";
44.     }
45.
46.     function setTransfrom(t) {
47.         imgBox.style.transform = 'translateX(' + t + 'px)';
48.         imgBox.style.webkitTransform = 'translateX(' + t + 'px)';
49.     }
50.
51.     // 开始动画部分
52.     pointBox.children[0].className = "now";
53.     for (var i = 0; i < ols.length; i++) {
54.         ols[i].index = i; // 获得当前第几个小li 的索引号
55.         ols[i].onmouseover = function() {
56.             // 所有的都要清空
57.             for (var j = 0; j < ols.length; j++) {
58.                 ols[j].className = "";
59.             }
60.             this.className = "now";
61.             setTransfrom(-indexx * width);
62.             square = indexx;
63.         }
64.     }
65.     timer = setInterval(function() {
66.         indexx++;
67.         addTransition();
68.         setTransfrom(-indexx * width);
69.         // 小方块
70.         square++;
71.         if (square > ols.length - 1) {
72.             square = 0;
73.         }
74.         // 先清除所有的
75.         for (var i = 0; i < ols.length; i++)
76.         {
77.             ols[i].className = "";
78.         }
79.         // 留下当前的
80.         ols[square].className = "now";
81.     }, 3000);
82.
```

```
83.     imgBox.addEventListener('transitionEnd', function() {
84.         if (indexx >= 9) {
85.             indexx = 1;
86.         } else if (indexx <= 0) {
87.             indexx = 8;
88.         }
89.         removeTransition();
90.         setTransfrom(-indexx * width);
91.     }, false);
92.
93.     imgBox.addEventListener('webkitTransitionEnd', function() {
94.         if (indexx >= 9) {
95.             indexx = 1;
96.         } else if (indexx <= 0) {
97.             indexx = 8;
98.         }
99.         removeTransition();
100.        setTransfrom(-indexx * width);
101.    }, false);
102.
103.    /**
104.     * 触摸事件开始
105.     */
106.    imgBox.addEventListener("touchstart", function(e) {
107.        console.log("开始");
108.        var event = e || window.event;
109.        //记录开始滑动的位置
110.        startX = event.touches[0].clientX;
111.    }, false);
112.
113.    /**
114.     * 触摸滑动事件
115.     */
116.    imgBox.addEventListener("touchmove", function(e) {
117.        console.log("move");
118.        var event = e || window.event;
119.        event.preventDefault();
120.
121.        //清除定时器
122.        clearInterval(timer);
123.        //记录结束位置
124.        endX = event.touches[0].clientX;
```

```
125.         //记录移动的位置
126.         moveX = startX - endX;
127.         removeTransition();
128.         setTransfrom(-indexx * width - moveX);
129.     }, false);
130.
131. /**
132. * 触摸结束事件
133. */
134. imgBox.addEventListener("touchend", function() {
135.     console.log("end");
136.     //如果移动的位置大于三分之一，并且是移动过的
137.     if (Math.abs(moveX) > (1 / 3 * width) && endX != 0) {
138.         //向左
139.         if (moveX > 0) {
140.             indexx++;
141.         } else {
142.             indexx--;
143.         }
144.         //改变位置
145.         setTransfrom(-indexx * width);
146.     }
147.     //回到原来的位置
148.     addTransition();
149.     setTransfrom(-indexx * width);
150.     //初始化
151.     startX = 0;
152.     endX = 0;
153.
154.     clearInterval(timer);
155.     timer = setInterval(function() {
156.         indexx++;
157.         square++;
158.         if (square > ols.length - 1) {
159.             square = 0;
160.         }
161.         // 先清除所有的
162.         for (var i = 0; i < ols.length; i++) {
163.             {
164.                 ols[i].className = "";
165.             }
166.         // 留下当前的

```

```

167.         ols[square].className = "now";
168.         addTransition();
169.         setTransfrom(-indexx * width);
170.
171.         // 每3秒钟轮播图变化一次。
172.         }, 3000);
173.     }, false);
174. };
175.
176. module.exports = {
177.     bindEvent,
178.     scrollPic
179. }

```

3.2 轮播图的视图层

```

1. <!-- 轮播图-->
2. <HomeBannerView></HomeBannerView>

```

上面的代码， 会直接生成一个轮播图。

我们创建这个component， 增加： src/components/HomeBanner.vue，

```

1. <template>
2.   <div class="home_ban">
3.     <div class="m_banner clearfix" id="my_banner">
4.       <ul class="banner_box">
5.         <!-- 更改这里就可以替换轮播图的图片了 -->
6.         <li></li>
7.         <li></li>
8.       </ul>
9.       <ul class="point_box" >
10.        <li></li>
11.        <li></li>
12.      </ul>
13.    </div>
14.  </div>
15. </template>

```

这个component的内容非常简单。 它只是一个轮播图组件的View. 我们只需要更换上面代码中的两个图片文件就好了。

4. 增加物品分类

```
1. <!--商品分类-->
2. <HomeNavView></HomeNavView>
```

上面的代码， 会直接生成一个物品分类区域

我们创建这个component：增加： src/components/HomeNav.vue

```
1. <template>
2.   <div class="home_n">
3.     <nav class="m_nav">
4.       <ul>
5.         <li class="nav_item">
6.           <a href="#" class="nav_item_link">
7.             
8.             <span>草原特色肉</span>
9.           </a>
10.        </li>
11.        <li class="nav_item">
12.          <a href="#" class="nav_item_link">
13.            
14.            <span>特色干果</span>
15.          </a>
16.        </li>
17.        <li class="nav_item">
18.          <a href="#" class="nav_item_link">
19.            
20.            <span>特色瓜子</span>
21.          </a>
22.        </li>
23.        <li class="nav_item">
24.          <a href="#" class="nav_item_link">
25.            
26.            <span>特色大米</span>
27.          </a>
28.        </li>
29.      </ul>
```

```
30.          </nav>
31.      </div>
32.  </template>
```

之所以这样写，是考虑到前期的商品分类不多。在下一个版本会改成读取后台的接口。

看效果

默认页面效果如下：



三秒钟之后，轮播图发生了滚动：



商品列表



黄河滩枣

母亲河的馈赠!

总结

我们使用了 现成的轮播图组件。 可以看到， 非常简单。 步骤是：

1. 复制对应组件的js 文件到src/lib下。
2. 复制对应组件的vue文件到src/components下
3. 在对应的vue文件中使用它们。

底部Tab

页面的底部Tab是非常重要的部分， 哪个页面都会用到。

下面就是使用的步骤

1. 在首页中引用底部Tab

```

1. <template>
2.   <div class="background">
3.     <div class="home">
4.       <div class="m_layout">
5.         <!-- 轮播图-->
6.         <HomeBannerView></HomeBannerView>
7.       </div>
8.     </div>
9.     <!-- 这里就是底部Tab -->
10.    <NavBottomView :is_shops_index="is_shops_index"></NavBottomView>
11.  </div>
12. </template>
13. <script>
14.   // 这里就是底部Tab对应的vue文件
15.   import NavBottomView from '../../components/NavBottom.vue';
16. </script>

```

2. 增加对应的component文件

增加 /components/NavBottom.vue :

