

第一章 Vue.js 概要介绍



1.1 Vue.js 介绍

- Vue 是什么
 - 主流的渐进式 JavaScript 框架
- 什么是渐进式
 - 可以和传统的网站开发架构融合在一起，例如可以简单的把它当作一个类似 JQuery 库来使用。
 - 也可以使用Vue全家桶框架来开发大型的单页面应用程序。
- 使用它的原因
 - vue.js 体积小，编码简洁优雅，运行效率高，用户体验好。
 - 无Dom操作，它能提高网站应用程序的开发效率
- 什么场景下使用它
 - 一般是需要开发单页面应用程序（Single Page Application, 简称:SPA）的时候去用
 - 单页面应用程序，如：网易云音乐 <https://music.163.com/>
 - 因为 Vue 是 渐进式 的，Vue 其实可以融入到不同的项目中，即插即用

1.2 学习资源

英文官网：<https://vuejs.org/>

中文官网（中文文档很友好）：<https://cn.vuejs.org/>

官方教程：<https://cn.vuejs.org/v2/guide/>

GitHub：<https://github.com/yyx990803>

API文档：<https://cn.vuejs.org/v2/api/>

不建议买书，官方文档很详细，多查官方文档，因为很多书基本上都是直接抄官方文档的

1.3 发展历史

- 作者：尤雨溪（微博：尤小右），一位华裔前 Google 工程师，江苏无锡人。
 - 个人博客：<http://www.evanyou.me/>
 - 新浪微博：<http://weibo.com/arttechdesign>
 - 知乎：<https://www.zhihu.com/people/evanyou/activities>
- 2013年12月8号在 GitHub 上发布了 0.6 版
- 2015年10月份正式发布了 1.0 版本，开始真正的火起来
- 2016年10月份正式发布了 2.0 版
- 2019.4.8号发布了 Vue 2.5.10 版本 <https://github.com/vuejs/vue/releases>
- 1.x 版本老项目可能还在用，新项目绝对都是选择 2.x

1.4 对比其他前端 JS 框架

- Angular
 - 2009 年诞生的，起源于个人开发，后来被 Google 收购了。
 - 核心技术：模板 和 数据绑定 技术，体验越来越差，走下坡路了。
- React
 - 2013年5月开源的，起源于 Facebook 的内部项目，对市场上所有 JS 框架都不满意，于是自己写了一套。
 - 核心技术：组件化 和 虚拟DOM 技术。
- Vue.js
 - 吸收了上面两个框架的技术优点。

使用情况：

- BAT 级别的企业：React 最多 > Angular > Vue.js
- 中小型公司：Vue.js 更多一些，有中文文档学习成本低。

Vue **不支持** IE8 及以下版本，因为 Vue 使用了 IE8 无法模拟 ECMAScript 5 特性。

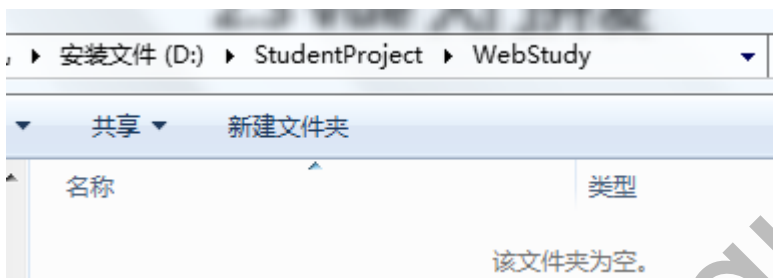
推荐使用最新谷歌浏览器。

第二章 Vue 核心技术

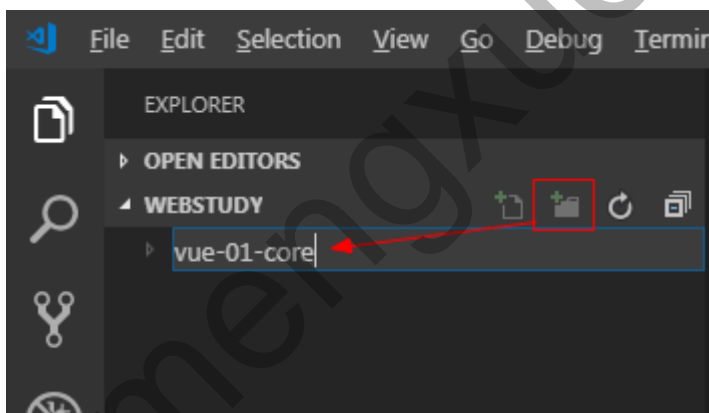
2.1 Vue 入门开发

2.1.1 创建工程

- 在本地创建文件夹 `D:\StudentProject\WebStudy`



- 打开 VS Code, 点击 `File > Open Folder`, 找到 `D:\StudentProject\WebStudy` 打开
- 单击 WEBSTUDY 右侧的新建目录图标, 创建目录: `vue-01-core`



2.1.2 创建 HTML 和 安装 vue 模块

- 在 `vue-01-core` 目录下新建一个页面 `01-helloworld.html`
- 在 `vue-01-core` 目录下的命令行窗口, 安装2.6.10版本的 vue 模块

```
1 npm install vue@2.6.10
```

2.1.3 编写HTML页面

- 编写步骤:
 - 采用 `<script>` 标签引入 `vue.js` 库
 - 定义一个 `<div>`
 - `new Vue()` 实例化Vue应用程序

`el` 选项：元素element的缩写，指定被 Vue 管理的 Dom 节点入口（值为选择器），必须是一个普通的 HTML 标签节点，一般是 `div`。

`data` 选项：指定初始化数据，在 Vue 所管理的 Dom 节点下，可通过模板语法来进行使用

4. 标签体显示数据：`{{xxxxx}}`

5. 表单元素双向数据绑定：`v-model`

6. 注意：`el` 的值不能为 `html` 或 `body`

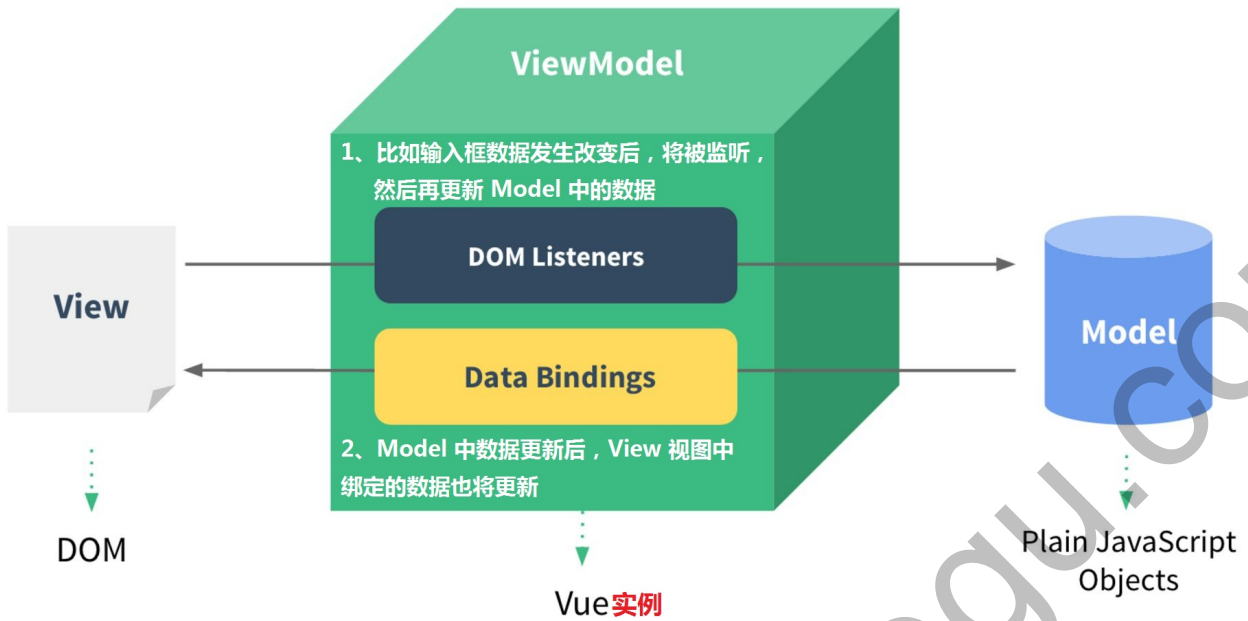
- 源码实现：

```
1 <body>
2   <div id="app">
3     <p>Hello, {{ msg }}</p>
4     <input type="text" v-model="msg">
5   </div>
6   <script src="/node_modules/vue/dist/vue.js"></script>
7   <script type="text/javascript">
8     var vm = new Vue({
9       el: '#app', // el选项的值不能指定html或 body
10      data: {
11        msg: 'Vue.js'
12      }
13    })
14  </script>
15 </body>
```

2.2 分析 MVVM 模型

常见面试题：什么是 MVVM 模型？

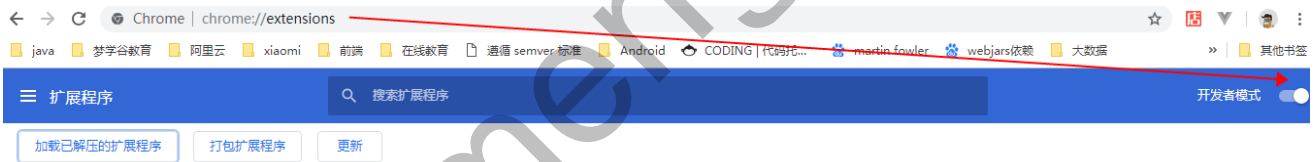
- `MVVM` 是 Model-View-ViewModel 的缩写，它是一种软件架构风格
 - Model：模型，数据对象（`data`选项当中的）
 - View：视图，模板页面（用于渲染数据）
 - ViewModel：视图模型，其实本质上就是 Vue 实例
 - 它的哲学思想是：
 - 通过数据驱动视图
 - 把需要改变视图的数据初始化到 Vue 中，然后再通过修改 Vue 中的数据，从而实现对视图的更新。
 - 声明式编程
 - 按照 Vue 的特定语法进行声明开发，就可以实现对应功能，不需要我们直接操作 Dom 元素
- 命令式编程：jQuery 它就是，需要手动去操作 Dom 才能实现对应功能。



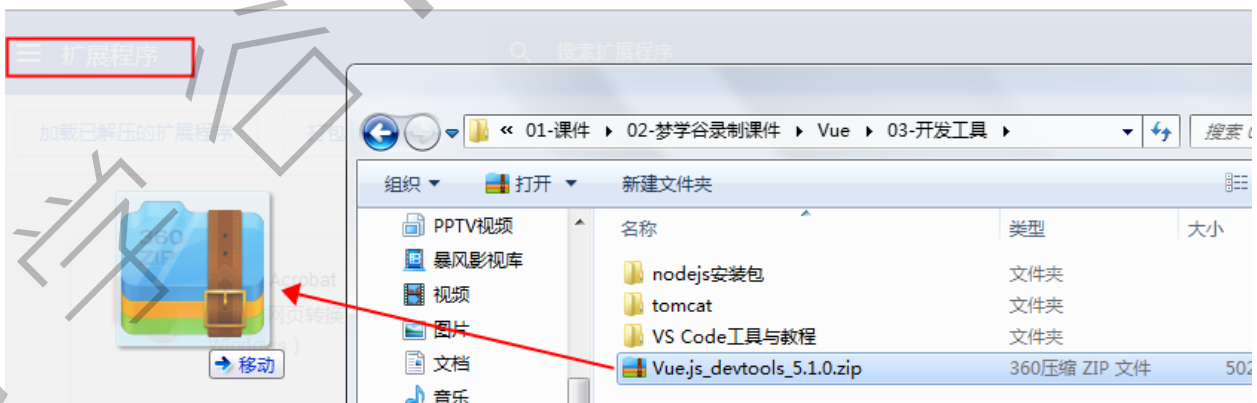
2.3 Vue Devtools 插件安装

Vue Devtools 插件让我们在一个更友好的界面中审查和调试 Vue 项目。

- 谷歌浏览器访问：`chrome://extensions`，然后右上角打开 开发者模式，打开的效果如下，



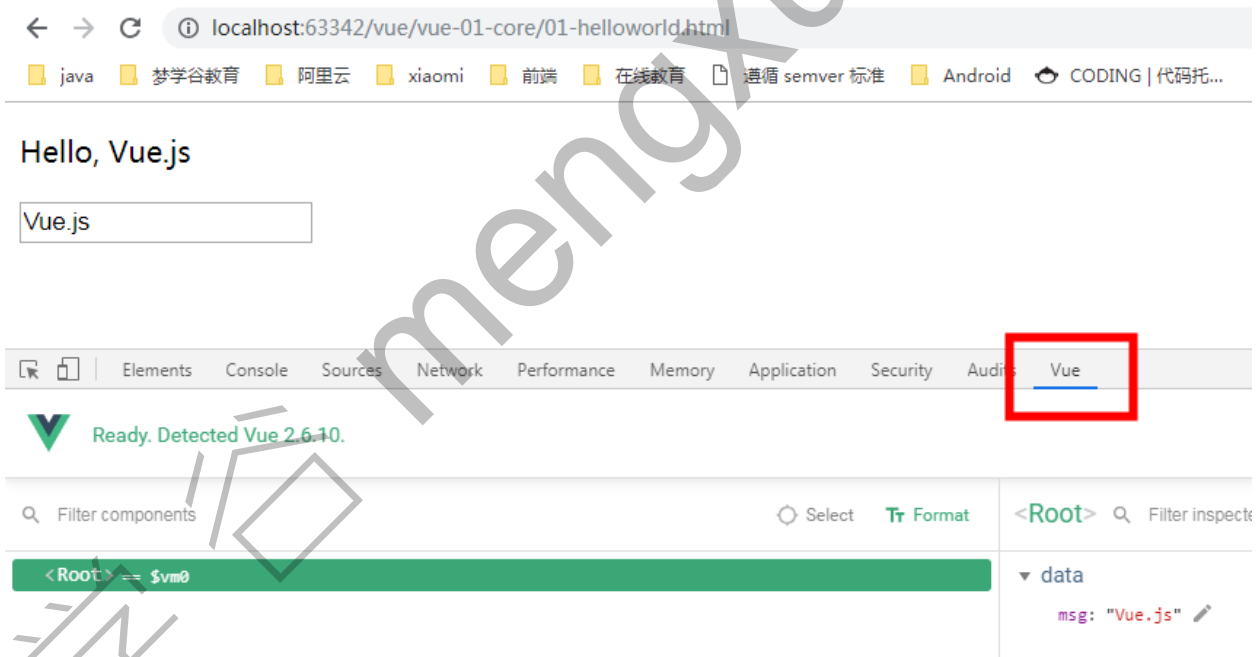
- 将 直接拖到上面页面空白处，会自动安装



- 效果如下则安装成功



- 当你访问Vue开发的页面时，按 F12 可 Vue 标签页



2.4 模板数据绑定渲染

可生成动态的HTML页面，页面中使用嵌入 Vue.js 语法可动态生成

1. `{{xxxx}}` 双大括号文本绑定
2. `v-xxxx` 以 `v-` 开头用于标签属性绑定，称为指令

在 vue-01-core 目录下新建一个页面：`02-模板数据绑定渲染.html`

2.4.1 双大括号语法 {{}}

- 格式：{{表达式}}
- 作用：
 - 使用在标签体中，用于获取数据
 - 可以使用 JavaScript 表达式
- 案例源码：

```
1 <body>
2   <div id="app">
3     <h3>1、双大括号输出文本内容</h3>
4     <!--文本内容-->
5     <p>普通文本：{{ message }}</p>
6     <!--使用JS表达式-->
7     <p>JS表达式：{{ number + 1 }}</p>
8   </div>
9   <script src="./node_modules/vue/dist/vue.js"></script>
10  <script type="text/javascript">
11    var vm = new Vue({
12      el: '#app',
13      data: {
14        message: 'haha',
15        number: 1
16      }
17    })
18  </script>
19 </body>
```

2.4.2 一次性插值 v-once

- 通过使用 `v-once` 指令，你也能执行一次性地插值，当数据改变时，插值处的内容不会更新。

```
1 <h3>2、一次性插值 v-once </h3>
2 <span v-once> 这个将不会改变: {{ message }}</span>
```

2.4.3 输出HTML指令 v-html

- 格式：v-html='xxx'
- 作用：
 - 如果是HTML格式数据，双大括号会将数据解释为普通文本，为了输出真正的 HTML，你需要使用 `v-html` 指令。
 - Vue 为了防止 XSS 攻击，在此指令上做了安全处理，当发现输出内容有 `script` 标签时，则不渲染

- XSS 攻击主要利用 JS 脚本注入到网页中，读取 Cookie 值（Cookie 一般存储了登录身份信息），读取到了发送到黑客服务器，从而黑客可以使用你的帐户做非法操作。
- XSS 攻击还可以在你进入到支付时，跳转到钓鱼网站。

• 案例源码：

```
1 <body>
2   <div id="app">
3     <h3>3、v-html 指令输出真正的 HTML 内容</h3>
4     <p>双大括号：{{ contentHtml }}</p>
5     <!-- 指令的值不需要使用双大括号获取，直接写获取的属性名 -->
6 <!--   <p>v-html指令：<span v-html="{{contentHtml}}"></span></p>-->
7     <p>v-html指令：<span v-html="contentHtml"></span></p>
8   </div>
9   <script src="./node_modules/vue/dist/vue.js"></script>
10  <script type="text/javascript">
11    var vm = new Vue({
12      el: '#app',
13      data: {
14        message: 'haha',
15        number: 1,
16        //contentHtml: '<span style="color:red">红色字体内容</span>'
17        contentHtml: '<span style="color:red">红色字体内容<>script>alert("hello vue")</Vscript></span>'
18      }
19    })
20  </script>
21 </body>
```

• 效果图

3、v-html 指令输出真正的 HTML 内容

双大括号：红色字体内容

v-html指令：红色字体内容

2.4.4 元素绑定指令 v-bind

- 完整格式：v-bind:元素的属性名='xxxx'
- 缩写格式：:元素的属性名='xxxx'
- 作用：将数据动态绑定到指定的元素上
- 案例源码：

```
1 <body>
2   <div id="app">
3     <h3>4、v-bind 属性绑定指令</h3>
4     <!-- 红色字体是正常的 -->
5     
6     <!-- 缩写 -->
```



```
7     
8   </div>
9
10  <script src="./node_modules/vue/dist/vue.js"></script>
11  <script type="text/javascript">
12    var vm = new Vue({
13      el: '#app',
14      data: {
15        message: 'haha',
16        number: 1,
17        contentHtml: '<span style="color:red">红色字体内容</span>',
18        imgUrl: 'https://cn.vuejs.org/images/logo.png'
19      }
20    })
21  </script>
22 </body>
```

- 效果图

4、v-bind 属性绑定指令



VueLogo

2.4.5 事件绑定指令 v-on

- 完整格式: v-on:事件名称="事件处理函数名"
- 缩写格式: @事件名称="事件处理函数名" 注意: @ 后面没有冒号
- 作用: 用于监听 DOM 事件
- 案例源码: 每点击1次, 数据就加1

```
1 <body>
2   <div id="app">
3     <h3>5、v-on 事件绑定指令</h3>
4     <input type="text" v-model="num">
5     <button v-on:click="add">点击+1</button>
6   </div>
```

```
7
8 <script src="./node_modules/vue/dist/vue.js"></script>
9 <script type="text/javascript">
10   var vm = new Vue({
11     el: '#app',
12     data: {
13       message: 'haha',
14       number: 1,
15       contentHtml: '<span style="color:red">红色字体内容</span>',
16       imgUrl: 'https://cn.vuejs.org/images/logo.png',
17       num: 2
18     },
19     methods: { //指定事件处理方法, 在模板页面中通过 v-on:事件名 来调用
20       add: function () { //key为方法名
21         console.log('add被调用')
22         // this 表示当前 vm 实例
23         this.num++ //每点击1次num加1
24       }
25     }
26   })
27 </script>
28 </body>
```

- 效果图

5、v-on 事件绑定指令

2

2.4.6 完整源码

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Vue-模板数据绑定渲染</title>
6 </head>
7 <body>
8   <div id="app">
9     <h3>1、双大括号输出文本内容</h3>
10    <!-- 文本内容 -->
11    <p>普通文本：{{ message }}</p>
12    <!-- 使用JS表达式 -->
13    <p>JS表达式：{{ number + 1 }}</p>
14
15    <h3>2、一次性插值 v-once </h3>
16    <span v-once> 这个将不会改变: {{ message }}</span>
17
18    <h3>3、v-html 指令输出真正的 HTML 内容</h3>
19    <p>双大括号：{{ contentHtml }}</p>
20    <!-- 指令的值不需要使用双大括号获取 -->
21    <!-- <p>v-html指令：<span v-html="{{contentHtml}}"></span></p> -->
22    <p>v-html指令：<span v-html="contentHtml"></span></p>
```

```
23
24     <h3>4、v-bind 属性绑定指令</h3>
25     <!-- 直接写属性名是获取不到-->
26     
27     <!-- 红色字体是正常的 -->
28     
29     <!-- 缩写 -->
30     
31
32     <h3>5、v-on 事件绑定指令</h3>
33     <input type="text" v-model="num">
34     <button v-on:click="add">点击+1</button>
35 </div>
36 <br>
37
38 <script src="./node_modules/vue/dist/vue.js"></script>
39 <script type="text/javascript">
40     var vm = new Vue({
41       el: '#app',
42       data: {
43         message: 'haha',
44         number: 1,
45         contentHtml: '<span style="color:red">红色字体内容</span>',
46         imgUrl: 'https://cn.vuejs.org/images/logo.png',
47         num: 2
48       },
49       methods: { //指定事件处理方法, 在模板页面中通过 v-on:事件名 来调用
50         add: function () { //key为方法名
51           console.log('add被调用')
52           vm.num++ //每点击1次num加1
53         }
54       }
55     })
56 </script>
57 </body>
58 </html>
```

2.5 计算属性和监听器

在 `vue-01-core` 目录下新建一个页面 `03-计算属性和监听器.html`

2.5.1 计算属性 `computed`

- `computed` 选项定义计算属性
- 计算属性 类似于 `methods` 选项中定义的函数
 - 计算属性 会进行缓存，只在相关响应式依赖发生改变时它们才会重新求值。
 - 函数 每次都会执行函数体进行计算。

- 需求：输入数学与英语分数，采用 `methods` 与 `computed` 分别计算出总得分
- 案例源码：

```
1 <body>
2   <div id="app">
3     数学：<input type="text" v-model="mathScore" >
4     英语：<input type="text" v-model="englishScore">
5     总分(方法-单向)：<input type="text" v-model="sumScore()">
6     总分(计算属性-单向)：<input type="text" v-model="sumScore1">
7   </div>
8   <script src="./node_modules/vue/dist/vue.js"></script>
9   <script type="text/javascript">
10    var vm = new Vue({
11      el: '#app',
12      data: {
13        mathScore: 80,
14        englishScore: 90,
15      },
16      methods: { //不要少了s
17        sumScore: function () {
18          //在控制台输入 vm.sumScore() 每次都会被调用
19          console.info('sumScore被调用')
20          // `this` 指向当前 vm 实例，减 0 是为了字符串转为数字运算
21          return (this.mathScore 0) + (this.englishScore 0)
22        }
23      },
24      computed: { //计算属性
25        sumScore1: function () {
26          //在控制台输入 vm.sumScore1 不会被重新调用,说明计算属性有缓存
27          console.info('sumScore1被调用')
28          return (this.mathScore - 0) + (this.englishScore - 0)
29        }
30      }
31    })
32   </script>
33 </body>
```

`computed` 选项内的计算属性默认是 `getter` 函数，所以上面只支持单向绑定，当修改数学和英语的数据才会更新总分，而修改总分不会更新数据和英语

2.5.2 计算属性（双向绑定）

- 计算属性默认只有 `getter`，不过在需要时你也可以提供一个 `setter`
- 案例源码：

```
1 <body>
2   <div id="app">
3     数学：<input type="text" v-model="mathScore" ><br>
4     英语：<input type="text" v-model="englishScore"><br>
```

```
5      总分(方法-单向) : <input type="text" v-model="sumScore()"><br>
6      总分(计算属性-单向) : <input type="text" v-model="sumScore1"><br>
7      总分(计算属性-双向) : <input type="text" v-model="sumScore2"><br>
8  </div>
9
10     <script src="./node_modules/vue/dist/vue.js"></script>
11     <script type="text/javascript">
12         var vm = new Vue({
13             el: '#app',
14             data: {
15                 mathScore: 80,
16                 englishScore: 90,
17             },
18             methods: { //不要少了s
19                 sumScore: function () {
20                     //在控制台输入 vm.sumScore() 每次都会被调用
21                     console.info('sumScore被调用')
22                     // `this` 指向当前 vm 实例，减 0 是为了字符串转为数字运算
23                     return (this.mathScore - 0) + (this.englishScore - 0)
24                 }
25             },
26             computed: {
27                 //计算属性 默认 getter 只支持单向绑定
28                 sumScore1 : function () {
29                     //在控制台输入vm.sumScore1 不会被重新调用,说明计算属性有缓存
30                     console.info('sumScore1被调用')
31                     return (this.mathScore - 0) + (this.englishScore - 0)
32                 },
33                 //指定 getter/setter 双向绑定
34                 sumScore2 : {
35                     get: function () {
36                         console.info('sumScore2被调用')
37                         return (this.mathScore - 0) + (this.englishScore - 0)
38                     },
39                     set: function (newValue) { //value为更新后的值
40                         // 被调用则更新了sumScore2，然后将数学和英语更新为平均分
41                         var avgScore = newValue / 2
42                         this.mathScore = avgScore
43                         this.englishScore = avgScore
44                     }
45                 }
46             }
47         })
48     </script>
49 </body>
```

2.5.3 监听器 watch

- 当属性数据发生变化时,对应属性的回调函数会自动调用, 在函数内部进行计算
- 通过 `watch` 选项 或者 `vm 实例的 $watch()` 来监听指定的属性

- 需求：
 - 通过 `watch` 选项 监听数学分数，当数学更新后回调函数中重新计算总分`sumScore3`
 - 通过 `vm.$watch()` 选项 监听英语分数，当英语更新后回调函数中重新计算总分`sumScore3`
- 源码实现：

注意: 在`data` 选择中添加一个 `sumScore3` 属性

```
1 <body>
2   <div id="app">
3     数学 : <input type="text" v-model="mathScore" ><br>
4     英语 : <input type="text" v-model="englishScore"><br>
5     总分(方法-单向) : <input type="text" v-model="sumScore()"><br>
6     总分(计算属性-单向) : <input type="text" v-model="sumScore1"><br>
7     总分(计算属性-双向) : <input type="text" v-model="sumScore2"><br>
8     总分(监听器) : <input type="text" v-model="sumScore3"><br>
9   </div>
10
11 <script src="./node_modules/vue/dist/vue.js"></script>
12 <script type="text/javascript">
13   var vm = new Vue({
14     el: '#app',
15     data: {
16       mathScore: 80,
17       englishScore: 90,
18       sumScore3: 170
19     },
20     methods: { //不要少了s
21       sumScore: function () {
22         //在控制台输入 vm.sumScore() 每次都会被调用
23         console.log('sumScore被调用')
24         // `this` 指向当前 vm 实例，减 0 是为了字符串转为数字运算
25         return (this.mathScore - 0) + (this.englishScore - 0)
26       }
27     },
28     // 计算属性
29     computed: {
30       // 默认 getter 只支持单向绑定
31       sumScore1: function () {
32         //在控制台输入 vm.sumScore1 不会被重新调用,说明计算属性有缓存
33         console.log('sumScore1被调用')
34         return (this.mathScore - 0) + (this.englishScore - 0)
35       },
36       //指定 getter/setter 双向绑定
37       sumScore2: {
38         get: function () {
39           console.log('sumScore2被调用')
40           return (this.mathScore - 0) + (this.englishScore - 0)
41         },
42         set: function (newValue) { //value为更新后的值
43           // 被调用则更新了sumScore2，然后将数学和英语更新为平均分
44           var avgScore = newValue / 2
45           this.mathScore = avgScore
```

```
46         this.englishScore = avgScore
47     }
48 }
49 },
50 //监听器方式1：watch选项
51 watch:{
52     //当数学修改后，更新总分sumScore3
53     mathScore: function (newValue, oldValue) {
54         //newValue 就是新输入的数学得分
55         this.sumScore3 = (newValue 0) + (this.englishScore 0)
56     }
57 }
58 })
59 //监听器方式2：通过vm对象调用
60 //第1个参数为监听的属性名，第2个回调函数
61 vm.$watch('englishScore', function (newValue) {
62     //newValue 就是新输入的英语得分
63     this.sumScore3 = (newValue 0) + (this.mathScore 0)
64 })
65 </script>
66 </body>
```

2.6 Class 与 Style 绑定 v-bind

通过 `class` 列表和 `style` 指定样式是数据绑定的一个常见需求。它们都是元素的属性，都用 `v-bind` 处理，其中表达式结果的类型可以是：字符串、对象或数组。

2.6.1 语法格式

- `v-bind:class='表达式'` 或 `:class='表达式'`
 - `class` 的表达式可以为：
 - 字符串 `:class="activeClass"`
 - 对象 `:class="{active: isActive, error: hasError}"`
 - 数组 `:class="['active', 'error']"` 注意要加上单引号,不然是获取data中的值

`v-bind:style='表达式'` 或 `:style='表达式'`

- `style` 的表达式一般为对象

`:style="{color: activeColor, fontSize: fontSize + 'px'}"`

注意：对象中的value值 `activeColor` 和 `fontSize` 是data中的属性

2.6.2 案例源码

在 `vue-01-core` 目录下新建一个页面 `04-Class与Style绑定.html`

- 效果图

Class绑定, v-bind:class 或 :class

字符串表达式

对象表达式

数组表达式

Style绑定, v-bind:style 或 :class

Style绑定

- 源码实现 :

```
1 <body>
2   <!-- 第2步:定义样式 -->
3   <style>
4     .active {
5       color: green;
6     }
7     .delete {
8       background: red;
9     }
10    .error {
11      font-size: 30px;
12    }
13  </style>
14  <div id="app">
15    <h2>Class绑定, v-bind:class 或 :class</h2>
16    <!-- activeClass会从data中获取值为active,则对应样式为绿色-->
17    <p v-bind:class="activeClass">字符串表达式</p>
18
19    <!-- isDelete为 true, 渲染delete样式; 当 hasError为false, 不取 error 样式-->
20    <p :class="{delete: isDelete, error: hasError}">对象表达式</p>
21
22    <!-- 渲染 'active', 'error' 样式,注意要加上单引号,不然是获取data中的值 -->
23    <p :class="['active', 'error']">数组表达式</p>
24
25    <h2>Style绑定, v-bind:style 或 :class</h2>
26    <p :style="{color: activeColor, fontSize: fontSize + 'px'}">Style绑定</p>
27
28  </div>
29
30  <script src="./node_modules/vue/dist/vue.js"></script>
31  <script type="text/javascript">
32    new Vue({
```



```
33     el: '#app',
34     data: {
35       activeClass: 'active',
36       isDelete: true,
37       hasError: false,
38       //演示 Style 绑定
39       activeColor: 'red',
40       fontSize: 20
41     }
42   })
43 </script>
44 </body>
```

2.7 条件渲染 v-if

2.7.1 条件指令

- `v-if` 是否渲染当前元素
- `v-else`
- `v-else-if`
- `v-show` 与 `v-if` 类似, 只是元素始终会被渲染并保留在 DOM 中, 只是简单切换元素的 CSS 属性 `display` 来显示或隐藏

2.7.2 案例源码

在 `vue-01-core` 目录下新建一个页面 `05-条件渲染.html`

- 效果图:

v-if 条件渲染

☒ 勾选后显示红色小块



v-show 条件渲染



- 源码实现

```
1 <body>
2   <style>
3     .box {
4       width: 200px;
5       height: 200px;
6       background: red;
7     }
8   </style>
9
10  <div id="app">
11    <h2>v-if 条件渲染</h2>
12    <input v-model="seen" type="checkbox" >勾选后显示红色小块
13    <!-- v-if 为 true则显示渲染当前元素, -->
14    <div v-if="seen" class="box" ></div>
15    <p v-else="seen">红块已隐藏</p>
16
17    <h2>v-show 条件渲染</h2>
18    <!-- v-show 的元素始终会被渲染并保留在 DOM 中,
```

```
19 只是简单切换元素的 CSS 属性 display 显示隐藏,而不是重新加载div-->
20 <div v-show="seen" class="box" ></div>
21
22 </div>
23 <script src="./node_modules/vue/dist/vue.js"></script>
24 <script type="text/javascript">
25   var vm = new Vue({
26     el: '#app',
27     data: {
28       seen: false
29     }
30   })
31 </script>
32 </body>
```

2.7.3 v-if 与 v-show 比较

1. 什么时候元素被渲染

v-if 如果在初始条件为假，则什么也不做，每当条件为真时，都会重新渲染条件元素

v-show 不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换

2. 使用场景选择

v-if 有更高的切换开销，

v-show 有更高的初始渲染开销。

因此，如果需要非常频繁地切换，则使用 **v-show** 较好；如果在运行后条件很少改变，则使用 **v-if** 较好。

2.8 列表渲染 **v-for**

2.8.1 列表渲染指令

1. v-for 迭代数组

○ 语法：**v-for="(alias, index) in array"**

- 说明：**alias**：数组元素迭代的别名；**index**：数组索引值从0开始(可选)
- 举例：

```
1 <div v-for="item in items" :key="item.id"></div>
2 <div v-for="(item, index) in items" :key="item.id"></div>
```

1 ``items`` 是源数据数组，``item`` 是数组元素迭代的别名。

2

3 ==注意：使用 ``key`` 特殊属性，它会基于 `key` 的变化重新排列元素顺序，并且会移除 `key` 不存在的元素==

2. v-for 迭代对象的属性

- 语法：`v-for="(value, key, index) in Object"`
- 说明：`value`：每个对象的属性值；`key`：属性名(可选)；`index`：索引值(可选)。
- 举例：

```
1 <div v-for="value in object" ></div>
2 <div v-for="(value, key) in object"></div>
3 <div v-for="(value, key, index) in object"></div>
```

注意：在遍历对象时，是按 `Object.keys()` 的结果遍历，但不能保证它的结果在不同的 JavaScript 引擎下是顺序一致的。

3. 可用 of 替代 in 作为分隔符

2.8.2 案例源码

在 `vue-01-core` 目录下新建一个页面 `06-列表渲染.html`

- 效果图

1. 迭代数组

- 编号：1，姓名：马云，工资：20000
- 编号：2，姓名：马化腾，工资：18000
- 编号：3，姓名：刘强东，工资：13000

2. 迭代对象

- 第1个属性为：`name = 马云`
- 第2个属性为：`salary = 20000`

- 源码实现

```
1 <body>
2   <div id="app">
3     <h2>1. 迭代数组</h2>
4     <ul>
5       <!-- e 为当前对象别名，index 数组下标0开始-->
6       <li v-for="(e, index) in emps" :key="index">
7         编号：{{index+1}}，姓名：{{e.name}}，工资：{{e.salary}}
8       </li>
```

```
9      </ul>
10     <br>
11     <h2>2. 迭代对象</h2>
12     <ul>
13       <!-- value是属性值，key是属性名，index索引值-->
14       <li v-for="(value, key, index) in emps[0]">
15         第{{index+1}}个属性为：{{key}} = {{value}}
16       </li>
17     </ul>
18   </div>
19   <script src="./node_modules/vue/dist/vue.js"></script>
20   <script type="text/javascript">
21     var vm = new Vue({
22       el: '#app',
23       data: {
24         emps: [ //数组
25           {name: '马云', salary: '20000'},
26           {name: '马化腾', salary: '18000'},
27           {name: '刘强东', salary: '13000'}
28         ]
29       }
30     })
31   </script>
32 </body>
```

2.9 事件处理 v-on

在 `vue-01-core` 目录下新建一个页面 `07-事件处理.html`

2.9.1 事件处理方法

- 完整格式：`v-on:事件名=函数名` 或 `v-on:事件名=函数名(参数.....)`
- 缩写格式：`@事件名=函数名` 或 `@事件名=函数名(参数.....)` 注意：`@` 后面没有冒号
- `event`：函数中的默认形参，代表原生 DOM 事件
 - 当调用的函数，有多个参数传入时，需要使用原生DOM事件，则通过 `$event` 作为实参传入
- 作用：用于监听 DOM 事件
- 案例源码：

```
1 <body>
2   <div id="app">
3     <h2>1. 事件处理方法</h2>
4     <button @click="say">Say {{msg}}</button>
5     <button @click="warn('hello', $event)">Warn</button>
6   </div>
7   <script src="./node_modules/vue/dist/vue.js"></script>
8   <script type="text/javascript">
```

```
9     var vm = new Vue({
10       el: '#app',
11       data: {
12         msg: 'Hello, Vue.js'
13       },
14       methods: {
15         say: function (event) {
16           // `this` 在方法里指向当前 Vue 实例
17           alert(this.msg)
18           // `event` 是原生 DOM 事件
19           alert(event.target.innerHTML)
20         },
21         // 多个参数如果需要使用原生事件，将 $event 作为实参传入
22         warn: function (msg, event) {
23           alert(msg + " " + event.target.tagName)
24         }
25       }
26     })
27   </script>
28 </body>
```

2.9.2 事件修饰符

- `.stop` 阻止单击事件继续传播 `event.stopPropagation()`
- `.prevent` 阻止事件默认行为 `event.preventDefault()`
- `.once` 点击事件将只会触发一次

```
1 <body>
2   <div id="app">
3     <h2>1. 事件处理方法</h2>
4     <button @click="say">Say {{msg}}</button>
5     <button @click="warn('hello', $event)">Warn</button>
6     <br>
7
8     <h2>2. 事件修饰符</h2>
9     <!-- 单击事件继续传播 -->
10    <div @click="todo">
11      <!-- 点击后会调用doThis再调用todo -->
12      <button @click="doThis">单击事件会继续传播</button>
13    </div>
14    <br>
15    <!-- 阻止单击事件继续传播， -->
16    <div @click="todo">
17      <!-- 点击后只调用doThis -->
18      <button @click.stop="doThis">阻止单击事件会继续传播</button>
19    </div>
20
21    <!-- 阻止事件默认行为 -->
22    <a href="http://www.mengxuegu.com" @click.prevent="doStop">梦学谷官网</a>
```

```
23
24   <!-- 点击事件将只会触发一次 -->
25   <button @click.once="doOnly">点击事件将只会触发一次: {{num}}</button>
26 </div>
27
28 <script src="../node_modules/vue/dist/vue.js"></script>
29 <script type="text/javascript">
30   var vm = new Vue({
31     el: '#app',
32     data: {
33       msg: 'Hello, Vue.js',
34       num: 1
35     },
36     methods: {
37       say: function (event) {
38         // `this` 在方法里指向当前 Vue 实例
39         alert(this.msg)
40         // `event` 是原生 DOM 事件
41         alert(event.target.innerHTML)
42       },
43       // 多个参数如果需要使用原生事件，将 $event 作为实参传入
44       warn: function (msg, event) {
45         alert(msg + " " + event.target.tagName)
46       },
47       todo: function () {
48         alert("todo....");
49       },
50       doThis: function () {
51         alert("doThis....");
52       },
53       doStop: function () {
54         alert("href默认跳转被阻止....")
55       },
56       doOnly: function() {
57         this.num++
58       }
59     }
60   })
61 </script>
62 </body>
```

2.9.3 按键修饰符

- 格式：v-on:keyup.按键名 或 @keyup.按键名
- 常用按键名：
 - .enter
 - .tab
 - .delete (捕获“删除”和“退格”键)
 - .esc
 - .space

- o .up
- o .down
- o .left
- o .right

- 源码实现：

```
1 <body>
2   <div id="app">
3     <h2>1. 事件处理方法</h2>
4     <button @click="say">Say {{msg}}</button>
5     <button @click="warn('hello', $event)">Warn</button>
6     <br>
7
8     <h2>2. 事件修饰符</h2>
9     <!--单击事件继续传播-->
10    <div @click="todo">
11      <!--点击后会调用doThis再调用todo-->
12      <button @click="doThis">单击事件会继续传播</button>
13    </div>
14    <br>
15    <!-- 阻止单击事件继续传播， -->
16    <div @click="todo">
17      <!--点击后只调用doThis-->
18      <button @click.stop="doThis">阻止单击事件会继续传播</button>
19    </div>
20    <br>
21    <!-- 阻止事件默认行为 -->
22    <a href="http://www.mengxuegu.com" @click.prevent="doStop">梦学谷官网</a>
23    <br><br>
24    <!-- 点击事件将只会触发一次 -->
25    <button @click.once="doOnly">点击事件将只会触发一次: {{num}}</button>
26
27    <h2>3. 按键修饰符</h2>
28    <input @keyup.enter="keyEnter"><!--进入输入框按回车时调用keyEnter-->
29    <input @keyup.space="keySpace"><!--进入输入框按回车时调用keySpace-->
30
31  </div>
32
33  <script src="/node_modules/vue/dist/vue.js"></script>
34  <script type="text/javascript">
35    var vm = new Vue({
36      el: '#app',
37      data: {
38        msg: 'Hello, Vue.js',
39        num: 1
40      },
41      methods: {
42        say: function (event) {
43          // `this` 在方法里指向当前 Vue 实例
44          alert(this.msg)
45          // `event` 是原生 DOM 事件
46          alert(event.target.innerHTML)
```



```
47     },
48     //多个参数如果需要使用原生事件，将 $event 作为实参传入
49     warn: function (msg, event) {
50         alert(msg + "," + event.target.tagName)
51     },
52     todo: function () {
53         alert("todo...");
54     },
55     doThis: function () {
56         alert("doThis...");
57     },
58     doStop: function () {
59         alert("href默认跳转被阻止....")
60     },
61     doOnly: function() {
62         this.num++
63     },
64     keyEnter: function () {
65         alert("已按：回车键")
66     },
67     keySpace: function () {
68         alert("已按：空格键")
69     }
70 }
71 })
72 </script>
73 </body>
```

2.10 表单数据双向绑定v-model

- 单向绑定：数据变，视图变；视图变（浏览器控制台上更新html），数据不变；上面的都是单向绑定
- 双向绑定：数据变，视图变；视图变（在输入框更新），数据变；

2.10.1 基础用法

`v-model` 指令用于表单数据双向绑定，针对以下类型：

- `text` 文本
- `textarea` 多行文本
- `radio` 单选按钮
- `checkbox` 复选框
- `select` 下拉框

2.10.2 案例源码

在 `vue-01-core` 目录下新建一个页面 `08-表单数据双向绑定.html`

- 模板页面