```

1. <template>
2.   <div class="footer">
3.     <footer class="fixBottomBox">
4.       <ul>
5.         <router-link tag="li" to="/" class="tabItem">
6.           <a href="javascript:;" class="tab-item-link" v-
7. if="is_shops_index">
8.             
9.             <p class="tabbar-text" style="color: rgba(234, 49, 6, 0.66);">
10.            首页</p>

```

```
9.          </a>
10.         <a href="javascript:;" class="tab-item-link" else>
11.             
12.             <p class="tabbar-text">首页</p>
13.         </a>
14.     </router-link>
15.     <router-link tag="li" to="/shops/category" class="tabItem">
16.       <a href="javascript:;" class="tab-item-link" v-
17. if="is_category">
18.           
20.           <p class="tabbar-text" style="color: rgba(234, 49, 6,
21. 0.66);">分类</p>
22.       </a>
23.       <a href="javascript:;" class="tab-item-link" else>
24.           
26.           <p class="tabbar-text">分类</p>
27.       </a>
28.     </router-link>
29.     <router-link tag="li" to="/cart" class="tabItem">
30.       <a href="javascript:;" class="tab-item-link" v-if="is_cart">
31.           
33.           <p class="tabbar-text" style="color: rgba(234, 49, 6,
34. 0.66);">购物车</p>
35.       </a>
36.     </router-link>
37.     <router-link tag="li" to="/mine" class="tabItem">
38.       <a href="javascript:;" class="tab-item-link" v-if="is_mine">
39.           
41.           <p class="tabbar-text" style="color: rgba(234, 49, 6,
42. 0.66);">我的</p>
43.       </a>
```

```
44.          </router-link>
45.      </ul>
46.    </footer>
47.  </div>
48. </template>
49.
50. <script>
51. export default{
52.   data () {
53.     return {
54.       }
55.     },
56.   props: {
57.     is_shops_index: Boolean,
58.     is_category: Boolean,
59.     is_cart: Boolean,
60.     is_mine: Boolean,
61.   },
62.   mounted () {
63.     },
64.   computed: {
65.     },
66.   methods: {
67.     }
68. }
69. </script>
```

效果图

效果如下图所示：



总结

底部Tab 很简单，又很重要。

在本节中，我们使用了一个component来实现它，然后在所有用到它的页面来使用。是一个非常典型的重用过程。

商品列表页

商品列表， 在我们的首页有一部分， 在“列表页”（第二个Tab）也会存在。

所以我们可以直接把抽取成为组件。

下面以首页中引用为例：

1. 在首页中添加代码

```

1. <template>
2.   <div class="background">
3.     <div class="home">
4.       <div class="m_layout">
5.         <div class="product_top">
6.           <div class="product_left">
7.             <div>商品列表</div>
8.           </div>
9.         </div>
10.        <span class="divider" style="height: 2px;"></span>
11.        <!-- 这里循环显示特产商品列表 -->
12.        <SpecialMarket :id="good.id" :name="good.name"
13.          :description="good.description" :image_url="good.image_url" v-for="good in
14.          goods"></SpecialMarket>
15.      </div>
16.    </div>
17.  </template>
18.  <script>
19.    // 在这里引入 特产component
20.    import SpecialMarket from '../../components/SpecialMarket.vue';
21.  </script>

```

上面的核心代码按如下：

```

1. <!-- 这里循环显示特产商品列表 -->
2. <SpecialMarket :id="good.id" :name="good.name" :description="good.description"
3.   :image_url="good.image_url" v-for="good in goods"></SpecialMarket>

```

使用了v-for 和 component的组合，来显示列表。

2. 在component中添加该文件

新增文件 src/components/SpecialMarket.vue:

```
1. <template>
2.   <div>
3.     <div @click="show_goods_details" class="fu_li_zhuan_qu" >
4.       
5.       <div class="content" >
6.         <div class="title">
7.           {{name}}
8.         </div>
9.         <div class="logo_and_shop_name">
10.           <span v-html="description"></span>
11.         </div>
12.       </div>
13.     </div>
14.     <span class="divider" style="height: 2px;"></span>
15.   </div>
16. </template>
17. <script>
18. import { go } from '../libs/router'
19.
20. export default{
21.   data(){
22.     return {
23.       }
24.     },
25.   props: {
26.     id: Number,
27.     name: String,
28.     description: String,
29.     image_url: String,
30.     },
31.   mounted(){
32.     },
33.   methods:{
34.     show_goods_details () {
35.       go("/shops/goods_details?good_id=" + this.id, this.$router)
36.     },
37.   }
38. }
```

```
37.     },
38.     components:{ 
39.       },
40.     }
41. </script>
```

可以看到，该段代码会接受一个数组，然后循环显示。 点击任意一个按钮， 跳转到详情页面。

总结

这里算是最简单的地方了。

商品详情页

当用户在商品列表页中点击时，就会跳转到该页面。

步骤如下：

1. 新增路由

向src/router/index.js中增加：

```
1. import GoodsDetails from '@/views/shops/goods_details'  
2.  
3. Vue.use(Router)  
4.  
5. export default new Router({  
6.   routes: [  
7.     {  
8.       path: '/shops/goods_details',  
9.       name: 'GoodsDetails',  
10.      component: GoodsDetails  
11.    },  
12.  ]  
13. })
```

2. 新增vue页面

向 src/views/shops/goods_details 中增加：

```
1. <template>  
2.   <div class="background">  
3.     <div class="goods_detail" style="height: 100%;"  
4.       <header class="top_bar">  
5.         <a onclick="window.history.go(-1)" class="icon_back"></a>  
6.         <h3 class="cartname">商品详情</h3>  
7.       </header>  
8.       <div class="tast_list_bd" style="padding-top: 44px;">  
9.         <main class="detail_box">  
10.           <!-- 轮播图 -->
```

```
12.      <div class="home_ban">
13.          <div class="m_banner clearfix" id="my_banner">
14.              <ul class="banner_box" >
15.                  <div v-for="image in good_images">
16.                      <li></li>
17.                  </div>
18.              </ul>
19.              <ul class="point_box" >
20.                  <li></li>
21.              </ul>
22.          </div>
23.      </div>
24.
25.      <section class="product_info clearfix">
26.          <div class="product_left">
27.              <p class="p_name">{{good.name}}</p>
28.              <div class="product_pric">
29.                  <span>¥</span>
30.                  <span class="rel_price">{{good.price}}</span>
31.                  <span></span>
32.
33.                  <span style='color: grey;
34.                     text-decoration: line-through;
35.                     font-size: 18px;
36.                     margin-left: 14px;'>
37.                      原价: ¥{{good.original_price}}
38.                  </span>
39.          </div>
40.          <!--
41.          <div class="product_right">
42.              降价通知
43.          </div>
44.          -->
45.      </div>
46.  </section>
47.
48.      <span class="divider" style="height: 2px;"></span>
49.      <div id="choose_number" style="height: 40px; background-color: #fff;">
50.          <label style="font-size: 18px; float: left; margin-left: 10.5px;
51. margin-top: 7.5px;">购买数量</label>
52.          <div style="padding-top: 5px;">
53.              
```