```
1 <body>
2   <div id="demo">
3     <form action="#">
4       姓名(文本) : <input type="text" >
5       <br><br>
6
7       性别(单选按钮) :
8         <input name="sex" type="radio" value="1"/>男
9         <input name="sex" type="radio" value="0"/>女
10      <br><br>
11
12      技能(多选框) :
13        <input type="checkbox" name="skills" value="java">Java开发
14        <input type="checkbox" name="skills" value="vue">Vue.js开发
15        <input type="checkbox" name="skills" value="python">Python开发
16      <br><br>
17
18      城市(下拉框) :
19      <select name="citys">
20        <option value="bj">北京</option>
21      </select>
22      <br><br>
23
24      说明(多行文本) : <textarea cols="30" rows="5"></textarea>
25      <br><br>
26      <button type="submit" >提交</button>
27    </form>
28  </div>
29 </body>
```

- Vue.js源码实现

```
1 <body>
2   <div id="demo">
3     <!-- @submit.prevent 阻止事件默认提交行为 -->
4     <form action="#" @submit.prevent="submitForm">
5       姓名(文本) : <input type="text" v-model="name">
6       <br><br>
7
8       性别(单选按钮) :
9         <input name="sex" type="radio" value="1" v-model="sex"/>男
10        <input name="sex" type="radio" value="0" v-model="sex"/>女
11      <br><br>
12
13      技能(多选框) :
14        <input type="checkbox" name="skills" value="java" v-model="skills">Java开发
15        <input type="checkbox" name="skills" value="vue" v-model="skills">Vue.js开发
16        <input type="checkbox" name="skills" value="python" v-model="skills">Python开发
17      <br><br>
18
```

```
19 城市(下拉框):
20 <select name="citys" v-model="city">
21   <option v-for="c in citys" :value="c.code">
22     {{c.name}}
23   </option>
24 </select>
25 <br><br>
26
27 说明(多行文本): <textarea cols="30" rows="5" v-model="desc"></textarea>
28 <br><br>
29 <button type="submit">提交</button>
30 </form>
31 </div>
32 <script src="./node_modules/vue/dist/vue.js"></script>
33 <script type="text/javascript">
34   var vm = new Vue({
35     el: '#demo',
36     data: {
37       name: '',
38       sex: '0', //默认选中: 女
39       skills: ['vue'], //默认勾选: vue.js开发
40       citys: [ //初始化下拉框
41         {code: 'bj', name: '北京'},
42         {code: 'sh', name: '上海'},
43         {code: 'sz', name: '深圳'}
44       ],
45       city: 'sh', //默认选中: 上海,
46       desc: ''
47     },
48     methods: {
49       submitForm: function () {
50         //发送ajax请求
51         alert(this.name+","+this.sex+","+this.skills+","+this.city+","+this.desc)
52       }
53     }
54   })
55 </script>
56 </body>
```

第三章 Vue 过渡&动画和自定义指令

3.1 过渡&动画效果

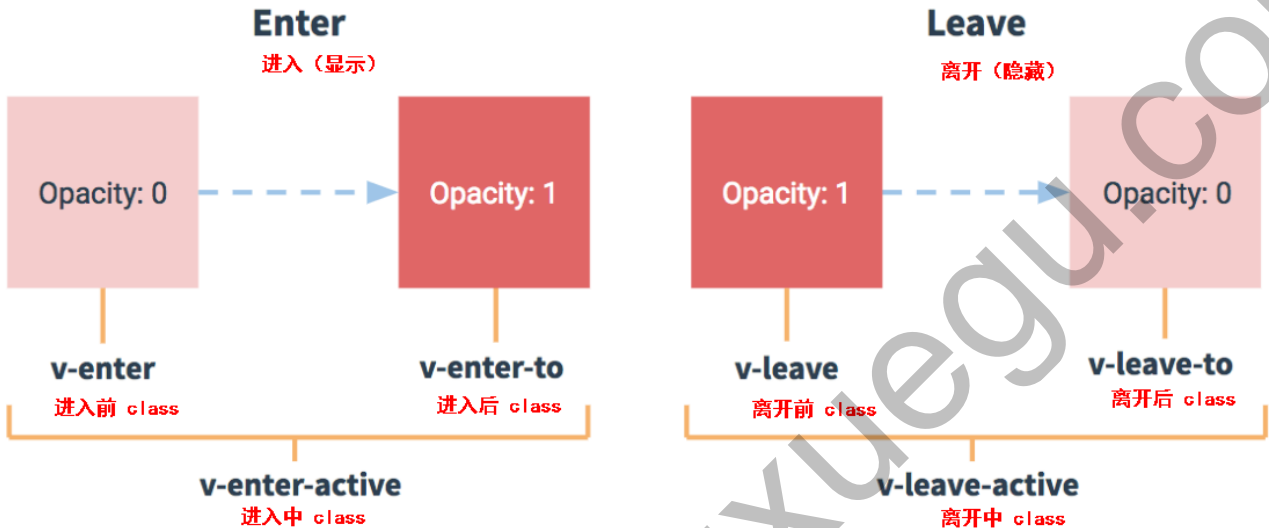
3.1.1 什么是过渡&动画

元素在显示和隐藏时，实现过滤或者动画的效果。常用的过渡和动画都是使用 CSS 来实现的

- 在 CSS 中操作 `transition`（过滤）或 `animation`（动画）达到不同效果
- 为目标元素添加一个父元素 `<transition name='xxx'>`，让父元素通过自动应用 `class` 类名来达到效果

- 过渡与动画时，会为对应元素动态添加的相关 class 类名：

1. xxx-enter：定义显示前的效果。
2. xxx-enter-active：定义显示过程的效果。
3. xxx-enter-to：定义显示后的效果。
4. xxx-leave：定义隐藏前的效果。
5. xxx-leave-active：定义隐藏过程的效果。
6. xxx-leave-to：定义隐藏后的效果。



3.1.2 过滤效果案例

1. 为目标元素添加父元素 `<transition name="xxx">`
2. 定义 class 过渡样式
 - 指定过渡样式: `transition`
 - 指定隐藏时的样式: `opacity(持续的时间)/其它`
3. 功能实现
 - 点击按钮后, 显示隐藏文本
 - 效果1：显示和隐藏有渐变效果
- 效果2：显示和隐藏的有平移效果，并持续时长不同

渐变过渡

效果1：显示和隐藏有渐变效果

渐变平滑过渡

效果2：显示和隐藏的有平移效果，并持续时长不同

- 在 `vue-02-过渡&动画和指令` 目录下创建 `01-过渡效果.html`
- 进入 `vue-02-过渡&动画和指令` 目录，执行 `npm install vue@2.6.10` 命令安装 vue 模块

```
1 <head>
2   <meta charset="UTF-8">
3   <title>过渡效果</title>
4   <style>
5     /*显示或隐藏的过渡效果*/
6     .mxg-enter-active, .mxg-leave-active {
7       transition: opacity 1s; /*过渡，渐变效果 1秒*/
8     }
9     /*显示前或隐藏后的效果*/
10    .mxg-enter, .mxg-leave-to {
11      opacity: 0 /*都是隐藏效果*/
12    }
13
14    /* 可以设置不同的进入和离开动画 */
15    /*显示过渡效果*/
16    .meng-enter-active{
17      transition: all 1s; /*all 所有效果，持续1秒 */
18    }
19    /*隐藏过渡效果*/
20    .meng-leave-active {
21      transition: all 5s;
22    }
23    /*显示前或隐藏后的效果*/
24    .meng-enter, .meng-leave-to {
25      opacity: 0; /*都是隐藏效果*/
26      transform: translateX(10px); /*水平方向 X 坐标移动10px*/
27    }
28  </style>
29 </head>
30 <body>
31   <div id="app1">
32     <button @click="show = !show">渐变过渡</button>
33     <!--在目标元素上添加此元素，结合name值来指定样式-->
34     <transition name="mxg">
35       <p v-if="show">mengxuegu</p>
36     </transition>
37   </div>
38   <br>
39   <!--可以设置不同的进入和离开动画 -->
40   <div id="app2">
41     <button @click="show = !show">渐变平滑过渡</button>
42     <!--在目标元素上添加此元素，结合name值来指定样式-->
43     <transition name="meng">
44       <p v-if="show">mengxuegu</p>
45     </transition>
46   </div>
47
48   <script src="./node_modules/vue/dist/vue.js"></script>
49   <script type="text/javascript">
50     var vm1 = new Vue({
51       el: '#app1',
52       data: {
53         show: true
```

```
54     }  
55   })  
56   var vm2 = new Vue({  
57     el: '#app2',  
58     data: {  
59       show: true  
60     }  
61   })  
62 </script>  
63 </body>
```

3.1.3 动画效果案例

- CSS 动画用法同 CSS 过渡，只不过采用 `animation` 为指定动画效果
- 功能实现：
 - 点击按钮后，文本内容有放大缩小效果
 - 在 `vue-02-过渡&动画和指令` 目录下创建 `02-动画效果.html`

注意：官网上面源码有问题，要在 `<p>` 元素上增加样式 `style="display: inline-block"`

```
1 <head>  
2   <meta charset="UTF-8">  
3   <title>动画效果</title>  
4   <style>  
5     /*显示过程中的动画效果*/  
6     .bounce-enter-active {  
7       animation: bounce-in 1s; /*bounce-in引用了下面@keyframes中定义的持续3秒*/  
8     }  
9     /*隐藏过程中的动画效果*/  
10    .bounce-leave-active {  
11      animation: bounce-in 3s reverse; /*reverse 相反的顺序*/  
12    }  
13    @keyframes bounce-in {  
14      0% { /*持续时长的百分比，如持续1s，0%表示当0秒，50%表示当0.5秒，100%表示当1秒*/  
15        transform: scale(0); /*缩小为0*/  
16      }  
17      50% {  
18        transform: scale(1.5); /*放大1.5倍*/  
19      }  
20      100% {  
21        transform: scale(1); /*原始大小*/  
22      }  
23    }  
24  </style>  
25 </head>  
26 <body>  
27   <div id="example-2">  
28     <button @click="show = !show">放大缩小动画</button>  
29     <br>  
30     <transition name="bounce">
```

```
31     <p v-if="show" style="display: inline-block">
32         陪你学习，伴你成长
33     </p>
34 </transition>
35 </div>
36
37 <script src="./node_modules/vue/dist/vue.js"></script>
38 <script type="text/javascript">
39     new Vue({
40         el: '#example-2',
41         data: {
42             show: true
43         }
44     })
45 </script>
46 </body>
```

3.2 Vue 内置指令总结

参考：<https://cn.vuejs.org/v2/api/#指令>

- [v-html](#) 内容按普通 HTML 插入，可防止 XSS 攻击
- [v-show](#) 根据表达式的真假值，切换元素的 `display` CSS 属性来显示隐藏元素
- [v-if](#) 根据表达式的真假值，来渲染元素
- [v-else](#) 前面必须有 `v-if` 或 `v-else-if`
- [v-else-if](#) 前面必须有 `v-if` 或 `v-else-if`
- [v-for](#) 遍历的数组或对象
- [v-on](#) 绑定事件监听器
- [v-bind](#) 用于绑定元素属性
- [v-model](#) 在表单控件或者组件上创建双向绑定
- [v-once](#) 一次性插值，当后面数据更新后视图数据不会更新
- [v-pre](#) 可以用来显示原始插入值标签 `{{}}`。并跳过这个元素和它的子元素的编译过程。加快编译。

例如：网页中的一篇文章，文章内容不需要被 Vue 管理渲染，则可以在此元素上添加 `v-pre` 忽略文章编译提高性能。

在 `vue-02-过渡&动画和指令` 目录下创建页面：`03-Vue内置指令.html`

```
1 <span v-pre>{{ this will not be compiled }}</span>
2
3 浏览页面显示内容：并没有识别{{
4 {{ this will not be compiled }}
```

- [v-text](#)
 - 等价于 `{{}}` 用于显示内容，但区别在于：

- `{{}}` 会造成闪烁问题，`v-text` 不会闪烁
 - 如果还想用 `{{}}` 又不想有闪烁问题，则使用 `v-cloak` 来处理

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>指令</title>
6   <style>
7     /*将带有 v-cloak 属性的标签隐藏*/
8     [v-cloak]{
9       display: none;
10    }
11  </style>
12 </head>
13 <body>
14   <!-- 在被 Vue 管理的模板入口节点上作用 v-cloak 指令-->
15   <div id="app" v-cloak>
16     <!--
17      用QQ浏览器刷新页面时, {{}} 会有明显闪烁现象,
18      原因:
19      是浏览器从上往下依次解析, 会先把 {{ message }} 当作标签体直接先渲染,
20      然后 Vue 再进行解析 {{ message }} 变成了 message 的值: 'hello mengxuegu'
21      -->
22     <h3>{{ message }}</h3>
23     <h3>{{ message }}</h3>
24     <h3 v-text="message"></h3>
25   </div>
26
27   <script src="./node_modules/vue/dist/vue.js"></script>
28   <script type="text/javascript">
29     new Vue({
30       el: '#app',
31       data: {
32         message: 'hello mengxuegu'
33       }
34     })
35   </script>
36 </body>
37 </html>
```

• [v-cloak](#)

- 如果想用 `{{}}` 又不想有闪烁问题，则使用 `v-cloak` 来处理，步骤如下：
 1. 在被 Vue 管理的模板入口节点上作用 `v-cloak` 指令
 1. 添加一个属性选择器 `[v-cloak]` 的CSS 隐藏样式：`[v-cloak] {display: none;}`
- 原理：默认一开始被 Vue 管理的模板是隐藏着的，当 Vue 解析处理完 DOM 模板之后，会自动把这个样式去除，然后就显示出来。

3.3 自定义指令

3.3.1 自定义指令的作用

除了内置指令外，Vue 也允许注册自定义指令。有的情况下，你仍然需要对普通 DOM 元素进行底层操作，这时候使用自定义指令更为方便。

自定义指令文档：<https://cn.vuejs.org/v2/guide/custom-directive.html>

3.3.2 注册与使用自定义指令方式

1. 注册全局指令：

```
1 // 指令名不要带 v-  
2 Vue.directive('指令名', {  
3   // el 代表使用了此指令的那个 DOM 元素  
4   // binding 可获取使用了此指令的绑定值 等  
5   inserted: function (el, binding) {  
6     // 逻辑代码  
7   }  
8 })
```

2. 注册局部指令

```
1 directives: {  
2   '指令名': { // 指令名不要带 v-  
3     inserted (el, binding) {  
4       // 逻辑代码  
5     }  
6   }  
7 }
```

注意：注册时，指令名不要带 v-

3. 使用指令：

- 引用指令时，指令名前面加上 `v-`
- 直接在元素上使用即可：`v-指令名='表达式'`

3.3.3 案例演示

- 需求：

1. 实现输出文本内容全部自动转为大写，字体为红色（功能类型于 `v-text`，但显示内容为大写）
2. 当页面加载时，该元素将获得焦点（注意：`autofocus` 在移动版 Safari 上不工作）

MENGXUEGU 陪你学习伴你梦想

自动获取焦点：

- 实现：在 `vue-02-过渡&动画和指令` 目录下创建页面：`04-自定义指令.html`

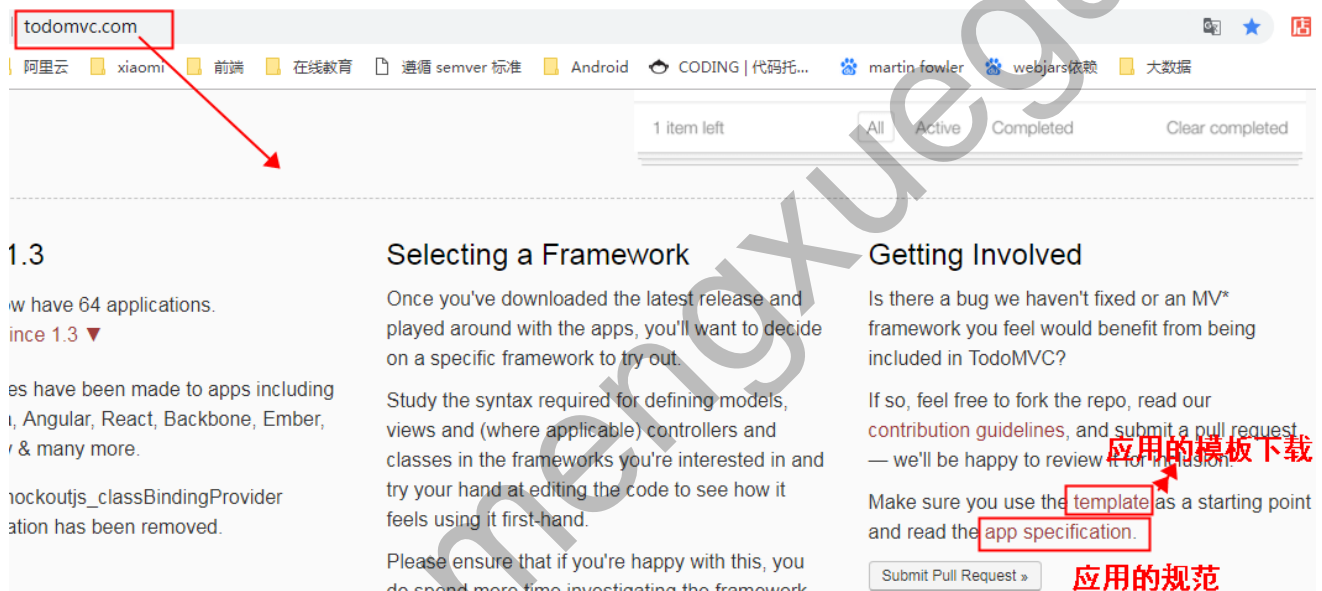
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>自定义指令</title>
6 </head>
7 <body>
8   <div id="app">
9     <p v-upper-text="message"></p>
10    自动获取焦点：<input type="text" v-focus>
11  </div>
12  <script src="../js/vue.js" type="text/javascript"></script>
13  <script type="text/javascript">
14    // 1. 注册一个全局 v-upper-text 指令，注意指令名不要带 v-
15    Vue.directive('upper-text', {
16      // 因为是样式，所以不需要元素插入到DOM中，就好像link引入CSS文件时并不关心元素是否加载
17      bind: function (el) {
18        el.style.color = 'red'
19      },
20      // el 代表使用了此指令的那个 DOM 元素
21      // binding 可获取使用了引指令的绑定值 等
22      inserted: function (el, binding) {
23        // 将在 v-upper-text 指令中获取到的值，变成大写输出到标签体中
24        el.innerHTML = binding.value.toUpperCase()
25      }
26    })
27
28    new Vue({
29      el: '#app',
30      data: {
31        message: 'mengxuegu，陪你学习伴你梦想'
32      },
33      // 2. 注册一个局部指令 v-focus
34      directives: {
35        'focus': {
36          // 和js行为有关的操作，最好在inserted中执行，和样式相关的操作都可在bind中执行
37          inserted: function (el) {
38            // 聚焦元素
39            el.focus()
40          }
41        }
42      }
43    })
```

```
44 </script>
45 </body>
46 </html>
```

第四章 经典实战项目-TodoMVC

4.1 项目介绍与演示

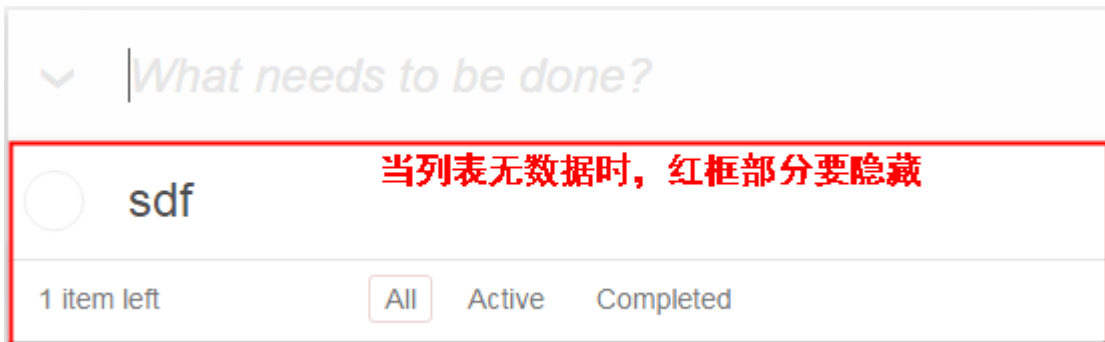
- TodoMVC 是一个非常经典的案例，功能非常丰富，并且针对多种不同技术分别都开发了此项目，比如 React、AngularJS、jQuery 等等。
- TodoMVC 案例官网：<http://todomvc.com/>
- 在官网首页右下角，有 案例的模板下载 和 开发规范（需求文档），如下图：



4.2 需求说明

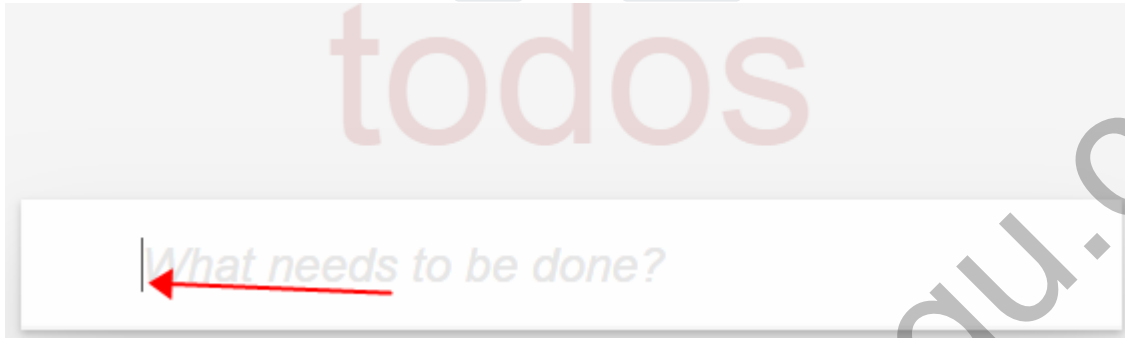
4.2.1 数据列表渲染

- 当任务列表（items）没有数据时，#main 和 #footer 标识的标签应该被隐藏
任务涉及字段：id、任务名称（name）、是否完成（completed true 为已完成）



4.2.2 添加任务

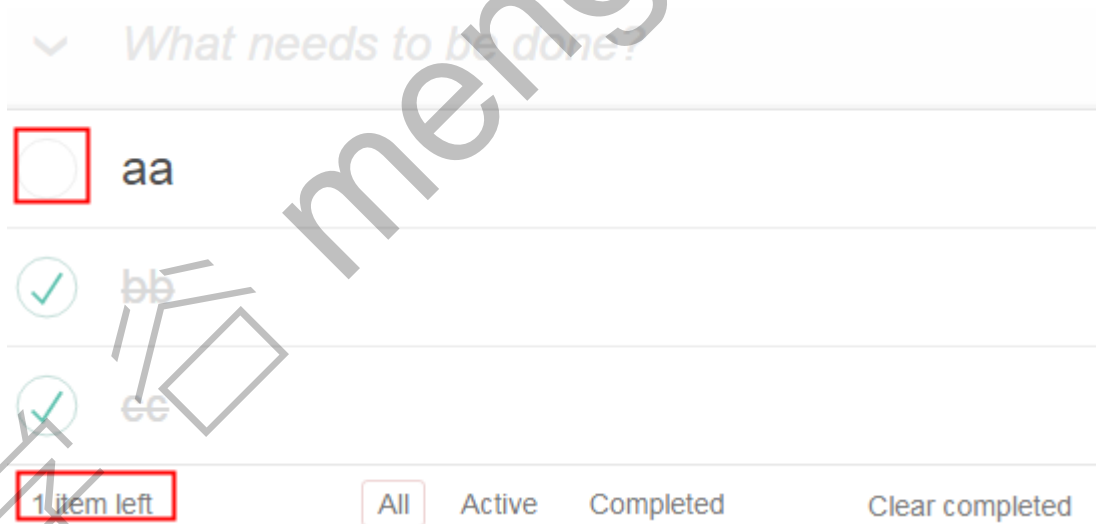
1. 在最上面的文本框中添加新的任务。
2. 不允许添加非空数据。
3. 按 `Enter` 键添加任务列表中，并清空文本框。
4. 当加载页面后文本框自动获得焦点，在 `input` 上使用 `autofocus` 属性可获得。



4.2.3 显示所有未完成任务数

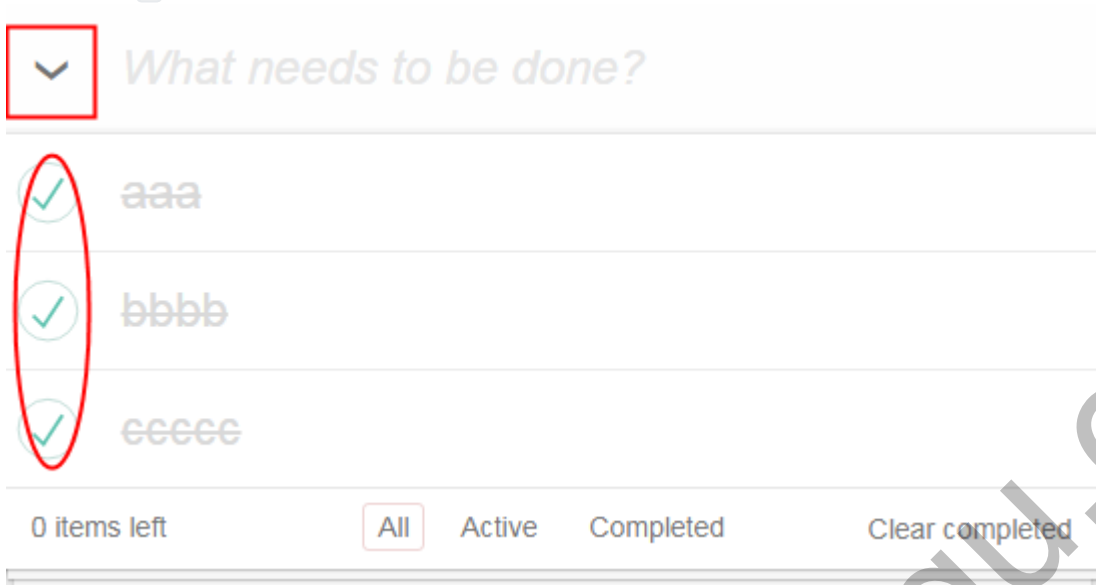
1. 左下角要显示未完成的任务数量。确保数字是由 `` 标签包装的。
2. 还要将 `item` 单词多元化(1 没有 s ,其他数字均有 s) : `0 items` , `1 item` , `2 items`

示例 : `2 items left`



4.2.4 切换所有任务状态

1. 点击复选框 ☒ 后，将所有任务状态标记为复选框相同的状态。

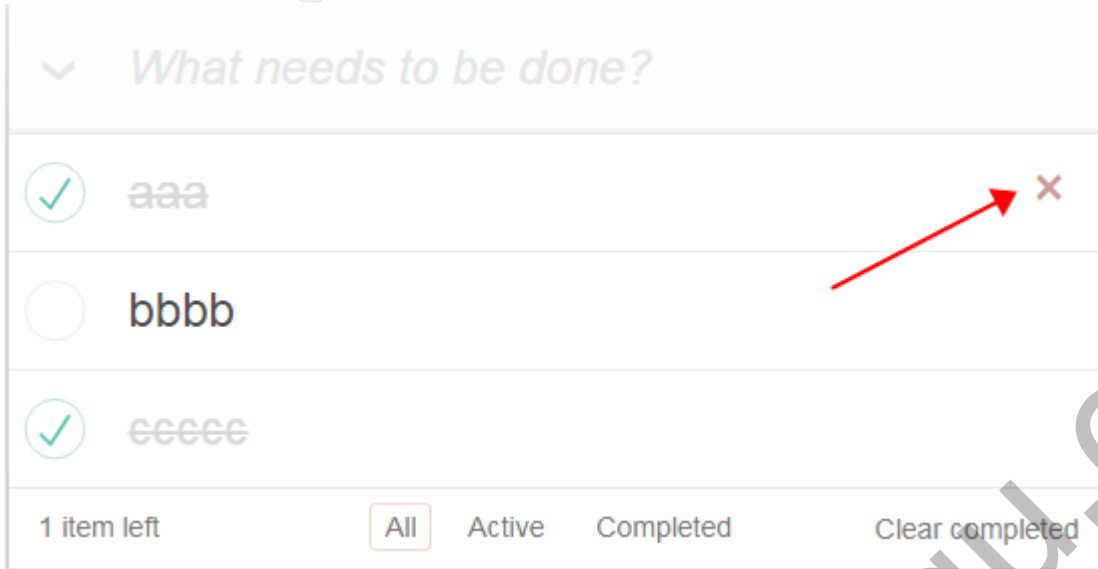


2. 当选中/取消了单个任务时，复选框 ☒ 也应同步更新。



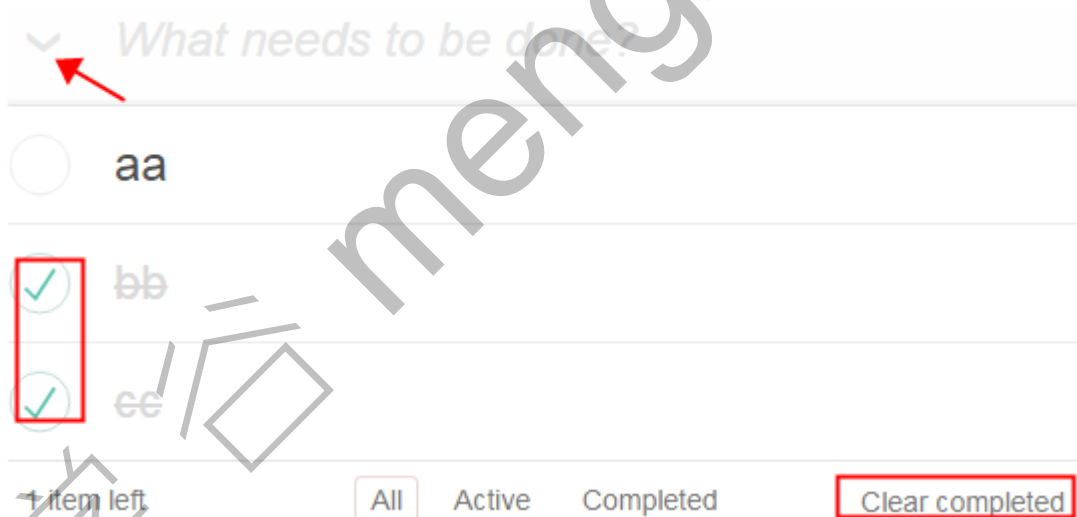
4.2.5 移除任务项

1. 悬停在某个任务项上显示 ☐ 移除按钮，可点击移除当前任务项。



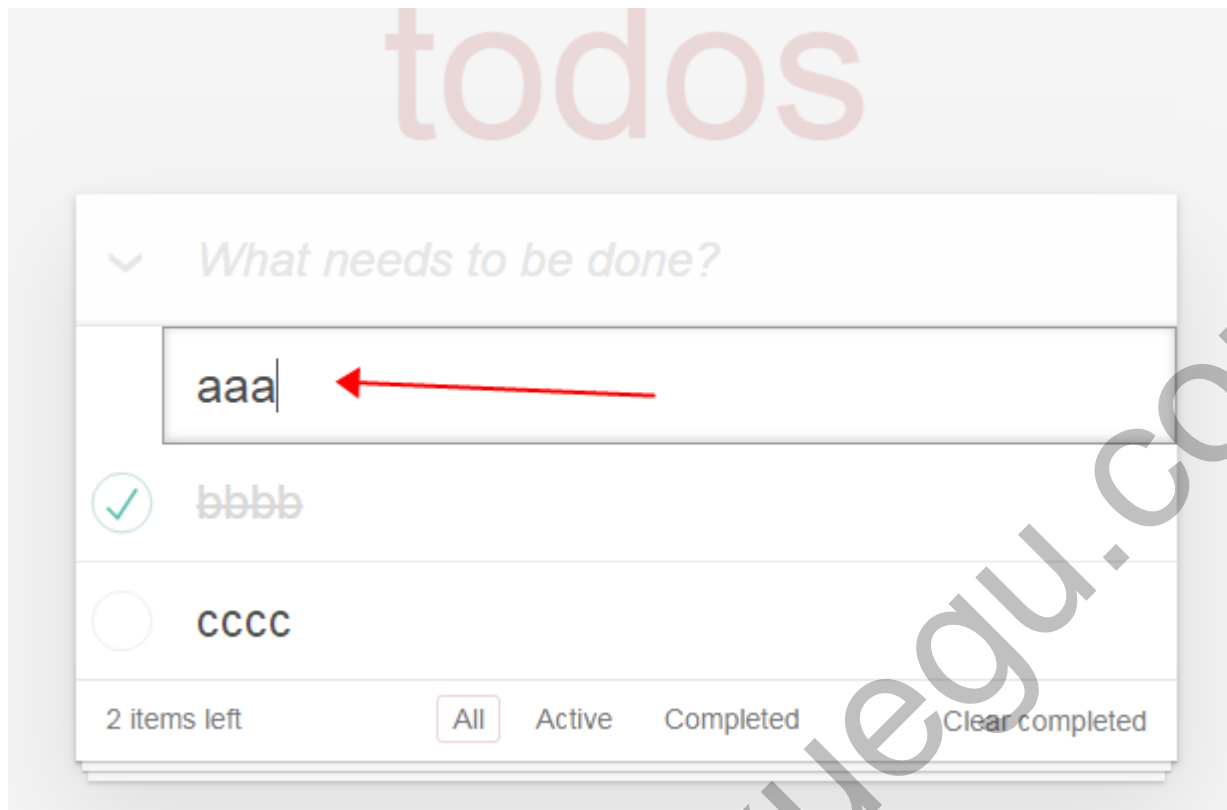
4.2.6 清除所有已完成任务

1. 单击右下角 按钮时，移除所有已完成任务。
2. 单击 按钮后，确保复选框清除了选中状态。
3. 当列表中没有已完成的任务时，应该隐藏 按钮。



4.2.7 编辑任务项

1. 双击 `<label>`（某个任务项）进入编辑状态（在 `` 上通过 `.editing` 进行切换状态）。
2. 进入编辑状态后输入框显示原内容，并获取编辑焦点。
3. 输入状态按 `Esc` 取消编辑，`editing` 样式应该被移除。
4. 按 `Enter` 键或失去焦点时保存改变数据，移除 `editing` 样式；



4.2.8 路由状态切换（过滤不同状态数据）

根据点击的不同状态（`All` / `Active` / `Completed`），进行过滤出对应的任务，并进行样式的切换。



4.2.9 数据持久化

将所有任务项数据持久化到 `localStorage` 中,它主要是用于本地存储数据。

4.3 下载与导入模板

在 Github 下载 TodoMVC 模板在：<https://github.com/tastejs/todomvc-app-template>

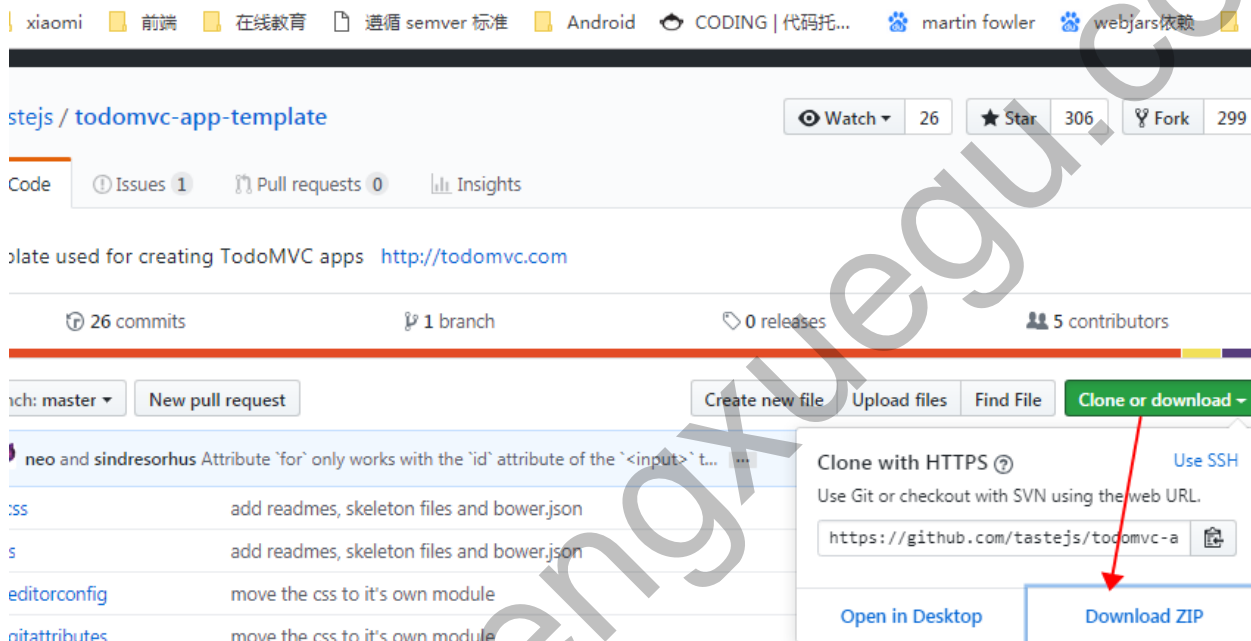
4.3.1 TodoMVC下载方式

1. 使用 git 克隆项目

```
1 git clone https://github.com/tastejs/todomvc-app-template.git
```

2. 直接下载 zip 压缩包

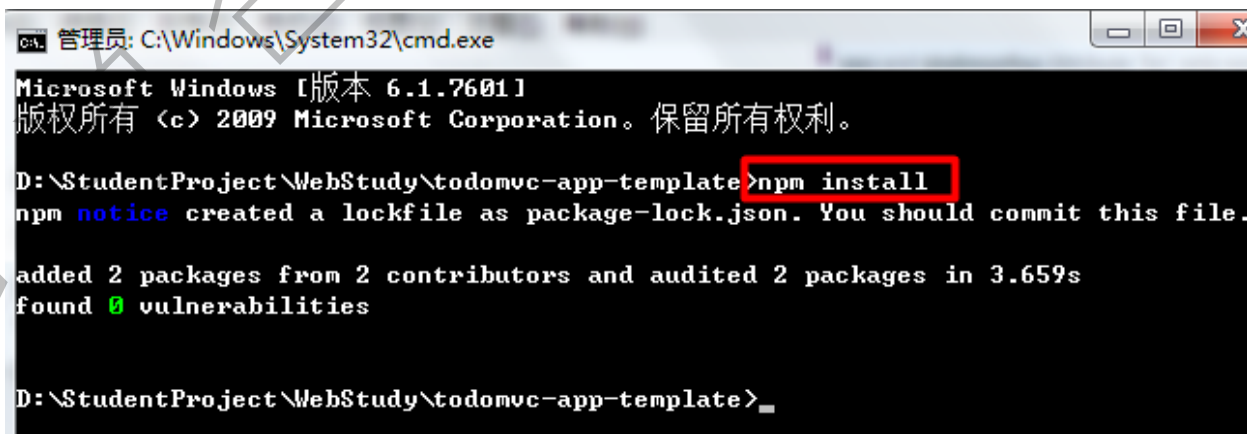
<https://github.com/tastejs/todomvc-app-template>



4.3.2 npm 安装依赖

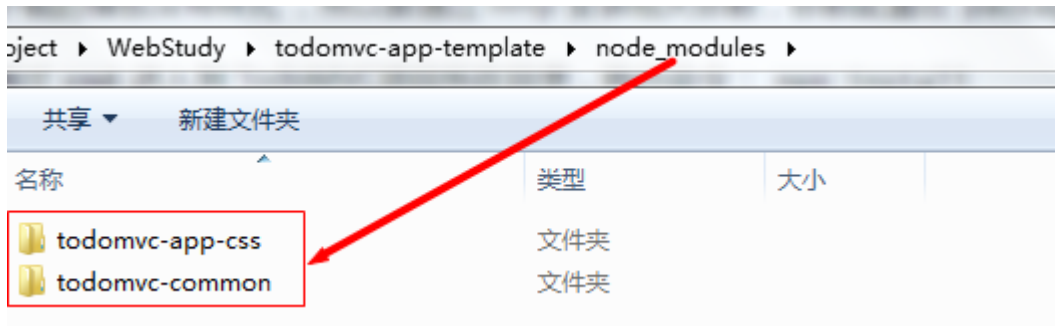
因为下载的模板没有样式，所以要通过 npm 安装相关依赖，依赖配置在 package.json 文件中

- 通过 cmd 进入到 TodoMVC 项目所在目录，执行命令：`npm install`



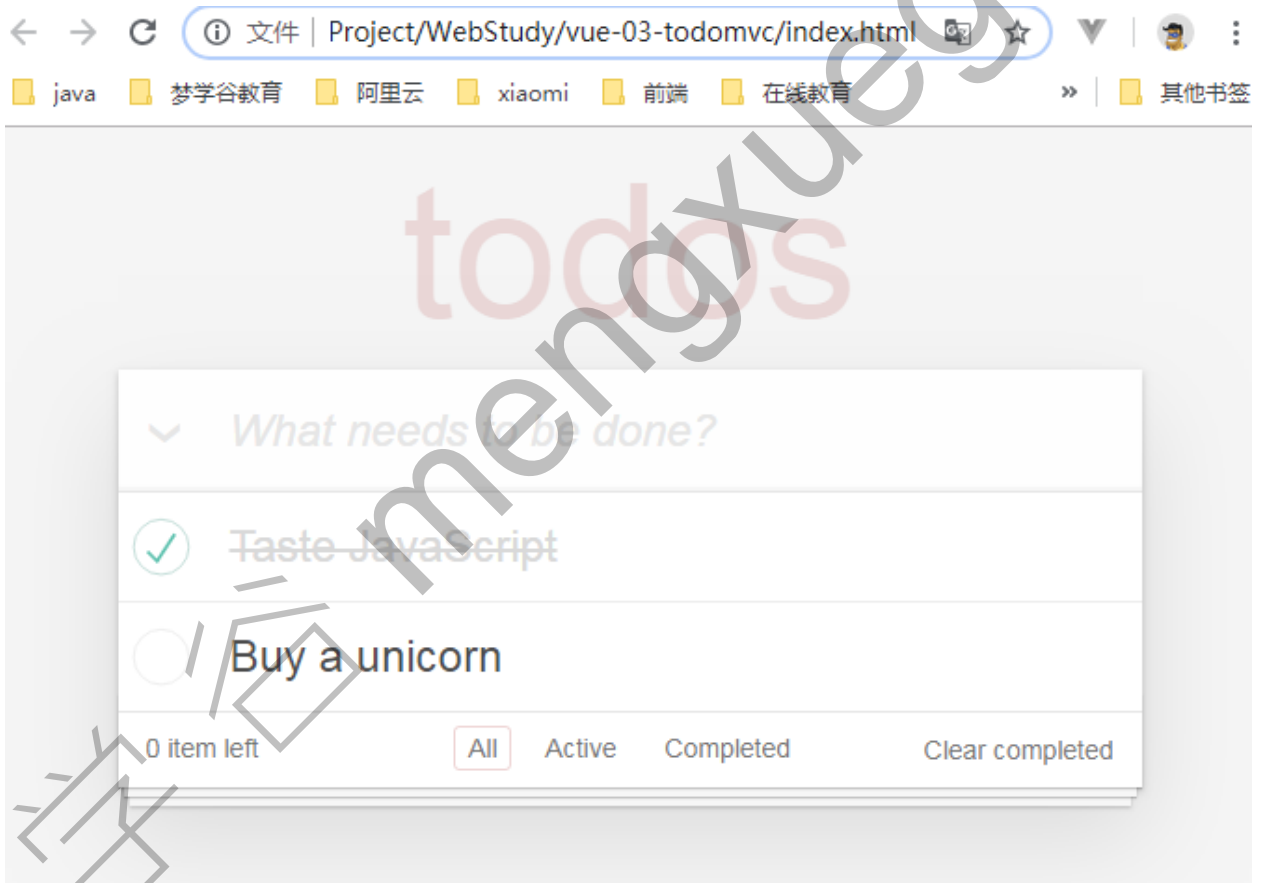
注意：要使用 npm 命令，需要安装 node.js 环境才可用。

- 依赖安装完成后，生成的 node_modules 目录效果如下：



4.3.3 导入到 VS Code

1. 将 todomvc-app-template 目录重命名为：vue-03-todomvc
2. 拷贝 vue-03-todomvc 拷贝到 D:\StudentProject\WebStudy 目录下
3. 在 VS Code 左侧资源管理窗口刷新一下就可以看到 vue-03-todomvc，直接访问 index.html，效果如下：



4.4 初始化项目

4.4.1 下载Vue依赖

- 下载 vue.js 后，会在 node_modules 目录下出现 vue