```
        <input pattern="[0-9]*" v-model="buy_count" type="text"
      name="counts" style="width:30px;display: inline;float:right; border: 0.5px solid
53. #e2e2e2;line-height:28px;text-align:center;"/>
          
55.       </div>
56.     </div>
57.
58.     <section class="product_intro">
      <div class="pro_det" v-html="good.description" style='padding: 0
59. 6.5px;'>
60.       </div>
61.     </section>
62.   </main>
63. </div>
64.
65. <footer class="cart_d_footer">
66.   <div class="m">
67.     <ul class="m_box">
68.       <li class="m_item">
69.         <a @click="toCart" class="m_item_link">
70.           <em class="m_item_pic three"></em>
71.           <span class="m_item_name">购物车</span>
72.         </a>
73.       </li>
74.     </ul>
75.     <div class="btn_box clearfix" >
76.       <a @click="addToCart" class="buy_now">加入购物车</a>
77.       <a @click="zhifu" class="buybuy">立即购买</a>
78.     </div>
79.   </div>
80. </footer>
81.
82.   </div>
83. </div>
84. </template>
85. <script>
86. import { go } from '../../../../../libs/router'
87. //import { swiper, swiperSlide } from 'vue-awesome-swiper'
88. import {scrollPic} from '../../../../../libs/index.js'
89.
90.   export default{
91.     data(){
```

```

92.         return {
93.             good_images: [],
94.             good: '',
95.             buy_count: 1,
96.             good_id: this.$route.query.good_id,
97.         }
98.     },
99.     watch:{
100.     },
101.     mounted(){
102.         scrollPic(); //轮播图
103.

        this.$http.get(this.$configs.api + 'goods/goods_details?good_id=' +
104. this.good_id).then((response)=>{
105.         console.info(this.good_id)
106.         console.info(response.body)
107.         this.good = response.body.good
108.         this.good_images = response.body.good_images
109.

110.     },(error) => {
111.         console.error(error)
112.     });
113.     },
114.     methods:{
115.         addToCart () {
116.             alert("商品已经加入到了购物车")
117.             let goods = {
118.                 id: this.good_id,
119.                 title: this.good.name,
120.                 quantity: this.buy_count,
121.                 price: this.good.price,
122.                 image: this.good_images[0]
123.             }
124.             this.$store.dispatch('addToCart', goods)
125.         },
126.         toCart () {
127.             go("/cart", this.$router)
128.         },
129.         plus () {
130.             this.buy_count = this.buy_count + 1
131.         },
132.         minus () {

```

```

133.         if(this.buy_count > 1) {
134.             this.buy_count = this.buy_count - 1
135.         }
136.     },
137.     zhifu () {
138.         go("/shops/dingdanzhifu?good_id=" + this.good_id + "&buy_count=" +
139.         this.buy_count, this.$router)
140.     },
141.     components: {
142.     },
143.     computed: {
144.     }
145.   }
146. </script>

```

在上面的代码中，

1. 实现了加入购物车的方法
2. 实现了对于支付页面的跳转
3. 实现了从远程接口读取数据

3. 添加物品到购物车

下面的代码是把某个商品添加到购物车中：

```

1. addToCart () {
2.   console.info('加入购物车')
3.   alert("商品已经加入了购物车")
4.   let goods = {
5.     id: this.good_id,
6.     title: this.good.name,
7.     quantity: this.buy_count,
8.     price: this.good.price,
9.     image: this.good_images[0]
10.   }
11.   this.$store.dispatch('addToCart', goods)
12. },

```

同时，在 `src/vuex/actions.js` 中，添加如下代码：

```

1. export const addToCart = ({ commit }, product) => {

```

```
2.     commit(types.ADD_TO_CART, {
3.       id: parseInt(product.id),
4.       image: product.image,
5.       title: product.title,
6.       quantity: product.quantity,
7.       price: product.price
8.     })
9.   }
```

看效果



总结

- 购物车使用了Vuex来保存数据。 下一节会详述。
- 进入到支付页面，在后面会详述。 这个页面我们只加上一个链接就好了
- 本页面使用了后台提供的接口，会返回必要的数据。 接口结构略。

购车

购车具备两个功能：

1. 保存用户需要的数据
2. 清空

所以，我们使用vuex来实现购车

1. 添加购车 路由

向 文件 `src/router/index.js` 中增加内容：

```
1. import Cart from '@/components/Cart'  
2.  
3. Vue.use(Router)  
4.  
5. export default new Router({  
6.   routes: [  
7.     { path: '/cart',  
8.       name: 'Cart',  
9.       component: Cart  
10.    },  
11.  ]  
12. })
```

2. 添加查看购物车的vue页面

新增 `src/components/Car.vue` 文件，内容如下：

```
1. <template>  
2.   <div class="background">  
3.     <div id="my_cart">  
4.       <CartHeaderView></CartHeaderView>  
5.       <CartMainView></CartMainView>  
6.       <NavBottomView :is_cart="is_cart"></NavBottomView>  
7.     </div>  
8.   </div>  
9. </template>  
10.
```

```

11. <script>
12.
13. import CartHeaderView from './CartHeader.vue';
14. import CartMainView from './CartMain.vue';
15. import NavBottomView from './NavBottom.vue';
16.
17. export default{
18.     data () {
19.         return {
20.             is_cart: true
21.         }
22.     },
23.     mounted(){
24.     },
25.     components:{
26.         CartHeaderView,
27.         CartMainView,
28.         NavBottomView
29.     }
30. }
31. </script>

```

3. 增加对应的组件

3.1 新增购物车的头部文件 src/components/CartHeader.vue:

```

1. <template>
2.     <div id="carttp">
3.         <header class="top_bar">
4.             <a onclick="window.history.go(-1)" class="icon_back"></a>
5.             <h3 class="cartname">购物车</h3>
6.         </header>
7.     </div>
8. </template>

```

3.2 新增购物车的主体内容 src/components/CartMain.vue:

```

1. <template>
2.     <main class="cart_box">
3.         <p v-show="!products.length"><i>请选择商品加入到购物车</i></p>

```