```
1 npm install vue@2.6.10
```

4.4.2 引入 vue.js

- 引入 `vue.js` 到 `index.html` 中

注意：vue.js 的引入要在使用 Vue 操作的前面，我们在 app.js 中编写Vue代码，所以要在 app.js 前面引入

```
1 <!-- 在 app.js 前面引入-->
2 <script src="./node_modules/vue/dist/vue.js" type="text/javascript"></script>
3 <script src="js/app.js"></script>
```

4.4.3 找到被 Vue 管理的元素

- 找到需要被 Vue 管理的元素 `<section>`，在第13行取一个id属性值为 `todoapp`

```
1 <section class="todoapp" id="todoapp">
```

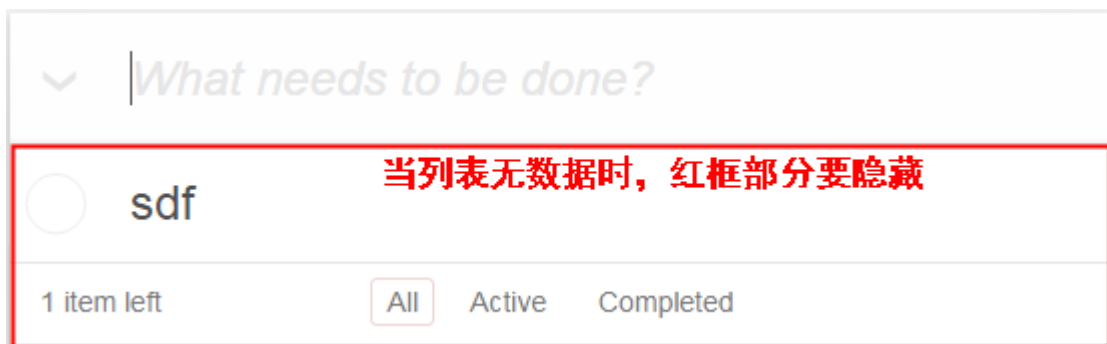
- 在 app.js 中添加如下 Vue 入口

```
1 (function (Vue) { //表示依赖了全局的 Vue
2   var app = new Vue({
3     el: '#todoapp'
4   })
5 })(Vue);
```

4.5 数据列表渲染实战

4.5.1 功能分析

- 有数据
 - 列表中的记录有3种状态且 `` 样式不一样：
未完成（没有样式）、已完成（`.completed`）、编辑中（`.editing`）
 - 任务字段：`id`（主键）、`content`（内容）、`completed`（状态；true 已完成, false 未完成）
- 无数据
 - `.main` 和 `.footer` 标识的标签应该被隐藏（`v-show`）



4.5.2 有数据列表功能实现

1. 在 app.js 声明一个存储任务数据的数组 items，并初始化一些数据。

```
1 (function (Vue) { //表示依赖了全局的 Vue, 其实不加也可以，只是更加明确点
2   //初始化任务
3   const items = [
4     {
5       id: 1,
6       content: 'vue.js',
7       completed: false //是否完成
8     },
9     {
10      id: 2,
11      content: 'java',
12      completed: true
13    },
14    {
15      id: 3,
16      content: 'pyhton',
17      completed: false
18    }
19  ]
20
21  var app = new Vue({
22    el: '#todoapp',
23    data: {
24      items // ES6中对象属性简写，等价于items: items
25    }
26  })
27 })(Vue);
```

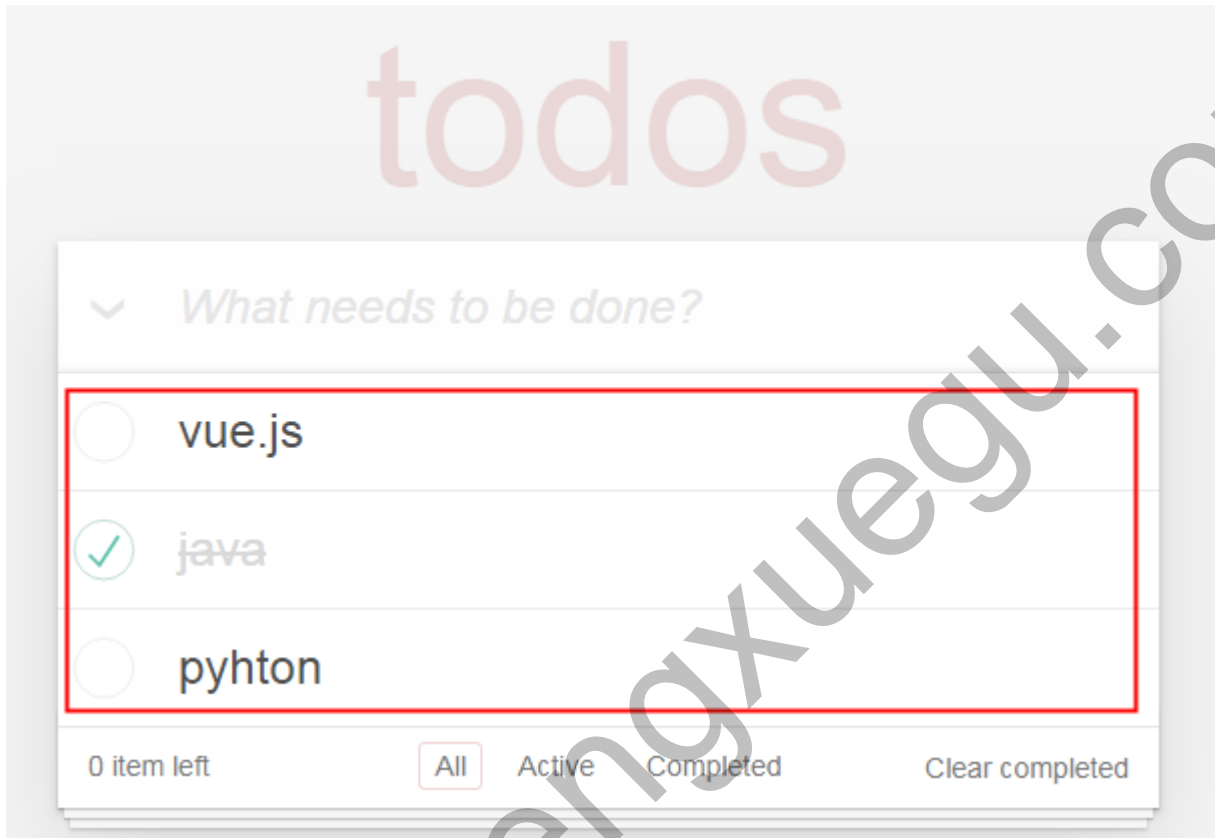
2. 修改 list.html 列表从第25行开始 `<li class="completed">`

如果修改 app.js 或 html 后，刷新浏览器发现没有变，关闭当前浏览器窗口，重新打开访问。

```
1 <ul class="todo-list">
2   <!--
3   三种状态：未完成（没有样式）、已完成（.completed）、编辑中(.editing)
4   -->
5   <!--修改: 1、v-for迭代; 2、:class={key为class样式名,value为获取的数据true或false}-->
6   <li v-for="(item, index) in items" :key="item.id" :class="{completed: item.completed}">
7     <div class="view">
8       <!-- 修改：1、v-model 绑定状态值是否选中 -->
9       <input class="toggle" type="checkbox" v-model="item.completed">
10      <!-- 修改：1、{{ content }} 显示内容 -->
11      <label>{{ item.content }}</label>
12      <!--修改: 1、:value 绑定id删除 -->
13      <button class="destroy" :value="item.id"></button>
14    </div>
```

```
15 <input class="edit" value="Create a TodoMVC template">
16 </li>
17 </ul>
```

3. 效果图



4.5.3 无数据隐藏功能实现

只要判断 items 数组 length 等于0, 则表示没有数据, 结合 `v-show` 即可

- 修改 index.html 中 `<section class="main">` 和 `<footer class="footer">`

```
1 方式1(todomvc官网方法): 在两个标签上处理, 要添加2次麻烦
2 <!-- items.length 值为 0 就是 false -->
3 <section class="main" v-show="items.length">
4   ...
5 </section>
6 <footer class="footer" v-show="items.length">
7   ...
8 </footer>
9
10 方式2: 使用 div 元素将下面包住, 但是页面渲染后多出 div 元素
11 <div v-show="items.length">
12   <section class="main">
13     ...
14   </section>
15   <footer class="footer">
```

```
16     ....
17     </footer>
18 </div>
19
20 方式3：可使用 Vue 提供的 template 元素，页面渲染后不会有 template 元素，
21 但是不能使用 v-show, 因为template渲染后就消失了，而v-show是 display:none控制显示隐藏的。
22 需要使用 v-if 才可以
23 <template v-if="items.length">
24     <section class="main">
25         ....
26     </section>
27     <footer class="footer">
28         ....
29     </footer>
30 </template>
```

- 在 app.js 中注释 items 数组中的元素后，重新访问即可演示效果



4.5.4 完整源码

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <title>Template • TodoMVC</title>
7     <link rel="stylesheet" href="node_modules/todomvc-common/base.css">
8     <link rel="stylesheet" href="node_modules/todomvc-app-css/index.css">
9     <!-- CSS overrides - remove if you don't need it -->
10    <link rel="stylesheet" href="css/app.css">
11  </head>
12  <body>
13    <section class="todoapp" id="todoapp">
14      <header class="header">
15        <h1>todos</h1>
16        <!-- 添加任务 -->
```

```
17     <input class="new-todo" placeholder="What needs to be done?" autofocus>
18   </header>
19   <!-- This section should be hidden by default and shown when there are todos -->
20   <!--修改: items.length 值为 0 就是 false -->
21   <template v-if="items.length">
22     <section class="main">
23       <input id="toggle-all" class="toggle-all" type="checkbox">
24       <label for="toggle-all">Mark all as complete</label>
25       <ul class="todo-list">
26         <!-- These are here just to show the structure of the list items -->
27         <!-- List items should get the class `editing` when editing and `completed` when marked as
completed -->
28         <!--
29         三种状态：未完成（没有样式）、已完成（.completed）、编辑中(.editing)
30         -->
31         <!--修改: 1、v-for迭代; 2、:class={key为class样式名,value为获取的数据true或false}-->
32         <li v-for="(item, index) in items" :class="{completed: item.completed}">
33           <div class="view">
34             <!-- 修改：1、v-model 绑定状态值是否选中 -->
35             <input class="toggle" type="checkbox" v-model="item.completed">
36             <!-- 修改：1、{{ content }} 显示内容 -->
37             <label>{{ item.content }}</label>
38             <!--修改: 1、:value 绑定id删除 -->
39             <button class="destroy" :value="item.id"></button>
40           </div>
41           <input class="edit" value="Create a TodoMVC template">
42         </li>
43       </ul>
44     </section>
45     <!-- This footer should hidden by default and shown when there are todos -->
46     <!--修改: items.length 值为 0 就是 false -->
47     <footer class="footer">
48       <!-- This should be `0 items left` by default -->
49       <span class="todo-count"><strong>0</strong> item left</span>
50       <!-- Remove this if you don't implement routing -->
51       <ul class="filters">
52         <li>
53           <a class="selected" href="#">All</a>
54         </li>
55         <li>
56           <a href="#/active">Active</a>
57         </li>
58         <li>
59           <a href="#/completed">Completed</a>
60         </li>
61       </ul>
62       <!-- Hidden if no completed items are left ↓ -->
63       <button class="clear-completed">Clear completed</button>
64     </footer>
65
66   </template>
67 </section>
68 <footer class="info">
```

```
69     <p>Double-click to edit a todo</p>
70     <!-- Remove the below line ↓ -->
71     <p>Template by <a href="http://sindresorhus.com">Sindre Sorhus</a></p>
72     <!-- Change this out with your name and url ↓ -->
73     <p>Created by <a href="http://todomvc.com">you</a></p>
74     <p>Part of <a href="http://todomvc.com">TodoMVC</a></p>
75   </footer>
76   <!-- Scripts here. Don't remove ↓ -->
77   <script src="node_modules/todomvc-common/base.js"></script>
78   <!-- 在 app.js 前面-->
79   <script src="node_modules/vue/dist/vue.js" type="text/javascript"></script>
80   <script src="js/app.js"></script>
81 </body>
82 </html>
```

4.6 添加任务实战

4.6.1 功能分析

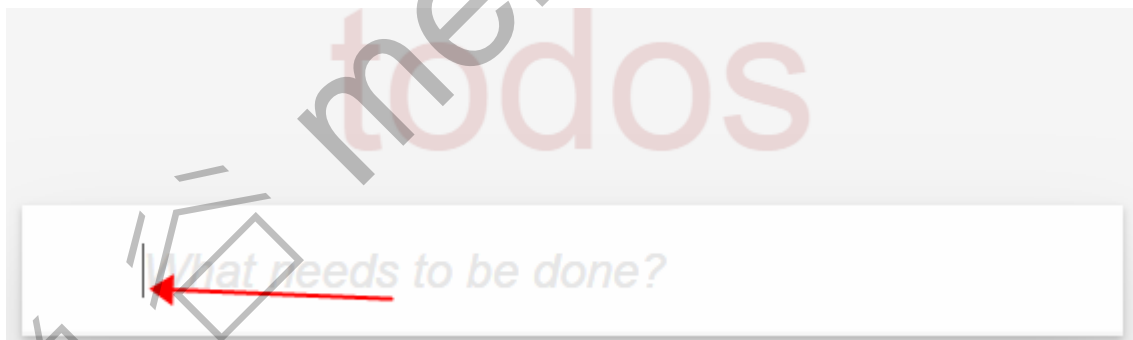
1. 在最上面的文本框中添加新的任务。

2. 不允许添加非空数据。

分析：在添加到列表前，使用 `.trim()` 去除空格，如果去除后是空的就不添加。

3. 按 `Enter` 键添加任务列表中，并清空文本框。

分析：在页面添加 `Enter` 按键监听事件，最后向文本框赋空值 `"`



4.6.2 添加任务功能实现

1. 在 `hello.html` 中的文本输入框，添加 `Enter` 按键监听事件 `@keyup.enter="addItem"`

```
1 <header class="header">
2   <h1>todos</h1>
3   <!-- 添加任务, keyup.enter 回车键监听-->
4   <input @keyup.enter="addItem"
5     class="new-todo" placeholder="What needs to be done?" autofocus>
6 </header>
```

2. 在 `app.js` 中添加 `addItem` 函数，步骤如下：

- 1、获取文本框输入的数据
- 2、判断数据如果为空，则什么都不做
- 3、如果不为空，则添加到数组中
 - 生成id值
 - 添加到数组中(默认状态为 未完成)
- 4、清空文本框内容

```
1 var app = new Vue({
2   el: '#todoapp',
3   data: {
4     items // 对象属性简写，等价于items: items
5   },
6   methods: {
7     addItem(event) { //对象属性函数简写，等价于addItem: function () {
8       console.log('addItem', event.target.value)
9       //1. 获取文本框输入的数据
10      const content = event.target.value.trim()
11      //2. 判断数据如果为空，则什么都不做
12      if (content.length) {
13        return
14      }
15      //3.如果不为空，则添加到数组中
16      // 生成id值
17      const id = this.items.length + 1
18      this.items.push({
19        id, //等价于 id:id
20        content,
21        completed: false
22      })
23      //4. 清空文本框内容
24      event.target.value = ""
25    }
26  }
27 })
```

4.6.3 完整源码

• `app.js`

```
1 (function (Vue) { //表示依赖了全局的 Vue, 其实不加也可以，只是更加明确点
2   //初始化任务
3   const items = [
4     /*{
5       id: 1,
6       content: 'vue.js',
7       completed: false //是否完成
```



```
8      },
9      {
10         id: 2,
11         content: 'java',
12         completed: true
13      },
14      {
15         id: 3,
16         content: 'python',
17         completed: false
18      }
19   ]
20
21   var app = new Vue({
22     el: '#todoapp',
23     data: {
24       items // 对象属性简写，等价于 items: items
25     },
26     methods: {
27       addItem (event) { // 对象属性函数简写，等价于 addItem: function () {
28         console.log('addItem', event.target.value)
29         //1. 获取文本框输入的数据
30         const content = event.target.value.trim()
31         //2. 判断数据如果为空，则什么都不做
32         if (content.length) {
33           return
34         }
35         //3. 如果不为空，则添加到数组中
36         // 生成id值
37         const id = this.items.length + 1
38         // 添加到数组中
39         this.items.push({
40           id, // 等价于 id: id
41           content,
42           completed: false
43         })
44         //4. 清空文本框内容
45         event.target.value = ""
46       }
47     }
48   })
49 })(Vue);
```

4.7 显示所有未完成任务数实战

4.7.1 功能分析

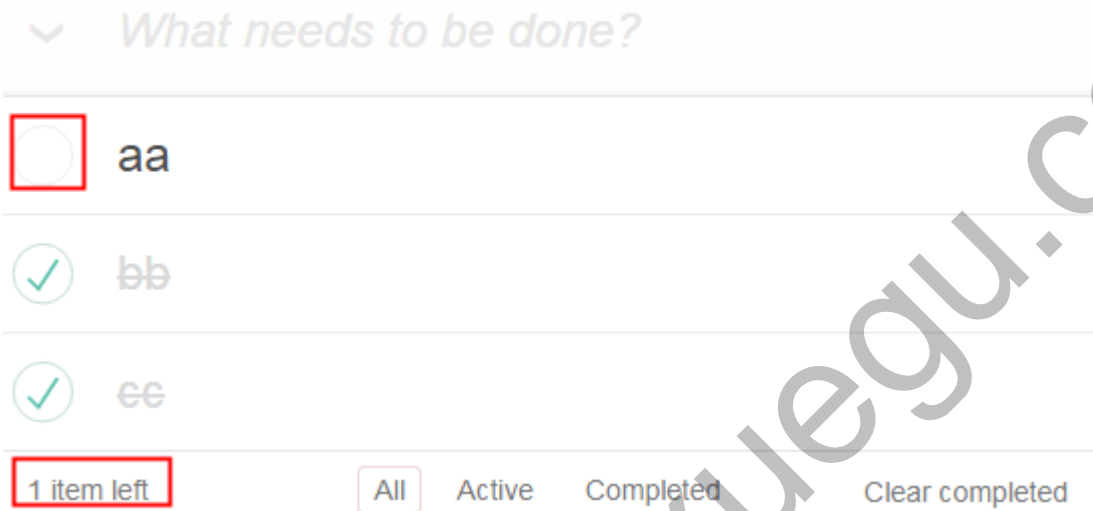
1. 左下角要显示**未完成**的任务数量。数字是由 `` 标签包装的。

分析:

- 当 items 数组中的元素有改变，则重新计算未完成任务数量，可通过 **计算属性** 来获取 **未完成** 的任务数量
 - 通过数组函数 **filter** 过滤未完成任务，然后进行汇总
2. 还要将 item 单词多元化(1 没有 s，其他数字均有 s)：0 items, 1 item, 2 items

分析：

- 当任务数据为 1 不显示 s，否则显示



4.7.2 功能实现

- 在 list.html 页面添加剩余任务数 **计算属性** {{ remaining }}

```
1 <span class="todo-count"><strong>{{ remaining }}</strong> item left</span>
```

- 在 app.js 的Vue实例的 methods 上一行添加一个 computed 选项，其中定义一个计算属性 remaining

注意 computed 选项的大括号最后不要少了逗号 ,

```
1 (function (Vue) { //表示依赖了全局的 Vue, 其实不加也可以，只是更加明确点
2   //初始化任务
3   const items = []
4
5   var app = new Vue({
6     el: '#todoapp',
7     data: {
8       items // 对象属性简写，等价于items: items
9     },
10    // 定义计算属性选项
11    computed: {
12      // 过滤出所有未完成的任务项
13      remaining () {
14        /*
15         return this.items.filter(function (item) {
16           return !item.completed
17         }).length
```

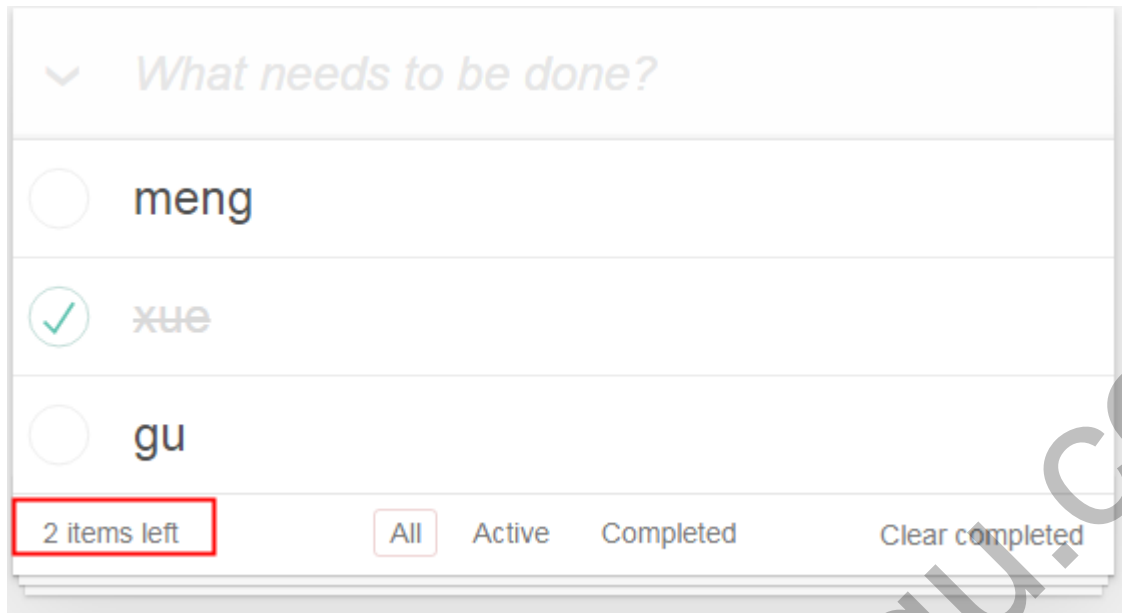
```
18      */
19      //ES6 箭头函数
20      return this.items.filter(item => item.completed).length
21    }
22  }, // **注意** 后面不要少了逗号 ,
23
24  methods: {
25    addItem (event) { //对象属性函数简写，等价于addItem: function () {
26      console.log('addItem', event.target.value)
27      //1. 获取文本框输入的数据
28      const content = event.target.value.trim()
29      //2. 判断数据如果为空，则什么都不做
30      if (content.length) {
31        return
32      }
33      //3. 如果不为空，则添加到数组中
34      // 生成id值
35      const id = this.items.length + 1
36      // 添加到数组中
37      this.items.push({
38        id, //等价于 id:id
39        content,
40        completed: false
41      })
42      //4. 清空文本框内容
43      event.target.value = ""
44    }
45  }
46 })
47 })(Vue);
```

3. 将 list.html 显示的 item 单词多元化(1 没有 s , 其他数字均有 s)

添加 {{ remaining === 1 ? '' : 's' }}

```
1 <span class="todo-count">
2   <strong>{{ remaining }}</strong> item{{ remaining === 1 ? '' : 's' }} left
3 </span>
```

4. 效果图



4.8 切换所有任务状态实战

4.8.1 功能分析

1. 点击复选框 ☐ 后，将所有任务状态标记为复选框相同的状态。

分析：

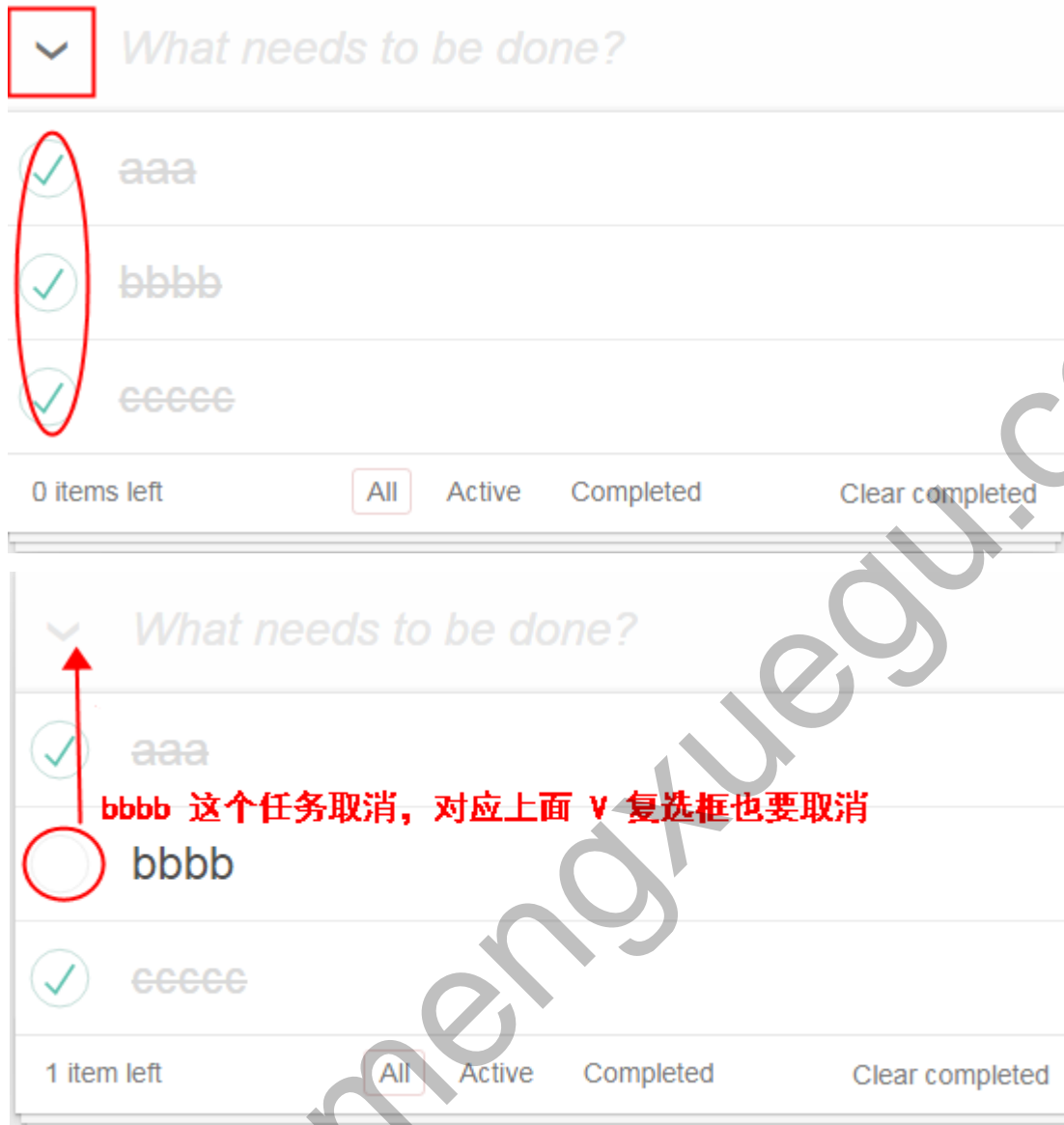
- 复选框状态发生变化后，就迭代出每一个任务项，再将复选框状态值赋给每个任务项即可。
- 为复选框绑定一个 **计算属性**，通过这个计算属性的 `set` 方法监听复选框更新后就更新任务项状态。

2. 当 **选中/取消** 某个任务后，复选框 ☐ 也应同步更新状态。

分析：

- 当所有未完成任务数(`remaining`)为 0 时，表示所有任务都完成了，就可以选中复选框。
 - 在复选框绑定的 **计算属性** 的 `get` 方法中判断所有 `remaining` 是否为 0，从而绑定了 `remaining`，当 `remaining` 发生变化后，会自动更新复选框状态（为 0 复选框会自动选中，反之不选中）。

综合上述：计算属性默认情况只有 `getter` 方法，而当前要用到 `setter` 方法，所以采用 计算属性 双向绑定。



4.8.2 切换所有任务状态功能实现

1. 在 `index.html` 中的 `id="toggle-all"` 复选框，添加计算属性 `toggleAll` 双向绑定

```
1 <section class="main">
2   <!-- 添加计算属性 toggleAll 双向绑定-->
3   <input v-model="toggleAll"
4     id="toggle-all" class="toggle-all" type="checkbox">
```

2. 在 `app.js` 中的步骤如下：

在 `computed` 选项中，添加计算属性 `toggleAll` 进行双向绑定 (getter/setter)

```
1 // 定义计算属性选项
2 computed: {
3   //复选框计算属性(双向绑定)
4   toggleAll: {
```

```
5      get() { //等价于 get : function () {...}
6          console.log(this.remaining)
7          //2. 当 this.remaining 发生变化后，会触发该方法运行
8          // 当所有未完成任务数为 0，表示全部完成，则返回 true 让复选框选中
9          //反之就 false 不选中
10         return this.remaining === 0
11     },
12     set(newStatus) {
13         // console.log(newStatus)
14         //1. 当点击 checkbox 复选框后状态变化后，就会触发该方法运行，
15         // 迭代出数组每个元素，把当前状态值赋给每个元素的 completed
16         // 下面箭头函数等价于：this.items.forEach(function (item) {
17         this.items.forEach((item) => {
18             item.completed = newStatus
19         })
20     }
21 },
22
23 // 过滤出所有未完成的任务项
24 remaining() {
25     /*
26     return this.items.filter(function (item) {
27         return !item.completed
28     }).length
29     */
30     //ES6 箭头函数
31     return this.items.filter(item => !item.completed).length
32 }
33 }, // **注意** 后面不要少了逗号
```

4.8.3 完整源码

1. index.html

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1">
6      <title>Template • TodoMVC</title>
7      <link rel="stylesheet" href="node_modules/todomvc-common/base.css">
8      <link rel="stylesheet" href="node_modules/todomvc-app-css/index.css">
9      <!-- CSS overrides - remove if you don't need it -->
10     <link rel="stylesheet" href="css/app.css">
11 </head>
12 <body>
13     <section class="todoapp" id="todoapp">
14         <header class="header">
15             <h1>todos</h1>
16
17     <!-- 添加任务, keyup.enter 回车键监听 -->
```

```
17 <input @keyup.enter="addItem"
18 class="new-todo" placeholder="What needs to be done?" autofocus>
19 </header>
20 <!-- This section should be hidden by default and shown when there are todos -->
21 <!--修改: items.length 值为 0 就是 false -->
22 <template v-if="items.length">
23 <section class="main">
24 <!-- 添加计算属性 toggleAll 双向绑定-->
25 <input v-model="toggleAll"
26 id="toggle-all" class="toggle-all" type="checkbox">
27 <label for="toggle-all" >Mark all as complete</label>
28 <ul class="todo-list">
29 <!-- These are here just to show the structure of the list items -->
30 <!-- List items should get the class `editing` when editing and `completed` when marked
as completed -->
31 <!--
32 三种状态：未完成（没有样式）、已完成（.completed）、编辑中(.editing)
33 -->
34 <!-- 修改: 1、v-for 迭代; 2、:class {key为class样式名, value为获取的数据true或false} -->
35 <li v-for="(item, index) in items" :class="{completed: item.completed}">
36 <div class="view">
37 <!-- 修改：1、v-model 绑定状态值是否选中 -->
38 <input class="toggle" type="checkbox" v-model="item.completed">
39 <!-- 修改：1、{{ content }} 显示内容 -->
40 <label>{{ item.content }}</label>
41 <!--修改: 1、:value 绑定id删除 -->
42 <button class="destroy" :value="item.id"></button>
43 </div>
44 <input class="edit" value="Create a TodoMVC template">
45 </li>
46 </ul>
47 </section>
48 <!-- This footer should hidden by default and shown when there are todos -->
49 <!--修改: items.length 值为 0 就是 false -->
50 <footer class="footer">
51 <!-- This should be `0 items left` by default -->
52 <span class="todo-count">
53 <strong>{{ remaining }}</strong> item{{ remaining === 1 ? '' : 's' }} left
54 </span>
55 <!-- Remove this if you don't implement routing -->
56 <ul class="filters">
57 <li>
58 <a class="selected" href="#/">All</a>
59 </li>
60 <li>
61 <a href="#/active">Active</a>
62 </li>
63 <li>
64 <a href="#/completed">Completed</a>
65 </li>
66 </ul>
67 <!-- Hidden if no completed items are left ↓ -->
68 <button class="clear-completed">Clear completed</button>
```

```
69     </footer>
70
71   </template>
72 </section>
73 <footer class="info">
74   <p>Double-click to edit a todo</p>
75   <!-- Remove the below line ↓ -->
76   <p>Template by <a href="http://sindresorhus.com">Sindre Sorhus</a></p>
77   <!-- Change this out with your name and url ↓ -->
78   <p>Created by <a href="http://todomvc.com">you</a></p>
79   <p>Part of <a href="http://todomvc.com">TodoMVC</a></p>
80 </footer>
81 <!-- Scripts here. Don't remove ↓ -->
82 <script src="node_modules/todomvc-common/base.js"></script>
83 <!-- 在 app.js 前面-->
84 <script src="node_modules/vue/dist/vue.js" type="text/javascript"></script>
85 <script src="js/app.js"></script>
86 </body>
87 </html>
```

2. app.js

```
1 (function (Vue) { //表示依赖了全局的 Vue, 其实不加也可以, 只是更加明确点
2   //初始化任务
3   const items = []
4
5   var app = new Vue({
6     el: '#todoapp',
7     data: {
8       items // 对象属性简写, 等价于 items: items
9     },
10    // 定义计算属性选项
11    computed: {
12      //复选框计算属性
13      toggleAll: {
14        get () { //等价于 get : function () {...}
15          console.log(this.remaining)
16          //2. 当 this.remaining 发生变化后, 会触发该方法运行
17          // 当所有未完成任务数为 0, 表示全部完成, 则返回 true 让复选框选中
18          //反之就 false 不选中
19          return this.remaining === 0
20        },
21        set (newStatus) {
22          // console.log(newStatus)
23          //1. 当点击 checkbox 复选框后状态变化后, 就会触发该方法运行,
24          // 迭代出数组每个元素,把当前状态值赋给每个元素的 completed
25          this.items.forEach((item) => {
26            item.completed = newStatus
27          })
28        }
29      },
30  })
```



```
31 // 过滤出所有未完成任务项
32 remaining() {
33   /*
34   return this.items.filter(function (item) {
35     return !item.completed
36   }).length
37   */
38   //ES6 箭头函数
39   return this.items.filter(item => !item.completed).length
40 }
41 }, // **注意** 后面不要少了逗号 ,
42
43 methods: {
44   addItem(event) { //对象属性函数简写，等价于addItem: function () {
45     console.log('addItem', event.target.value)
46     //1. 获取文本框输入的数据
47     const content = event.target.value.trim()
48     //2. 判断数据如果为空，则什么都不做
49     if (content.length) {
50       return
51     }
52     //3. 如果不为空，则添加到数组中
53     // 生成id值
54     const id = this.items.length + 1
55     // 添加到数组中
56     this.items.push({
57       id, //等价于 id:id
58       content,
59       completed: false
60     })
61     //4. 清空文本框内容
62     event.target.value = ""
63   }
64 }
65 })
66 })(Vue);
```

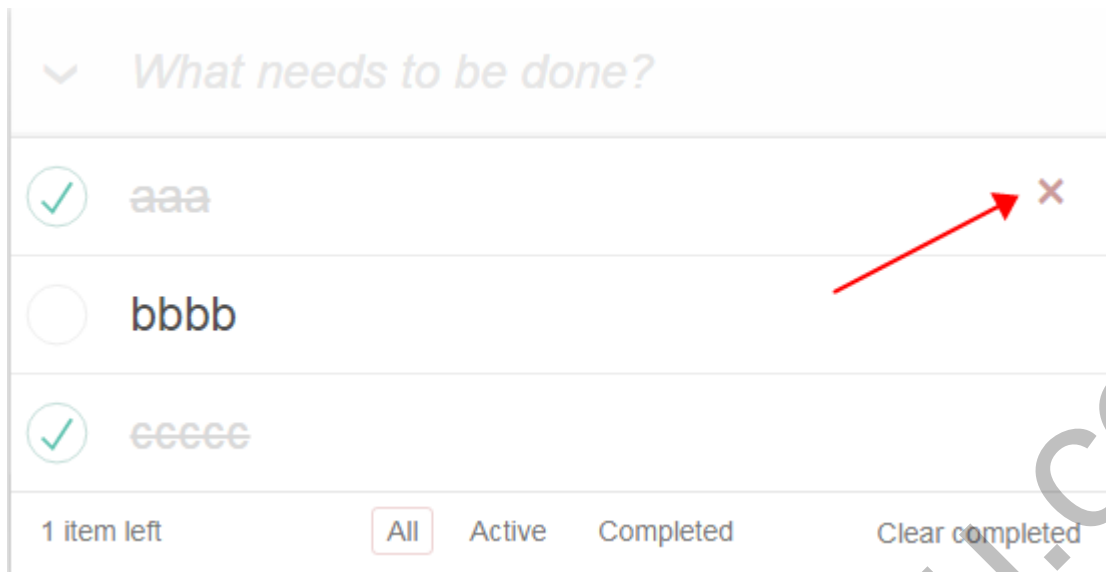
4.9 移除任务项

4.9.1 功能分析

1. 悬停在某个任务项上显示 ☐ 移除按钮，可点击移除当前任务项。

分析：

- ☐ 移除按钮处添加点击事件
- 通过数组函数 `splice` 移除任务项



4.9.2 移除任务项功能实现

- index.html 添加点击事件：`@click="removeItem(index)"`

```
1 <button class="destroy" :value="item.id" @click="removeItem(index)"></button>
```

- app.js 添加函数 `removeItem`，通过 `this.items.splice(index, 1)` 移除

```
1 methods: {  
2   // 移除任务项  
3   removeItem (index) {  
4     // 移除索引为index的一条记录  
5     this.items.splice(index, 1)  
6   },  
7  
8   //增加任务项  
9   addItem (event) { //对象属性函数简写，等价于addItem: function () {  
10    console.log('addItem', event.target.value)  
11    //1. 获取文本框输入的数据  
12    const content = event.target.value.trim()  
13    //2. 判断数据如果为空，则什么都不做  
14    if (!content.length) {  
15      return  
16    }  
17    //3.如果不为空，则添加到数组中  
18    // 生成id值  
19    const id = this.items.length + 1  
20    // 添加到数组中  
21    this.items.push({  
22      id, //等价于 id:id  
23      content,  
24      completed: false  
25    })
```