```
    <div class="cart_content clearfix" v-for="item in products"
4. style="position: relative;">
5.     <div class="cart_shop clearfix">
6.         <div class="cart_check_box">
7.             <div class="check_box" checked>
8.             </div>
9.         </div>
10.        <div class="shop_info clearfix">
11.            <span class="shop_name" style="font-size: 14px;">丝路乐购新疆
12.            商城</span>
13.        </div>
14.
15.        <div @click="find_item_id(item)" class="cart_del clearfix"
16. style="display: inline-block; position: absolute; top: 10px; right: 10px;">
17.            <div class="del_top">
18.            </div>
19.            <div class="del_bottom">
20.            </div>
21.            <div class="cart_item">
22.                <div class="cart_item_box">
23.                    <div class="check_box">
24.                    </div>
25.                </div>
26.                <div class="cart_detial_box clearfix">
27.                    <a class="cart_product_link">
28.                        
29.                    </a>
30.                    <div class="item_names">
31.                        <a>
32.                            <span>{{item.title}}</span>
33.                        </a>
34.                    </div>
35.                    <div class="cart_weight">
36.                        <span class="my_color" style="color: #979292;">
37.                            {{item.title}}</span>
38.                        </div>
39.                        <div class="cart_product_sell">
40.                            <span class="product_money">¥<strong>
```

```
        <span @click="minus(item.id)" class="my_jian">-
41.  </span>
42.  :value="item.quantity">
43.          <span @click="add(item.id)" class="my_add">+</span>
44.      </div>
45.      </div>
46.      </div>
47.      </div>
48.  </div>
49.
50.      <div class="pop" style="display: none">
51.          <div class="pop_box">
52.              <div class="del_info">
53.                  确定要删除该商品吗？
54.              </div>
55.              <div class="del_cancel">
56.                  取消
57.              </div>
58.              <div @click="deleteItem" class="del_ok">
59.                  确定
60.              </div>
61.          </div>
62.      </div>
63.
64.      <div class="cart_fo">
65.          <footer class="cart_footer">
66.              <div class="count_money_box">
67.                  <div class="heji">
68.                      <strong>合计:</strong>
69.                      <strong style="color: #ff621a; font-size: 18px;">\{\ \{ total |
70. currency \}\}</strong>
71.                  </div>
72.                  <a :disabled="!products.length" @click="checkout(products)" class="go_pay">
73.                      <span style="color: #f5f5f5; font-weight: 600;">结算</span>
74.                  </a>
75.              </div>
76.          </div>
77.      </main>
78.  </template>
79.  <script>
```

```
80. import { mapGetters } from 'vuex'
81. import { go } from '../libs/router'
82. import {check, animatDelBox} from '../assets/js/cart.js'
83.
84. export default{
85.     data(){
86.         return{
87.             need_delete_item: {},
88.             cartDatas:[ ],
89.         }
90.     },
91.     mounted(){
92.         check();
93.         animatDelBox();
94.     },
95.     computed: {
96.         ...mapGetters({
97.             products: 'cartProducts',
98.             checkoutStatus: 'checkoutStatus'
99.         }),
100.        total () {
101.            return this.products.reduce((total, item) => {
102.                return total + item.price * item.quantity
103.            }, 0)
104.        }
105.    },
106.    methods: {
107.
108.        // 跳转到支付页面
109.        checkout (products) {
110.            go("/shops/dingdanzhifu", this.$router)
111.        },
112.
113.        // 对于商品的数量进行增加
114.        add (id) {
115.            this.$store.dispatch('changeItemNumber', {id, type: 'add'})
116.        },
117.
118.        // 对于商品的数量进行减少
119.        minus (id) {
120.            this.$store.dispatch('changeItemNumber', {id, type: 'minus'})
121.        },

```

```

122.
123.      // 删除某个商品
124.      deleteItem () {
125.          this.$store.dispatch('deleteItem', this.need_delete_item.id)
126.      },
127.      find_item_id (item) {
128.          this.need_delete_item = item
129.      }
130.  },
131. }
132. </script>

```

3.3 修改Vuex的函数

把下面代码添加到 `src/vuex/actions.js` 文件中：

```

1. export const deleteItem = ({ commit }, id) => {
2.     commit(types.DELETE_ITEM, {
3.         id: parseInt(id)
4.     })
5. }
6.
7. export const changeItemNumber = ({ commit }, {id, type}) => {
8.     console.info(id)
9.     commit(types.CHANGE_ONE_QUANTITY, {
10.         id: parseInt(id),
11.         type
12.     })
13. }

```

上面的代码，实现了购物车的若干功能：

- 对于商品数量的增减
- 实现了当商品数量改变时，商品总价也跟着修改。

看效果

购物车做好后，点击打开，如下图所示：



总结

- 购物车的数据，是通过Vuex来保存的
- 购物车中数量的加减，是需要直接影响到商品总价的。这个角度来看，用Vuex来做更加适合。

微信支付

微信支付，最难的地方不在于技术，而是在于微信有一套自己的技术规范。

建议每位同学都要从官方文档看起。 虽然市面上有一些集成工具，例如 Ping++，但是往往这些产品不是特别方便，收费也比较高昂。 出了问题不好调试。

所以，我们对于核心技术，一定要亲自掌握。

申请微信账号和配置

这里就不详述了。 我们假设全部的账号都已经做好了。

1. 添加支付页面的路由

```
1. import Pay from '@/components/pay'  
2.  
3. Vue.use(Router)  
4.  
5. export default new Router({  
6.   routes: [  
7.     {  
8.       path: '/shops/pay',  
9.       name: 'Pay',  
10.      component: Pay  
11.    },  
12.  ]  
13.  
14. })
```

2. 添加vue页面

```
1. <template>  
2.   <div class="background">  
3.     <header class="top_bar">  
4.       <a onclick="window.history.go(-1)" class="icon_back"></a>  
5.       <h3 class="cartname">订单支付</h3>  
6.     </header>  
7.
```

```
<div class="tast_list_bd" style="background-color: #F3F3F3; padding-top: 0;
8. padding-bottom: 80px;">
9.     <div class="goods_detail" style="">
10.
11.         <main class="detail_box">
12.             <span class="divider"></span>
13.
14.             <form style="margin-top: 45px;">
15.                 <div class="column is-12">
16.                     <label class="label">收货人</label>
17.                     <p class="control has-icon has-icon-right">
18.                         <input name="name" v-model="mobile_user_name" v-
validate="'required|required'" :class="{'input': true, 'is-danger': 
errors.has('name') }" type="text" placeholder="例如：张三"
18. autofocus="autofocus"/>
19.                         <span v-show="errors.has('name')" class="help is-danger">收
19. 货人不能为空</span>
20.                     </p>
21.                 </div>
22.
23.                 <div class="column is-12">
24.                     <label class="label">收货地址</label>
25.                     <p class="control has-icon has-icon-right">
26.                         <input name="url" v-model="mobile_user_address" v-
validate="'required|required'" :class="{'input': true, 'is-danger': 
errors.has('url') }" type="text" placeholder="例如：北京市朝阳区大望路西西里小区4栋2
26. 单元201"/>
27.                         <span v-show="errors.has('url')" class="help is-danger">收
27. 货地址不能为空</span>
28.                     </p>
29.                 </div>
30.
31.                 <div class="column is-12">
32.                     <label class="label">收货电话</label>
33.                     <p class="control has-icon has-icon-right">
34.                         <input name="phone" v-model="mobile_user_phone" v-
validate="'required|numeric'" :class="{'input': true, 'is-danger': 
errors.has('phone') }" type="text" placeholder="例如：18888888888"/>
34.                         <span v-show="errors.has('phone')" class="help is-danger">
35. 电话号码不能为空</span>
36.                     </p>
37.                 </div>
38.             </form>
39.
```

```
40.     <span class="divider"></span>
41.
42.     <section class="product_info clearfix" v-if="single_pay">
43.         <div>
44.             <div class="fu_li_zhuan_qu" >
45.                 
46.                 <div class="content" >
47.                     <div class="title">
48.                         {{good.name}}
49.                     </div>
50.                     <div class="logo_and_shop_name">
51.                         <div class="product_pric">
52.                             <span>¥</span>
53.                             <span class="rel_price">{{good.price}}</span>
54.                             <span> &ampnbspx {{buy_count}}</span>
55.                         </div>
56.                     </div>
57.                 </div>
58.             </div>
59.         </div>
60.     </section>
61.
62.     <section class="product_info clearfix" v-else v-for="product in
63. cartProducts">
64.         <div>
65.             <div class="fu_li_zhuan_qu" >
66.                 
67.                 <div class="content" >
68.                     <div class="title">
69.                         {{product.title}}
70.                     </div>
71.                     <div class="logo_and_shop_name">
72.                         <div class="product_pric">
73.                             <span>¥</span>
74.                             <span class="rel_price">{{product.price}}</span>
75.                             <span> &ampnbspx {{product.quantity}}</span>
76.                         </div>
77.                     </div>
78.                 </div>
79.             </div>
80.         </section>
```

```
81.
82.      <section>
83.          <span class="divider" style="height: 15px;"></span>
84.          <div class="extra_cost" style=" ">
85.              <span style="float: left; margin-left: 15px;"> 卖家留言:</span>
86.              <input v-model="guest_remarks" id="extra_charge" type="text"
     name="cost" placeholder="选填：对本次交易的说明" style="border: 0; background-
86. color: white;
     font-size: 15px; color: #48484b; outline: none; width: 60%;">
87.      </input>
88.      </div>
89.  </section>
90.
91.  <section>
92.      <span class="divider"></span>
93.      <div class="extra_cost" style=" ">
94.          <span style="float: left; margin-left: 15px;"> 应付金额:</span>
95.          <div v-if="single_pay" class="rel_price" type="text" name="cost"
95. style="border: 0; background-color: white;
     font-size: 20px; color: #ff621a; font-weight: bold; outline: none;
96. text-align: right; padding-right: 20px;"> \{\{ total_cost | currency \}}</div>
96.          <div v-else class="rel_price" type="text" name="cost"
97. style="border: 0; background-color: white;
     font-size: 20px; color: #ff621a; font-weight: bold; outline: none;
98. text-align: right; padding-right: 20px;"> \{\{ total | currency \}}</div>
99.
100.         </div>
101.     </section>
102.   </main>
103.
104.   <span class="divider"></span>
105.
105.   <div style="height: 55px; display: flex; width: 100%; padding: 0px
106. 10px; background-color: #fff;" @click="">
107.      <div style="flex: 1; display: flex;">
108.          <div style="margin-top: 10px;">
109.              
110.          </div>
111.          <span style="margin-top: 8px; font-size: 18px; line-height:40px;
111. margin-left: 10px;">微信支付</span>
112.      </div>
113.
114.      <div style=" padding: 14px 10px;" @click="user_wechat">
```

```
115.          
116.      </div>
117.    </div>
118.  </div>
119. </div>
120.
121.  <div class="shop_layout-scroll-absolute" style="">
122.    <div class="queding" @click="buy">
123.      立即支付
124.    </div>
125.  </div>
126.
127. </div>
128. </template>
129. <script>
130.   import { go } from '../../libs/router'
131.   import { mapGetters } from 'vuex'
132.   export default{
133.     data(){
134.       return {
135.         good_images: [],
136.         good: '',
137.         buy_count: this.$route.query.buy_count,
138.         good_id: this.$route.query.good_id,
139.         open_id: this.$store.state.userInfo.open_id,
140.         mobile_user_address: '',
141.         mobile_user_name: '',
142.         mobile_user_phone: '',
143.         guest_remarks: '',
144.         is_use_wechat: false,
145.       }
146.     },
147.     watch:{
148.     },
149.     mounted(){
150.       if (this.single_pay) {
151.         this.$http.get(this.$configs.api + 'goods/goods_details?good_id='
152.           + this.good_id).then((response)=>{
153.             console.info(this.good_id)
154.             console.info(response.body)
155.             this.good = response.body.good
156.             this.good_images = response.body.good_images
157.           })
158.       }
159.     }
160.   }
161. 
```

```
156.          },(error) => {
157.            console.error(error)
158.          });
159.        }
160.      },
161.      computed: {
162.        total () {
163.          return this.cartProducts.reduce((total, p) => {
164.            return (total + p.price * p.quantity)
165.          }, 0)
166.        },
167.        single_pay () {
168.          return this.good_id && this.buy_count
169.        },
170.        total_cost () {
171.          return this.good.price * this.buy_count
172.        },
173.        ...mapGetters({
174.          cartProducts: 'cartProducts',
175.          checkoutStatus: 'checkoutStatus'
176.        })
177.      },
178.      methods:{
179.        validateBeforeSubmit() {
180.          //拦截异步操作
181.          return new Promise((resolve, reject) => {
182.            this.$validator.validateAll().then(result => {
183.              console.info(result)
184.              if (result) {
185.                console.info("=====表单验证成功====")
186.                resolve(true);
187.              } else {
188.                alert('请填写完整的收货信息！');
189.                resolve(false);
190.              }
191.            });
192.          });
193.        },
194.        plus () {
195.          this.buy_count = this.buy_count + 1
196.        },
197.        minus () {
```

```
198.         if(this.buy_count > 1) {
199.             this.buy_count = this.buy_count - 1
200.         }
201.     },
202.     user_wechat () {
203.         if (this.is_use_wechat === false) {
204.             this.is_use_wechat = true
205.         } else {
206.             this.is_use_wechat = false
207.         }
208.     },
209.     buy () {
210.         let result = this.validateBeforeSubmit().then((resolve)=>{
211.             if (resolve) {
212.                 console.info('true ---- ')
213.                 let params
214.                 if (this.single_pay) {
215.                     params = {
216.                         good_id: this.good_id,
217.                         buy_count: this.buy_count,
218.                         total_cost: this.total_cost,
219.                         guest_remarks: this.guest_remarks,
220.                         mobile_user_address: this.mobile_user_address,
221.                         mobile_user_name: this.mobile_user_name,
222.                         mobile_user_phone: this.mobile_user_phone,
223.                         open_id: this.open_id
224.                     }
225.                 } else {
226.                     console.info(this.total)
227.                     params = {
228.                         goods: this.cartProducts,
229.                         total_cost: this.total,
230.                         guest_remarks: this.guest_remarks,
231.                         mobile_user_address: this.mobile_user_address,
232.                         mobile_user_name: this.mobile_user_name,
233.                         mobile_user_phone: this.mobile_user_phone,
234.                         open_id: this.open_id
235.                     }
236.                 }
237.                 this.$http.post(this.$configs.api + 'goods/buy', params
238. ).then((response) => {
239.                 let order_number = response.body.order_number
```