```
26 //4. 清空文本框内容
27 event.target.value = ""
28 }
29 }
```

4.10 清除所有已完成任务

4.10.1 功能分析

1. 单击右下角 Clear completed 按钮时，移除所有已完成任务。

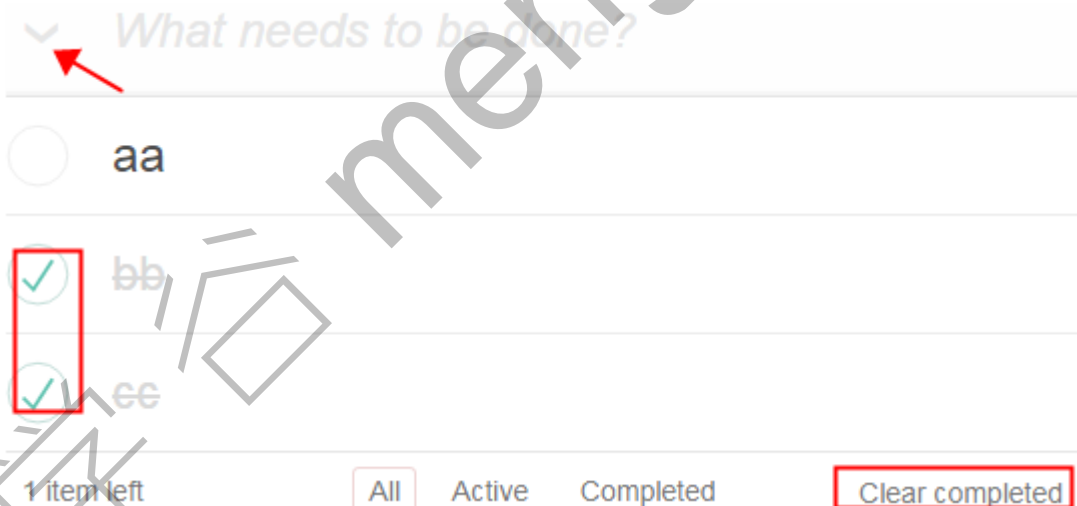
分析：

- 页面增加点击事件：@click="removeCompleted"
- 在 Vue 中添加 removeCompleted 函数：
通过数组的 filter 函数过滤出所有未完成的任务项，将过滤出来的未完成数据赋值给 items 数组，已完成的任务就被删除。

2. 当列表中没有已完成的任务时，应该隐藏 Clear completed 按钮。

分析：

- 在 Clear completed 按钮上使用 v-show，当总任务数 (items.length) > 未完成数 (remaining)，说明列表中还有已完成数据，则是显示按钮；反之不显示。
- 实现：v-show="items.length > remaining"



4.10.2 清除所有已完成任务功能实现

1. index.html 添加点击事件：@click="removeCompleted"

```
1 <button @click="removeCompleted" class="clear-completed">Clear completed</button>
```

2. app.js 添加函数 removeCompleted，通过数组的 filter 函数过滤出所有未完成的任务项，将过滤出来的未完成数据赋值给 items 数组

```
1 methods: {  
2   //移除所有未完成任务项  
3   removeCompleted () {  
4     // 过滤出所有未完成任务，重新赋值数组即可  
5     this.items = this.items.filter(item => !item.completed)  
6   },  
7  
8   // 移除任务项  
9   removeItem (index) {  
10    // 移除索引为index的一条记录  
11    this.items.splice(index, 1)  
12  },  
13  
14  //增加任务项  
15  addItem (event) { //对象属性函数简写，等价于addItem: function () {  
16    console.log('addItem', event.target.value)  
17    //1. 获取文本框输入的数据  
18    const content = event.target.value.trim()  
19    //2. 判断数据如果为空，则什么都不做  
20    if (!content.length) {  
21      return  
22    }  
23    //3. 如果不为空，则添加到数组中  
24    // 生成id值  
25    const id = this.items.length + 1  
26    // 添加到数组中  
27    this.items.push({  
28      id, //等价于 id:id  
29      content,  
30      completed: false  
31    })  
32    //4. 清空文本框内容  
33    event.target.value = "  
34  }  
35 }
```

3. 在 index.html 使用 v-show="items.length > remaining" 进行切换显示/隐藏 Clear completed 按钮

```
1 <button @click="removeCompleted" v-show="items.length > remaining"  
2   class="clear-completed">Clear completed</button>
```

4.11 编辑任务项

4.11.1 功能分析

1. 双击 <label> (某个任务项) 进入编辑状态 (在 上通过 .editing 进行切换状态)。

分析：

- 为 <label> 绑定双击事件 @dblclick=toEdit(item)

- 当 `item` (任务项) `=== currentItem` (当前点击的任务项, `data`中新定义的属性) 时, 在 `` 上就显示 `.editing` 样式, 格式 `:class={editing: item === currentItem}`

2. 进入编辑状态后输入框显示原内容, 并会自动获取编辑焦点。

分析:

- 在 `<input>` 单向绑定输入框的值即可 `:value="item.content"`
- 通过自定义指令获取编辑焦点

3. 输入状态按 `Esc` 取消编辑, `editing` 样式应该被移除。

分析: 为 `<input>` 绑定 `Esc` 按键事件 `@keyup.esc=cancelEdit`, 将 `currentItem` 值变为 `null`

4. 按 `Enter` 键或失去焦点时保存改变数据, 移除 `editing` 样式;

分析:

- 添加事件 `@keyup.enter=finishEdit(item, $event)` 与 `@blur="finishEdit(item, $event)"`
- 通过 `$event` 变量获取当前输入框的值, 使用 `.trim()` 去空格后进行判断是否为空, 如果为空则清除这条任务, 否则修改任务项;
- 添加数据保存任务项中
- 将 `currentItem` 值变为 `null`, 移除 `editing` 样式

4.11.2 进入编辑状态

4.11.2.1 修改 index.html

- 为 `<label>` 绑定双击事件 `@dblclick="toEdit(item)"`, 将迭代出来的每个 `item` 传入当前行的 `toEditItem` 函数中
- 在 `` 上判断是否显示 `.editing` 样式 `:class={editing: item === currentItem}`

```
1 <!-- 修改: 1、v-for 迭代; 2、:class {key为class样式名: value为获取的数据true或false} -->
2 <li v-for="(item, index) in items"
3   :class="{completed: item.completed, editing: item === currentItem}">
4   <div class="view">
5     <!-- 修改: 1、v-model 绑定状态值是否选中 -->
6     <input class="toggle" type="checkbox" v-model="item.completed">
7     <!-- 修改: 1、{{ content }} 显示内容 -->
8     <label @dblclick="toEdit(item)">{{ item.content }}</label>
9     <!--修改: 1、:value 绑定id删除 -->
10    <button class="destroy" :value="item.id" @click="removeItem(index)"></button>
11  </div>
12  <input class="edit" value="Create a TodoMVC template">
13 </li>
```

4.11.2.2 修改 app.js

- 在 `data` 选项中添加属性 `currentItem`
- 在 `methods` 选项中添加函数 `toEdit(item)` 接收到点击的那个 `item` 后, 将它赋值给 `currentItem`, 那对应的任务项就会进入编辑状态 `this.currentItem = item`

```
1 var app = new Vue({
2   el: '#todoapp',
3   data: {
4     items, // 对象属性简写，等价于items: items
5     currentItem: null //上面不要少了逗号，接收当前点击的任务项
6   },
7
8   methods: {
9     // 进入编辑状态,当前点击的任务项item赋值currentItem，用于页面判断显示 .editing
10    toEdit (item) {
11      this.currentItem = item
12    },
13
14    //移除所有未完成任务项
15    removeCompleted () {
16      // 过滤出所有未完成的任务，重新赋值数组即可
17      this.items = this.items.filter(item => !item.completed)
18    },
19
20    . . . . .
```

4.11.3 编辑窗口显示原内容

在 `<input>` 上显示当前点击的任务内容，单向绑定输入框的值即可：`:value="item.content"`

```
1 <input class="edit" :value="item.content">
```

4.11.4 取消编辑

为 `<input>` 绑定 `Esc` 按键事件 `@keyup.esc=cancelEdit`，将 `currentItem` 值变为 `null`，就 `:class={editing: item === currentItem}` 不成立了，样式就没有了

- index.html

```
1 <input class="edit" :value="item.content" @keyup.esc="cancelEdit" />
```

- app.js

```
1 methods: {
2   //取消编辑
3   cancelEdit () {
4     // 移除样式
5     this.currentItem = null
6   },
7
8   // 进入编辑状态,当前点击的任务项item赋值currentItem，用于页面判断显示 .editing
```

```
9   toEdit (item) {
10     this.currentItem = item
11   },
12
13   //移除所有未完成任务项
14   removeCompleted () {
15     // 过滤出所有未完成的任务，重新赋值数组即可
16     this.items = this.items.filter(item => !item.completed)
17   }
18 }
```

4.11.5 保存数据

- 按 `Enter` 键 或 失去焦点时 保存改变数据，移除 `editing` 样式；
 - 分析：
 - 添加事件 `@keyup.enter=finishEdit(item, $event)` 与 `@blur="finishEdit(item, $event)"`
 - 通过 `$event` 变量获取当前输入框的值，使用 `.trim()` 去空格后进行判断是否为空，如果为空则清除这条任务，否则修改任务项；
 - 添加数据保存任务项中
 - 将 `currentItem` 值变为 `null`，移除 `editing` 样式
- index.html

```
1 <input class="edit" :value="item.content" @keyup.esc="cancelEdit"
2   @keyup.enter="finishEdit(item, index, $event)"
3   @blur="finishEdit(item, index, $event)">
```

- app.js

```
1 methods: {
2   //编辑完成
3   finishEdit (item, index, event) {
4     const content = event.target.value.trim();
5     // 1. 如果为空，则进行删除任务项
6     if (event.target.value.trim()){
7       //重用 removeItem 函数进行删除
8       this.removeItem(index)
9       return
10    }
11    // 2. 添加数据到任务项中
12    item.content = content
13    // 3. 移除 .editing 样式
14    this.currentItem = null
15  },
16
17  //取消编辑
18  cancelEdit () {
19    // 移除样式
```

```
20     this.currentItem = null
21   },
22
23   // 进入编辑状态,当前点击的任务项item赋值currentItem，用于页面判断显示 .editing
24   toEdit (item) {
25     this.currentItem = item
26   },
27
28   //移除所有未完成任务项
29   removeCompleted () {
30     // 过滤出所有未完成的任务，重新赋值数组即可
31     this.items = this.items.filter(item => !item.completed)
32   }
33
34   ...
```

注意：有时候你可能按 `Esc`、`Enter`、或者失去焦点时，发现没有被触发事件，是因为编辑框没有获取焦点，只有当编辑框有焦点时才可以触发事件。

4.11.6 获取焦点(自定义指令)

- 刷新页面后，通过自定义全局指令 `v-app-focus`，内容输入框自动获取焦点
- 当进入编辑状态后，通过自定义局部指令 `v-todo-focus`，编辑窗口自动获取焦点

```
1 (function (Vue) { // 表示依赖了全局的 Vue
2   //初始化任务
3   const items = []
4
5   //自定义全局指令，用于 增加输入框
6   //定义时不要在前面加v-, 引用指令时要加上v-
7   Vue.directive('app-focus', {
8     //聚集元素
9     inserted (el, binding) {
10      el.focus()
11    }
12  })
13
14  new Vue({
15    el: '#todoapp',
16    data: {
17      items, //ES6, 这是对象属性的简写方式， 等价于items: items
18      currentItem: null //代表的是点击的那个任务项
19    },
20
21    //自定义局部指令，用于编辑输入框
22    directives: {
23      //定义时不要在前面加v-, 引用指令时要加上v-
24      'todo-focus': {
25        update (el, binding) { // 每当指令的值更新后，会调用此函数
```



```
26         if (binding.value) {
27             el.focus()
28         }
29     },
30 },
31 },
32
33     . . . . . 省略
34
35 })(Vue);
36
37
```

- 在 index.html 中引入指令
- 增加输入框引用全局指令 `v-app-focus`
- 编辑框中引用局部指令 `v-todo-focus="item === currentItem"`

```
1  <!-- 添加任务， -->
2  <input @keyup.enter="addItem"
3      class="new-todo" placeholder="What needs to be done?" v-app-focus>
4
5  <!-- 编辑任务， -->
6  <input class="edit" :value="item.content" @keyup.esc="cancelEdit"
7      @keyup.enter="finishEdit(item, index, $event)"
8      @blur="finishEdit(item, index, $event)"
9      v-todo-focus="item === currentItem" >
10
```

4.11.7 完整源码

4.11.6.1 index.html

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1">
6      <title>Template • TodoMVC</title>
7      <link rel="stylesheet" href="node_modules/todomvc-common/base.css">
8      <link rel="stylesheet" href="node_modules/todomvc-app-css/index.css">
9      <!-- CSS overrides - remove if you don't need it -->
10     <link rel="stylesheet" href="css/app.css">
11 </head>
12 <body>
13     <section class="todoapp" id="todoapp">
14         <header class="header">
15             <h1>todos</h1>
16             <!--添加任务, keyup.enter 回车键监听-->
17             <input @keyup.enter="addItem"
```

```
18     class="new-todo" placeholder="What needs to be done?" v-app-focus>
19   </header>
20   <!-- This section should be hidden by default and shown when there are todos -->
21   <!--修改: items.length 值为 0 就是 false -->
22   <template v-if="items.length">
23     <section class="main">
24       <!-- 添加计算属性 toggleAll 双向绑定-->
25       <input v-model="toggleAll"
26         id="toggle-all" class="toggle-all" type="checkbox">
27       <label for="toggle-all">Mark all as complete</label>
28       <ul class="todo-list">
29         <!-- These are here just to show the structure of the list items -->
30         <!-- List items should get the class `editing` when editing and `completed` when marked as
completed -->
31         <!--
32         三种状态：未完成（没有样式）、已完成（.completed）、编辑中(.editing)
33         -->
34         <!-- 修改: 1、v-for 迭代; 2、:class {key为class样式名: value为获取的数据true或false} -->
35         <li v-for="(item, index) in items"
36           :class="{completed: item.completed, editing: item === currentItem}">
37           <div class="view">
38             <!-- 修改: 1、v-model 绑定状态值是否选中 -->
39             <input class="toggle" type="checkbox" v-model="item.completed">
40             <!-- 修改: 1、{{ content }} 显示内容 -->
41             <label @dblclick="toEdit(item)">{{ item.content }}</label>
42             <!--修改: 1、:value 绑定id删除 -->
43             <button class="destroy" :value="item.id" @click="removeItem(index)"></button>
44           </div>
45           <input class="edit" :value="item.content"
46             @keyup.esc="cancelEdit"
47             @keyup.enter="finishEdit(item, index, $event)"
48             @blur="finishEdit(item, index, $event)"
49             v-todo-focus="item === currentItem">
50         </li>
51       </ul>
52     </section>
53     <!-- This footer should hidden by default and shown when there are todos -->
54     <!--修改: items.length 值为 0 就是 false -->
55     <footer class="footer">
56       <!-- This should be `0 items left` by default -->
57       <span class="todo-count">
58         <strong>{{ remaining }}</strong> item{{ remaining === 1 ? '' : 's' }} left
59       </span>
60       <!-- Remove this if you don't implement routing -->
61       <ul class="filters">
62         <li>
63           <a class="selected" href="#/">All</a>
64         </li>
65         <li>
66           <a href="#/active">Active</a>
67         </li>
68         <li>
69           <a href="#/completed">Completed</a>
```

```
70         </li>
71     </ul>
72     <!-- Hidden if no completed items are left ↓ -->
73     <button @click="removeCompleted" v-show="items.length > remaining"
74         class="clear-completed">Clear completed</button>
75 </footer>
76
77 </template>
78 </section>
79 <footer class="info">
80     <p>Double-click to edit a todo</p>
81     <!-- Remove the below line ↓ -->
82     <p>Template by <a href="http://sindresorhus.com">Sindre Sorhus</a></p>
83     <!-- Change this out with your name and url ↓ -->
84     <p>Created by <a href="http://todomvc.com">you</a></p>
85     <p>Part of <a href="http://todomvc.com">TodoMVC</a></p>
86 </footer>
87 <!-- Scripts here. Don't remove ↓ -->
88 <script src="node_modules/todomvc-common/base.js"></script>
89 <!-- 在 app.js 前面-->
90 <script src="node_modules/vue/dist/vue.js" type="text/javascript"></script>
91 <script src="js/app.js"></script>
92 </body>
93 </html>
```

4.11.6.2 app.js

```
1 (function (Vue) { //表示依赖了全局的 Vue, 其实不加也可以, 只是更加明确点
2     //初始化任务
3     const items = []
4
5     //自定义全局指令, 用于 增加输入框
6     //定义时不要在前面加v-, 引用指令时要加上v-
7     Vue.directive('app-focus', {
8         //聚集元素
9         inserted (el, binding) {
10             el.focus()
11         }
12     })
13
14     var app = new Vue({
15         el: '#todoapp',
16         data: {
17             items, // 对象属性简写, 等价于items: items
18             currentItem: null //上面不要少了逗号, 接收当前点击的任务项
19         },
20
21         //自定义局部指令, 用于编辑输入框
22         directives: {
23             //定义时不要在前面加v-, 引用指令时要加上v-
24             'todo-focus': {
```

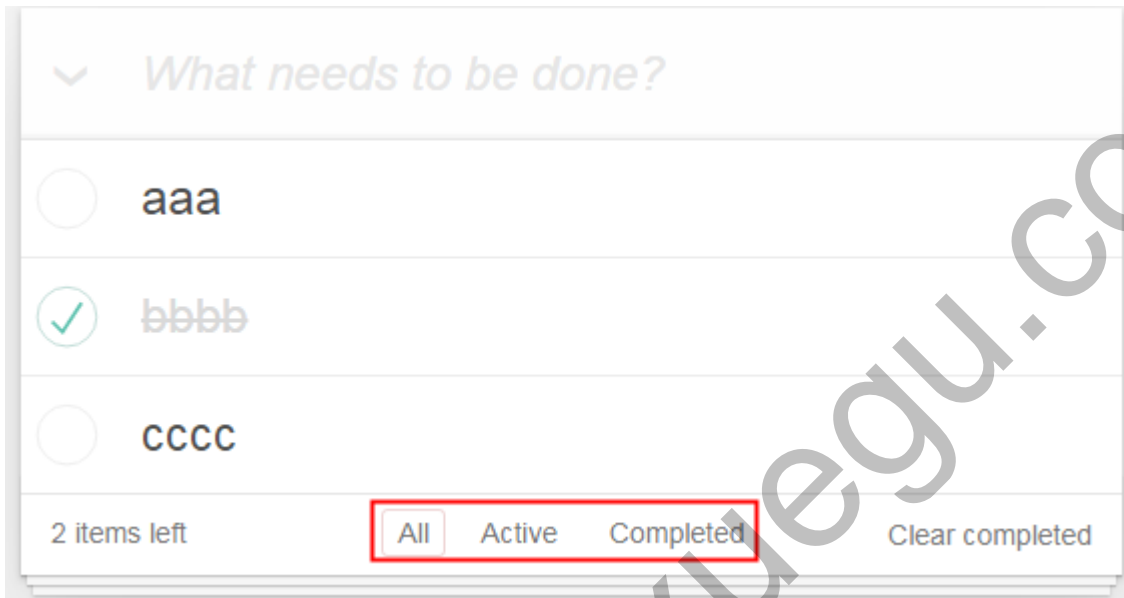
```
25     update (el, binding) { // 每当指令的值更新后，会调用此函数
26         if (binding.value) {
27             el.focus()
28         }
29     }
30 }
31 },
32
33 // 定义计算属性选项
34 computed: {
35     // 复选框计算属性
36     toggleAll: {
37         get () { // 等价于 get : function () {...}
38             console.log(this.remaining)
39             // 2. 当 this.remaining 发生变化后，会触发该方法运行
40             // 当所有未完成任务数为 0，表示全部完成，则返回 true 让复选框选中
41             // 反之就 false 不选中
42             return this.remaining === 0
43         },
44         set (newStatus) {
45             // console.log(newStatus)
46             // 1. 当点击 checkbox 复选框后状态变化后，就会触发该方法运行，
47             // 迭代出数组每个元素，把当前状态值赋给每个元素的 completed
48             this.items.forEach((item) => {
49                 item.completed = newStatus
50             })
51         }
52     },
53     // 过滤出所有未完成任务项
54     remaining () {
55         /*
56         return this.items.filter(function (item) {
57             return !item.completed
58         }).length
59         */
60         // ES6 箭头函数
61         return this.items.filter(item => !item.completed).length
62     }
63 }, // **注意** 后面不要少了逗号，
64
65 methods: {
66     // 编辑完成
67     finishEdit (item, index, event) {
68         const content = event.target.value.trim();
69         // 1. 如果为空，则进行删除任务项
70         if (event.target.value.trim()){
71             // 重用 removeItem 函数进行删除
72             this.removeItem(index)
73             return
74         }
75         // 2. 添加数据到任务项中
76         item.content = content
77
78         // 3. 移除 .editing 样式
```

```
78     this.currentItem = null
79   },
80
81   //取消编辑
82   cancelEdit () {
83     // 移除样式
84     this.currentItem = null
85   },
86
87   // 进入编辑状态,当前点击的任务项item赋值currentItem，用于页面判断显示 .editing
88   toEdit (item) {
89     this.currentItem = item
90   },
91   //移除所有未完成任务项
92   removeCompleted () {
93     // 过滤出所有未完成的任务，重新赋值数组即可
94     this.items = this.items.filter(item => !item.completed)
95   },
96
97   // 移除任务项
98   removeItem (index) {
99     // 移除索引为index的一条记录
100    this.items.splice(index, 1)
101  },
102
103  //增加任务项
104  addItem (event) { //对象属性函数简写，等价于addItem: function () {
105    console.log('addItem', event.target.value)
106    //1. 获取文本框输入的数据
107    const content = event.target.value.trim()
108    //2. 判断数据如果为空，则什么都不做
109    if (!content.length) {
110      return
111    }
112    //3.如果不为空，则添加到数组中
113    // 生成id值
114    const id = this.items.length + 1
115    // 添加到数组中
116    this.items.push({
117      id, //等价于 id:id
118      content,
119      completed: false
120    })
121    //4. 清空文本框内容
122    event.target.value = ""
123  }
124  }
125  })
126  })(Vue);
```