```
240.          this.purchase(order_number)
241.      }, (error) => {
242.          console.error(error)
243.      });
244.  } else {
245.      console.info('==> 请填写完整的收货信息')
246.  }
247. });
248. },
249. purchase (order_number) {
250.     //调起微信支付界面
251.     if (typeof WeixinJSBridge == "undefined"){
252.         if( document.addEventListener ){
253.             document.addEventListener('WeixinJSBridgeReady',
254.             this.onBridgeReady, false);
255.         }else if (document.attachEvent){
256.             document.attachEvent('WeixinJSBridgeReady',
257.             this.onBridgeReady);
258.         }
259.     }else{
260.         this.onBridgeReady(order_number);
261.     },
262.     onBridgeReady (order_number) {
263.         let that = this
264.         let total_cost
265.         if (this.single_pay) {
266.             total_cost = this.total_cost
267.         } else {
268.             total_cost = this.total
269.         }
270.         this.$http.post(this.$configs.api + 'payments/user_pay',
271. {
272.     open_id: this.$store.state.userInfo.open_id,
273.     total_cost: total_cost,
274.     order_number: order_number
275. }).then((response) => {
276.     WeixinJSBridge.invoke(
277.         'getBrandWCPayRequest', {
278.             "appId": response.data.appId,
279.             "timeStamp": response.data.timeStamp,
```

```

280.         "nonceStr": response.data.nonceStr,
281.         "package": response.data.package,
282.         "signType": response.data.signType,
283.         "paySign": response.data.paySign
284.     },
285.     function(res){
286.         // 下面代码仅用于调试
287.         // alert("res.err_msg: " + res.err_msg + ", err_desc: "
288.         + res.err_desc)
289.         在用户支付成功后返回 ok, 但并不保证它绝对可靠。
290.         that.$router.push({ path: '/shops/paysuccess?
291.         order_id=' + order_number });
292.         } else {
293.             // 显示取消支付或者失败
294.             that.$router.push({ path: '/shops/payfail?order_id='
295.             + order_number });
296.         }
297.     }, (error) => {
298.         console.error(error)
299.     });
300.     }
301. },
302. }
303. </script>

```

核心代码如下：

```

1. onBridgeReady (order_number) {
2.     //....
3.     this.$http.post(this.$configs.api + 'payments/user_pay',
4.     {
5.         open_id: this.$store.state.userInfo.open_id,
6.         total_cost: total_cost,
7.         order_number: order_number
8.     }).then((response) => {
9.         WeixinJSBridge.invoke(
10.             'getBrandWCPayRequest', {
11.                 "appId": response.data.appId,
12.                 "timeStamp": response.data.timeStamp,

```

```

13.          //....
14.      },
15.      function(res){
16.          //...
17.      }
18.  );
19. }, (error) => {
20.     console.error(error)
21. });
22. }

```

上面的代码是用来给页面一准备好(WeixinJSBridge 准备好了)的时候，页面就要调用的。

```

1. purchase (order_number) {
2.     //调起微信支付界面
3.     if (typeof WeixinJSBridge == "undefined"){
4.         if( document.addEventListener ){
5.             document.addEventListener('WeixinJSBridgeReady', this.onBridgeReady,
6.             false);
7.         }else if (document.attachEvent){
8.             document.attachEvent('WeixinJSBridgeReady', this.onBridgeReady);
9.             document.attachEvent('onWeixinJSBridgeReady', this.onBridgeReady);
10.        }
11.    }else{
12.        this.onBridgeReady(order_number);
13.    }
14. },

```

上面的代码用于调用出“微信支付页面”。其中的变量 `WeixinJSBridge` 是微信浏览器自带的变量，不必声明，直接拿过来用就行。

看效果

微信的支付页面会跳出来（图略）

总结

可以看到：

1. 微信支付的细节处理，都交给了后台服务器端。只要我们H5端把参数准备好，直接访问 http://shopweb.siwei.me/api/payments/user_pay 就可以了。

2. 微信支付，分成单笔商品支付和多笔商品支付两种情况。区别就是把参数重新组织一下即可。
3. 新手对于 WeixinJSBridge 这个变量很难掌握，一定要多看文档。这个文档是“微信公众号内支付”的文档，不是微信APP，或者微信普通H5的文档。一定要梳理好逻辑。另外，对于后端API的同学这里更难，建议多查多试。
4. 在微信的后台，要配置不同的支付目录。安卓和IOS 的粗略是不一样的。建议大家百度一下。
5. 微信的支付场景对应的支付方式和实现方式是不一样的。本例是“微信的公众号内支付”。

微信的官方文档中，提供的例子都是基于经典的WEB页面（整体刷新的那种）的，目前还没有看到SPA的例子。但是大家的问题很多。我的个人博客也记录了一些内容：

<http://siwei.me/blog/posts/--27>

由于篇幅限制，微信相关的内容不再赘述。

回顾

本章，我们把一个公益扶农的项目从0到1的搭建了起来。

同学们不要惊讶，实际上这是我们公司的一个真实项目。 我们使用Vue.js很快的交付了甲方使用。

使用Vue.js来做开发，有下面这些好处：

- 开发效率更高
- 页面耦合更加松散。 整体结构清晰， 文件组织合理
- 可以很方便的引用现成的组件
- 就算遇到难题，也比使用其他的框架简单不少
- 自带的双向绑定，极大地节省了我们的时间
- 前后端分离的非常彻底，特别适合做微信端，H5端的开发。

注意

1. 为了节约篇幅，本章的代码中，所有的vue页面，都省去了 `<style>` 这些内容。
2. 可以运行的完整源代码，在这里：https://github.com/sg552/happy_book_vuejs

- [vue-devtools](#)
- [如何学习文档](#)
- [查看Vue.js API](#)

安装 Vue.js 的开发工具： vuejs devtool

由于Vue.js是一个框架，它是构建于 javascript 的代码之上，而 javascript 语言的实现并不像其他后端语言（例如 java, python, ruby等）那样有着对debug友好的错误提示机制。

这个可以说原因有：

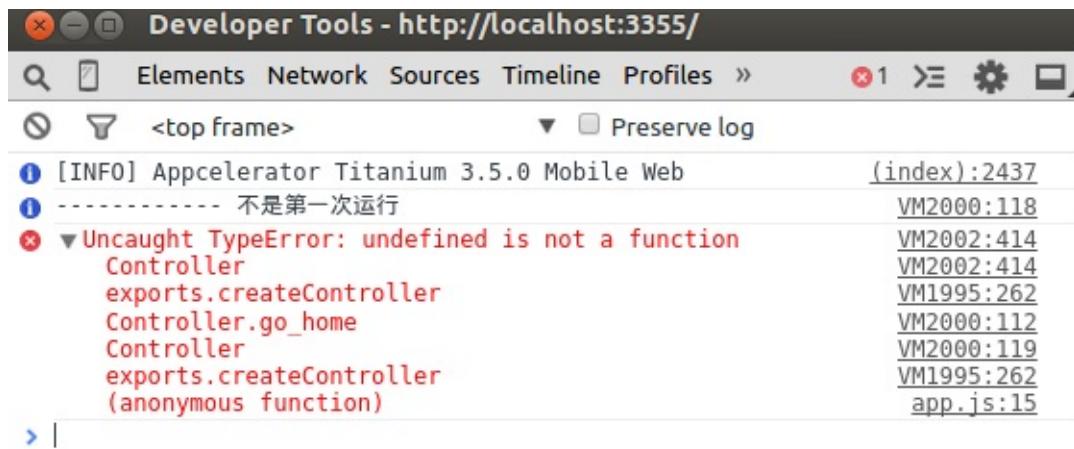
1. javascript语言是一个非常灵活的语言， 支持 “元编程”， 而对于任何一个“框架”来说， 都会大量的用到 “元编程”的能力。 例如：

```
1. var my_code = "var a = 1 + 1; console.info(a) "
2. eval(some_code)
```

在上面的代码中， `eval` 方法就是一个最典型的 “元编程”方法 (meta programming). 说的通俗些， 元编程就是为了“让程序来写程序”。 元编程能力是评估一个语言是否“高级”的一个重要指标。

在带来大量好处的同时， 元编程的缺点是：显示错误信息比较麻烦。 往往显示的错误提示或者stack trace 不是特别明朗。

1. javascript语言的实现机制， 是在各个不同的浏览器中实现的。 不同的浏览器厂家，都会实现不同的javascript虚拟机 (virtual machine)， 所以，我们往往你会发现，一些 javascript的错误往往是不可读的。例如：



上面出错的原因，就是由于， javascript的代码中，可能会有不同的“scope”， 而该浏览器为每个“scope”， 都会分配一个 “virtual machine”， 所以，上面打印出来的 stack trace ， 很忠实地反应出了问题所在， 就是几乎没有可读性。

所以，为了方便我们的开发， 各位同学一定要安装对应的开发组件， 例如：“Vue.js

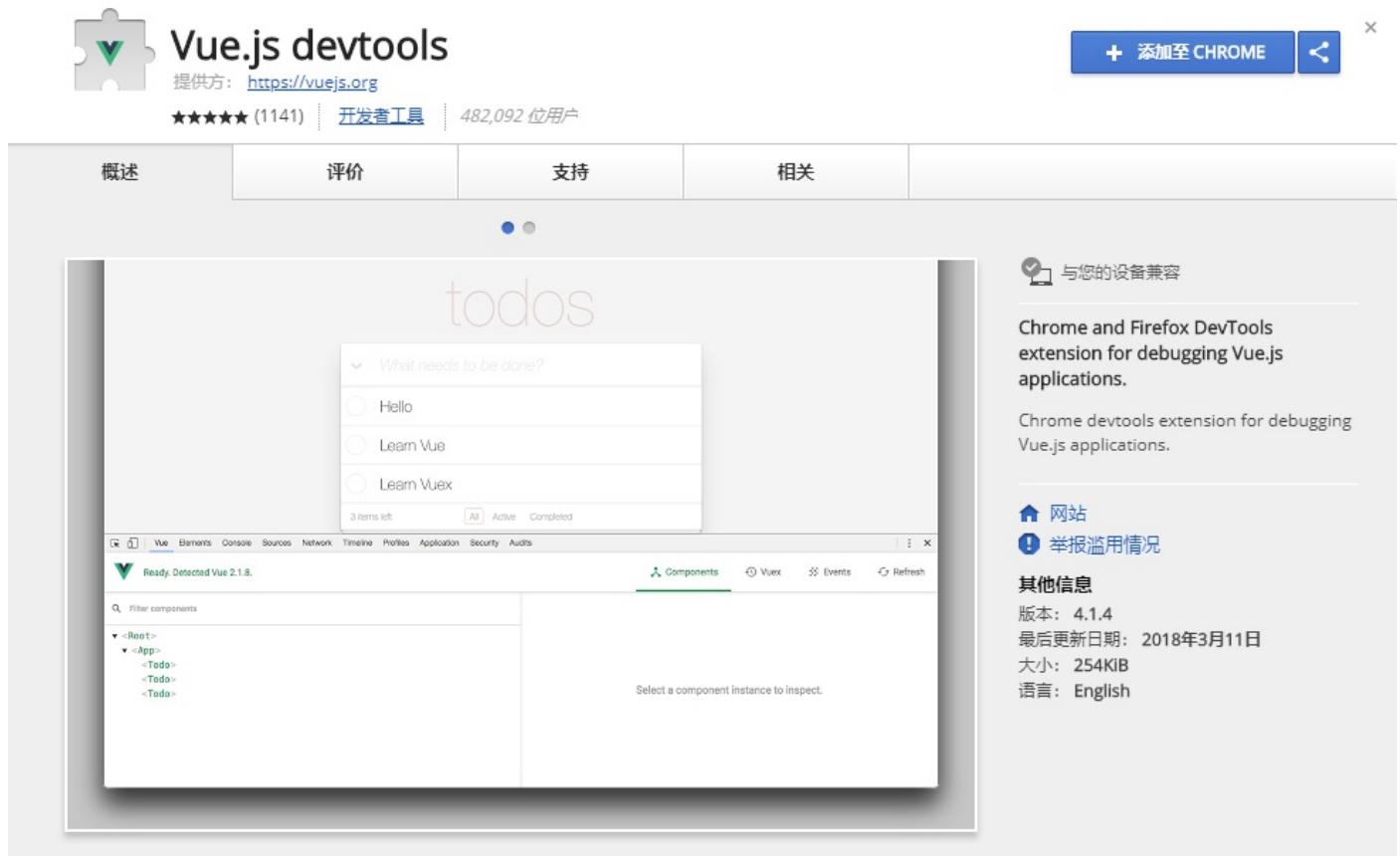
devtools”.

官方网址: <https://github.com/vuejs/vue-devtools>

安装步骤

非常简单。建议大家使用chrome，这样就可以以插件的形式来安装了。

1. 安装chrome
2. 使用科学上网，打开: <https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd>
3. 就会看到下图:



1. 点击后，会询问是否安装，我们点击“添加扩展程序”这个按钮就可以了：

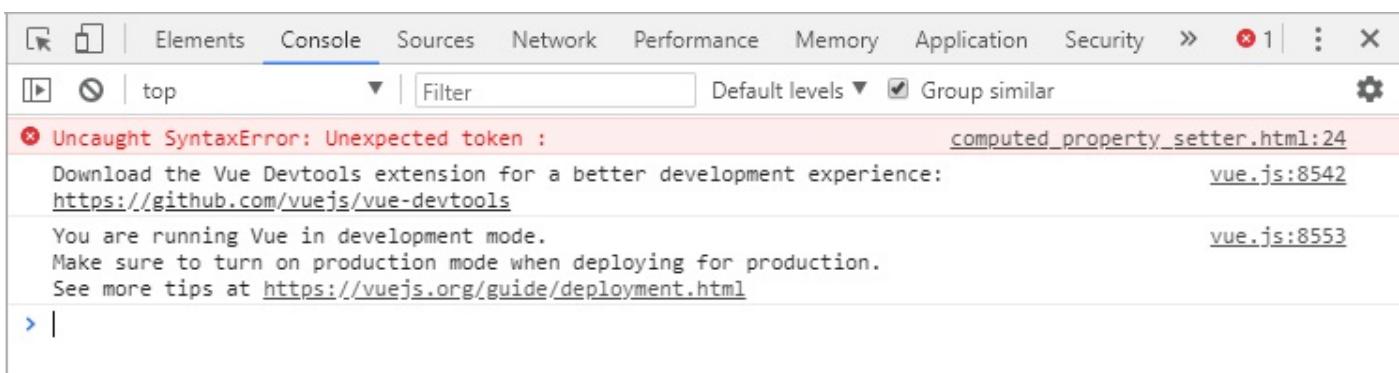


1. 就可以看到，浏览器的右上角新增加了灰色的图标。 安装成功。
2. 打开 “设置” -> ‘更多工具’ -> ‘扩展程序’，就可以看到刚才安装的Vuejs了，勾选下面的“允许访问文件网址”。如下图所示：

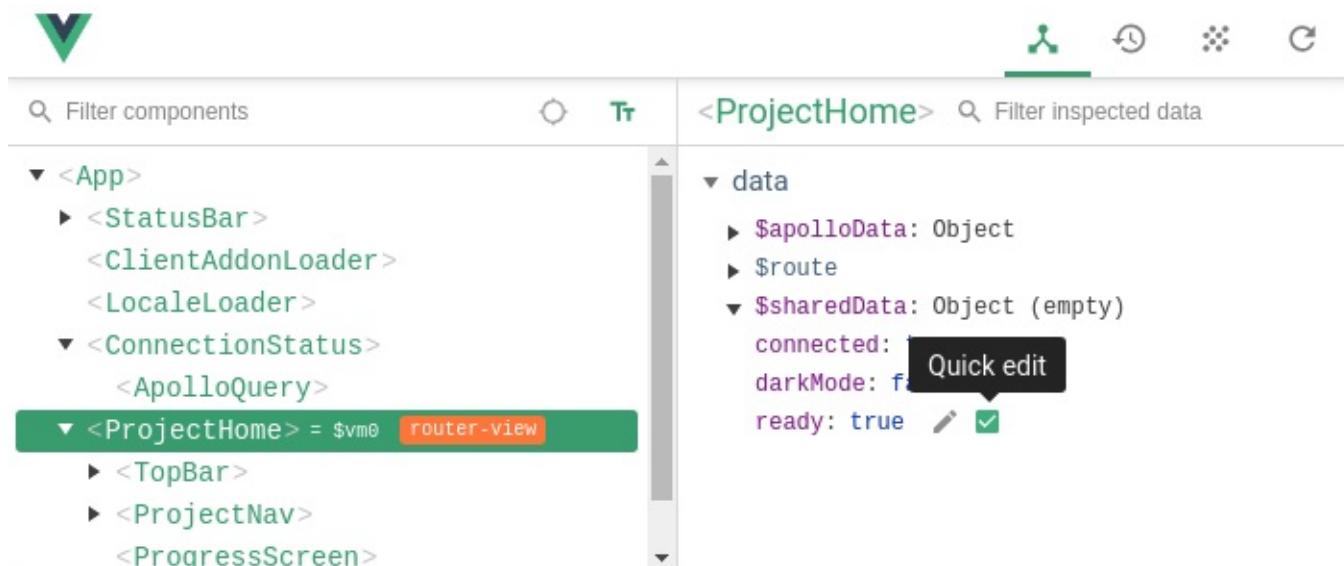


使用步骤

在安装 devtool 之前，我们调试的时候，都是打开浏览器的“开发者工具”， 然后看到的类似下图：



安装好之后，如果你的vuejs 项目是使用 `http` 服务器打开的话，（不是 `file:///...`）就可以看到了：



如何阅读官方文档

我们在查看vue.js的文档的时候，会发现它跟我们真正使用的项目的代码完全不一样。例如，vuejs的官方文档的讲解，都是这样：

（完全是把所有代码都写在了js中）

```

1. var Child = {
2.   template: `
3.   A custom component!
4.   `
5. }
6. new Vue({
7.   // ...
8.   components: {
9.     // 将只在父模板可用
10.    'my-component': Child
11.  }
12. })

```

而我们的实际代码是这样：

```

1. <template>
2. ....
3. </template>
4. <script>
5. ....
6. </script>
7. <style>
8. </style>

```

原因就在于我们在实际项目中使用了 `vue-loader`，它可以非常好的帮我们自动加载所有的内容，按照webpack + vue的约定。

webpack

webpack就是一种工具，可以把各种js/css/html代码最后打包编译到一起。

Vuejs中已经集成了这个工具，我们在使用vue-cli的时候，就会根据命令来生成 `webpack` 所要求的文件结构，然后在打包的时候(`npm run build`)，vuejs的源代码就会被webpack打包成正

常的html代码和文件目录。

这个内容我们以后会讲到。大家要知道在本书中所讲的知识，都是“基于webpack的vuejs”。否则光看vuejs的官方文档，是看不懂的。

官方文档地址

<https://cn.vuejs.org/> (点击右上角的 translation就可以看到入口)

如何查看API文档

Vuejs的API的查看方式，对于初学者需要适应。

想要读懂API的话，首先需要对Vuejs的各个部分有个明确的认识。

英文版：<https://vuejs.org/v2/api/>

中文版：<https://cn.vuejs.org/v2/api/>

推荐大家在查看文档的时候，查看英文版。很多专业的名词，例如：‘minxin’，‘component’都是原汁原味的。读起来更加明白一些。

建议每一位同学都要从头到尾看一下这里。大约1个小时就可以全部看完，之后就会对Vuejs有个非常全面的了解~