4.12 路由状态切换（过滤不同状态数据）

4.12.1 功能分析

- 根据点击的不同状态（All / Active / Completed），进行过滤出对应的任务，并进行样式的切换。



- 分析：
 - 在 data 中定义变量 filterStatus，用于接收变化的状态值
 - 通过 window.onhashchange 获取点击的路由 hash（# 开头的），来获取对应的那个状态值，并将状态值赋值给 filterStatus
 - 定义一个计算属性 filterItems 用于过滤出目标数据，用于感知 filterStatus 的状态值变化，当变化后，通过 switch-case + filter 过滤出目标数据。
 - 在 html 页面中，将 v-for 中之前的 items 数组替换为 filterItems 迭代出目标数据。
 - 将被点击状态的 样式切换为 .select，通过判断状态值实现，如：filterStatus === 'all'

4.12.2 功能实现

- 在 app.js 中 Vue 实例的 data 中定义变量 filterStatus，用于接收变化的状态值

声明一个变量 app 接收 Vue 实例对象，页面要使用到这个 app 变量

```
1 var app = new Vue({
2   el: '#todoapp',
3   data: {
4     items, // 对象属性简写，等价于 items: items
5     currentItem: null, // 上面不要少了逗号，接收当前点击的任务项
6     filterStatus: 'all' // 上面不要少了逗号，接收变化的状态值
7   },
8 })
```

- 在 app.js 通过 window.onhashchange 获取点击的路由 hash（# 开头的），来获取对应的那个状态值

注意：不是在 Vue 实例中定义，是在它同级结构下添加以下代码：

```
1 //当路由 hash 值改变后会自动调用此函数
2 window.onhashchange = function () {
3   console.log('hash改变了', window.location.hash)
4   // 1.获取点击的路由 hash, 当截取的 hash 不为空返回截取的, 为空时返回 'all'
5   const hash = window.location.hash.substr(2) || 'all'
6   console.log('hash', hash)
7   // 2. 状态一旦改变, 将 hash 赋值给 filterStatus
8   // 当计算属性 filterItems 感知到 filterStatus 变化后, 就会重新过滤
9   // 当 filterItems 重新过滤出目标数据后, 则自动同步更新到视图中
10  app.filterStatus = hash
11 }
12 // 第一次访问页面时,调用一次让状态生效
13 window.onhashchange()
```

3. 定义一个计算属性 `filterItems` 用于过滤出目标数据, 用于感知 `filterStatus` 的状态值变化, 当变化后, 通过 `switch-case` + `filter` 过滤出目标数据。

```
1 // 定义计算属性选项
2 computed: {
3   // 过滤出不同状态数据
4   filterItems () {
5     //this.filterStatus 作为条件, 变化后过滤不同数据
6     switch (this.filterStatus) {
7       case "active": // 过滤出未完成的数据
8         return this.items.filter( item => !item.completed)
9         break
10      case "completed": // 过滤出已完成的数据
11        return this.items.filter( item => item.completed)
12        break
13      default: // 其他, 返回所有数据
14        return this.items
15      }
16    },
```

4. 在index.html页面中, 将 `v-for` 中之前的 `items` 数组替换为 `filterItems` 迭代出目标数据。

```
1 <!--将之前 items 替换为 filterItems -->
2 <li v-for="(item, index) in filterItems"
3   :class="{completed: item.completed, editing: item === currentItem}">
4   ... 省略
5 </li>
```

5. 在index.html 中, 将被点击状态的 `` 样式切换为 `.select`, 通过判断状态值实现

```
1 <ul class="filters">
2   <li>
3     <a :class="{selected: filterStatus === 'all'}" href="#" /> All </a>
4   </li>
5   <li>
6     <a :class="{selected: filterStatus === 'active'}" href="#/active"> Active </a>
7   </li>
8   <li>
9     <a :class="{selected: filterStatus === 'completed'}" href="#/completed"> Completed </a>
10  </li>
11 </ul>
```

4.12.3 完整源码

4.12.2.1 index.html

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <title>Template • TodoMVC</title>
7     <link rel="stylesheet" href="node_modules/todomvc-common/base.css">
8     <link rel="stylesheet" href="node_modules/todomvc-app-css/index.css">
9     <!-- CSS overrides - remove if you don't need it -->
10    <link rel="stylesheet" href="css/app.css">
11  </head>
12  <body>
13    <section class="todoapp" id="todoapp">
14      <header class="header">
15        <h1>todos</h1>
16        <!-- 添加任务, keyup.enter 回车键监听 -->
17        <input @keyup.enter="addItem"
18          class="new-todo" placeholder="What needs to be done?" v-app-focus>
19      </header>
20      <!-- This section should be hidden by default and shown when there are todos -->
21      <!-- 修改: items.length 值为 0 就是 false -->
22      <template v-if="items.length">
23        <section class="main">
24          <!-- 添加计算属性 toggleAll 双向绑定 -->
25          <input v-model="toggleAll"
26            id="toggle-all" class="toggle-all" type="checkbox">
27          <label for="toggle-all"> Mark all as complete </label>
28          <ul class="todo-list">
29            <!-- These are here just to show the structure of the list items -->
30            <!-- List items should get the class `editing` when editing and `completed` when marked as
31              completed -->
32            <!--
33            三种状态：未完成（没有样式）、已完成（.completed）、编辑中(.editing)
34            -->
```



```
34      <!-- 修改: 1、v-for 迭代; 2、:class {key为class样式名: value为获取的数据true或false} -->
35      <li v-for="(item, index) in filterItems"
36          :class="{completed: item.completed, editing: item === currentItem}">
37          <div class="view">
38              <!-- 修改: 1、v-model 绑定状态值是否选中 -->
39              <input class="toggle" type="checkbox" v-model="item.completed">
40              <!-- 修改: 1、{{ content }} 显示内容 -->
41              <label @dblclick="toEdit(item)">{{ item.content }}</label>
42              <!--修改: 1、:value 绑定id删除 -->
43              <button class="destroy" :value="item.id" @click="removeItem(index)"></button>
44          </div>
45          <input class="edit" :value="item.content" @keyup.esc="cancelEdit"
46              @keyup.enter="finishEdit(item, index, $event)"
47              @blur="finishEdit(item, index, $event)"
48              v-todo-focus="item === currentItem" >
49      </li>
50  </ul>
51 </section>
52 <!-- This footer should hidden by default and shown when there are todos -->
53 <!--修改: items.length 值为 0 就是 false -->
54 <footer class="footer">
55     <!-- This should be `0 items left` by default -->
56     <span class="todo-count">
57         <strong>{{ remaining }}</strong> item{{ remaining === 1 ? '' : 's' }} left
58     </span>
59     <!-- Remove this if you don't implement routing -->
60     <ul class="filters">
61         <li>
62             <a :class="{selected: filterStatus === 'all'}" href="#">All</a>
63         </li>
64         <li>
65             <a :class="{selected: filterStatus === 'active'}" href="#/active">Active</a>
66         </li>
67         <li>
68             <a :class="{selected: filterStatus === 'completed'}" href="#/completed">Completed</a>
69         </li>
70     </ul>
71     <!-- Hidden if no completed items are left ↓ -->
72     <button @click="removeCompleted" v-show="items.length > remaining"
73         class="clear-completed">Clear completed</button>
74 </footer>
75
76 </template>
77 </section>
78 <footer class="info">
79     <p>Double-click to edit a todo</p>
80     <!-- Remove the below line ↓ -->
81     <p>Template by <a href="http://sindresorhus.com">Sindre Sorhus</a></p>
82     <!-- Change this out with your name and url ↓ -->
83     <p>Created by <a href="http://todomvc.com">you</a></p>
84     <p>Part of <a href="http://todomvc.com">TodoMVC</a></p>
85 </footer>
86 <!-- Scripts here. Don't remove ↓ -->
```

```
87 <script src="node_modules/todomvc-common/base.js"></script>
88 <!-- 在 app.js 前面-->
89 <script src="node_modules/vue/dist/vue.js" type="text/javascript"></script>
90 <script src="js/app.js"></script>
91
92 </body>
93 </html>
```

4.12.2.2 app.js

```
1 (function (Vue) { //表示依赖了全局的 Vue, 其实不加也可以, 只是更加明确点
2
3   //初始化任务
4   const items = []
5
6   //注册全局指令
7   //指令名不要加上v-, 在引用这个指令时才需要加上 v-
8   Vue.directive('app-focus', {
9     inserted (el, binding) {
10       //聚集元素
11       el.focus()
12     }
13   })
14
15   var app = new Vue({
16     el: '#todoapp',
17     data: {
18       items, // 对象属性简写, 等价于 items: items
19       currentItem: null, //上面不要少了逗号, 接收当前点击的任务项
20       filterStatus: 'all' // 上面不要少了逗号, 接收变化的状态值
21     },
22
23     //自定义局部指令
24     directives: {
25       'todo-focus': { //注意指令名称
26         update (el, binding) {
27           //只有双击的那个元素才会获取焦点
28           if (binding.value) {
29             el.focus()
30           }
31         }
32       }
33     },
34
35     // 定义计算属性选项
36     computed: {
37       // 过滤出不同状态数据
38       filterItems () {
39         //this.filterStatus 作为条件, 变化后过滤不同数据
40         switch (this.filterStatus) {
41           case "active": // 过滤出未完成的数据
```

```
42         return this.items.filter( item => item.completed)
43         break
44     case "completed": // 过滤出已完成的数据
45         return this.items.filter( item => item.completed)
46         break
47     default: // 其他，返回所有数据
48         return this.items
49     }
50 },
51 // 复选框计算属性
52 toggleAll : {
53     get () { //等价于 get : function () {...}
54         console.log(this.remaining)
55         //2. 当 this.remaining 发生变化后，会触发该方法运行
56         // 当所有未完成任务数为 0，表示全部完成, 则返回 true 让复选框选中
57         //反之就 false 不选中
58         return this.remaining === 0
59     },
60     set (newStatus) {
61         // console.log(newStatus)
62         //1. 当点击 checkbox 复选框后状态变化后，就会触发该方法运行,
63         // 迭代出数组每个元素,把当前状态值赋给每个元素的 completed
64         this.items.forEach((item) => {
65             item.completed = newStatus
66         })
67     }
68 },
69 // 过滤出所有未完成的任务项
70 remaining () {
71     /*
72     return this.items.filter(function (item) {
73         return !item.completed
74     }).length
75     */
76     //ES6 箭头函数
77     return this.items.filter(item => !item.completed).length
78 }
79 }, // **注意** 后面不要少了逗号 ,
80
81 methods: {
82     //编辑完成
83     finishEdit (item, index, event) {
84         const content = event.target.value.trim();
85         // 1. 如果为空, 则进行删除任务项
86         if (!event.target.value.trim()){
87             //重用 removeItem 函数进行删除
88             this.removeItem(index)
89             return
90         }
91         // 2. 添加数据到任务项中
92         item.content = content
93         // 3. 移除 .editing 样式
94
95         this.currentItem = null
```

```
95     },
96
97     //取消编辑
98     cancelEdit () {
99         // 移除样式
100         this.currentItem = null
101     },
102
103     // 进入编辑状态,当前点击的任务项item赋值currentItem，用于页面判断显示 .editing
104     toEdit (item) {
105         this.currentItem = item
106     },
107     //移除所有未完成任务项
108     removeCompleted () {
109         // 过滤出所有未完成任务，重新赋值数组即可
110         this.items = this.items.filter(item => !item.completed)
111     },
112
113     // 移除任务项
114     removeItem (index) {
115         // 移除索引为index的一条记录
116         this.items.splice(index, 1)
117     },
118
119     //增加任务项
120     addItem (event) { //对象属性函数简写，等价于addItem: function () {
121         console.log('addItem', event.target.value)
122         //1. 获取文本框输入的数据
123         const content = event.target.value.trim()
124         //2. 判断数据如果为空，则什么都不做
125         if (!content.length) {
126             return
127         }
128         //3.如果不为空，则添加到数组中
129         // 生成id值
130         const id = this.items.length + 1
131         // 添加到数组中
132         this.items.push({
133             id, //等价于 id:id
134             content,
135             completed: false
136         })
137         //4. 清空文本框内容
138         event.target.value = ""
139     }
140 }
141 })
142
143 //当路由 hash 值改变后会自动调用此函数
144 window.onhashchange = function () {
145     console.log('hash改变了' + window.location.hash)
146     // 1.获取点击的路由 hash，当截取的 hash 不为空返回截取的，为空时返回 'all'
147     var hash = window.location.hash.substr(2) || 'all'
```

```
148
149 // 2. 状态一旦改变，将 hash 赋值给 filterStatus
150 // 当计算属性 filterItems 感知到 filterStatus 变化后，就会重新过滤
151 // 当 filterItems 重新过滤出目标数据后，则自动同步更新到视图中
152 app.filterStatus = hash
153 }
154 // 第一次访问页面时，调用一次让状态生效
155 window.onhashchange()
156
157 })(Vue);
```

4.13 数据持久化

4.13.1 功能分析

- 将所有任务项数据持久化到 `localStorage` 中，它主要是用于本地存储数据。`localStorage` 中一般浏览器支持的是 5M 大小，这个在不同的浏览器中 `localStorage` 会有所不同。
- 分析：可以使用 Vue 中的 `watch` 监听器，监听任务数组 `items` 一旦有改变，则使用 `window.localStorage` 将它重新保存到 `localStorage`

4.13.2 功能实现

使用 `window.localStorage` 实例进行保存数据与获取数据

- 定义 `itemStorage` 数据存储对象，里面自定义 `fetch` 获取本地数据，`save` 存数据到本地。
- 修改 Vue 实例中 `data` 选项的 `items` 属性，通过 `itemStorage.fetch()` 方法初始化数据
- Vue 实例中增加一个 `watch` 选项，用于监听 `items` 的变化，一旦变化通过 `itemStorage.save()` 重新保存数据到本地

注意：因为 `items` 数组内部是对象，当对象的值发生变化后要被监听到，在选项参数中使用 `deep: true`

参考：<https://cn.vuejs.org/v2/api/#vm-watch>

```
1 (function (Vue) { //表示依赖了全局的 Vue, 其实不加也可以，只是更加明确点
2
3   var STORAGE_KEY = 'items-vuejs';
4
5   // 本地存储数据对象
6   const itemStorage = {
7     fetch: function () { // 获取本地数据
8       return JSON.parse(localStorage.getItem(STORAGE_KEY) || '[]');
9     },
10    save: function (items) { // 保存数据到本地
11      localStorage.setItem(STORAGE_KEY, JSON.stringify(items));
12    }
13  }
14
15  //初始化任务
```

```
16  const items = []
17  //注册全局指令
18  //指令名不要加上v-, 在引用这个指令时才需要加上 v-
19  Vue.directive('app-focus', {
20    inserted (el, binding) {
21      //聚集元素
22      el.focus()
23    }
24  })
25
26  var app = new Vue({
27    el: '#todoapp',
28
29    data: {
30      //items, // 对象属性简写，等价于items: items
31      items: itemStorage.fetch(), //获取本地数据进行初始化
32      currentItem: null, //上面不要少了逗号，接收当前点击的任务项
33      filterStatus: 'all' // 上面不要少了逗号，接收变化的状态值
34    },
35
36    // 监听器
37    watch: {
38      // 如果 items 发生改变，这个函数就会运行
39      items: {
40        deep: true, // 发现对象内部值的变化, 要在选项参数中指定 deep: true。
41        handler: function(newItems, oldItems) {
42          //本地进行存储
43          itemStorage.save(newItems)
44        }
45      }
46    },
47
48    //自定义局部指令
49    directives: {
50      'todo-focus': { //注意指令名称
51        update (el, binding) {
52          //只有双击的那个元素才会获取焦点
53          if(binding.value) {
54            el.focus()
55          }
56        }
57      }
58    },
59
60    // 定义计算属性选项
61    computed: {
62      // 过滤出不同状态数据
63      filterItems () {
64        //this.filterStatus 作为条件，变化后过滤不同数据
65        switch (this.filterStatus) {
66          case "active": // 过滤出未完成的数据
67            return this.items.filter( item => !item.completed)
68
69          break
```

```
69     case "completed": // 过滤出已完成的数据
70         return this.items.filter( item => item.completed)
71         break
72     default: // 其他，返回所有数据
73         return this.items
74     }
75 },
76 // 复选框计算属性
77 toggleAll : {
78     get () { //等价于 get : function () {...}
79         console.log(this.remaining)
80         //2. 当 this.remaining 发生变化后，会触发该方法运行
81         // 当所有未完成任务数为 0，表示全部完成，则返回 true 让复选框选中
82         //反之就 false 不选中
83         return this.remaining === 0
84     },
85     set (newStatus) {
86         // console.log(newStatus)
87         //1. 当点击 checkbox 复选框后状态变化后，就会触发该方法运行，
88         // 迭代出数组每个元素，把当前状态值赋给每个元素的 completed
89         this.items.forEach((item) => {
90             item.completed = newStatus
91         })
92     }
93 },
94 // 过滤出所有未完成的任务项
95 remaining () {
96     /*
97     return this.items.filter(function (item) {
98         return !item.completed
99     }).length
100     */
101     //ES6 箭头函数
102     return this.items.filter(item => !item.completed).length
103 }
104 }, // **注意** 后面不要少了逗号 ,
105
106 methods: {
107     //编辑完成
108     finishEdit (item, index, event) {
109         const content = event.target.value.trim();
110         // 1. 如果为空，则进行删除任务项
111         if (event.target.value.trim()){
112             //重用 removeItem 函数进行删除
113             this.removeItem(index)
114             return
115         }
116         // 2. 添加数据到任务项中
117         item.content = content
118         // 3. 移除 .editing 样式
119         this.currentItem = null
120     },
121
```

```
122 //取消编辑
123 cancelEdit () {
124     // 移除样式
125     this.currentItem = null
126 },
127
128 // 进入编辑状态,当前点击的任务项item赋值currentItem，用于页面判断显示 .editing
129 toEdit (item) {
130     this.currentItem = item
131 },
132 //移除所有未完成任务项
133 removeCompleted () {
134     // 过滤出所有未完成的任务，重新赋值数组即可
135     this.items = this.items.filter(item => !item.completed)
136 },
137
138 // 移除任务项
139 removeItem (index) {
140     // 移除索引为index的一条记录
141     this.items.splice(index, 1)
142 },
143
144 //增加任务项
145 addItem (event) { //对象属性函数简写，等价于addItem: function () {
146     console.log('addItem', event.target.value)
147     //1. 获取文本框输入的数据
148     const content = event.target.value.trim()
149     //2. 判断数据如果为空，则什么都不做
150     if (!content.length) {
151         return
152     }
153     //3.如果不为空，则添加到数组中
154     // 生成id值
155     const id = this.items.length + 1
156     // 添加到数组中
157     this.items.push({
158         id, //等价于 id:id
159         content,
160         completed: false
161     })
162     //4. 清空文本框内容
163     event.target.value = ""
164 }
165 }
166 })
167
168 //当路由 hash 值改变后会自动调用此函数
169 window.onhashchange = function () {
170     console.log('hash改变了' + window.location.hash)
171     // 1.获取点击的路由 hash，当截取的 hash 不为空返回截取的，为空时返回 'all'
172     var hash = window.location.hash.substr(2) || 'all'
173
174     // 2. 状态一旦改变，将 hash 赋值给 filterStatus
```



```
175 // 当计算属性 filterItems 感知到 filterStatus 变化后，就会重新过滤
176 // 当 filterItems 重新过滤出目标数据后，则自动同步更新到视图中
177 app.filterStatus = hash
178 }
179 // 第一次访问页面时,调用一次让状态生效
180 window.onhashchange()
181
182 })(Vue);
```

第五章 Vue 过滤器和插件

5.1 过滤器

5.1.1 什么是过滤器

- 过滤器对将要显示的文本，先进行特定格式化处理，然后再进行显示
- 注意：过滤器并没有改变原本的数据，只是产生新的对应的数据

5.1.2 使用方式

1. 定义过滤器：

- 全局过滤器：

```
1 Vue.filter(过滤器名称, function (value1[,value2, ...]) {
2   // 数据处理逻辑
3 })
```

- 局部过滤器：在Vue实例中使用 `filter` 选项，当前实例范围内可用

```
1 new Vue({
2
3   filters: {
4     过滤器名称: function (value1[,value2,...]) {
5       // 数据处理逻辑
6     }
7   }
8
9 })
```

2. 过滤器可以用在两个地方：双花括号 `{{}}` 和 `v-bind` 表达式

```
1 <!-- 在双花括号中 -->
2 <div>{{数据属性名称 | 过滤器名称}}</div>
3 <div>{{数据属性名称 | 过滤器名称(参数值)}}</div>
4
5 <!-- 在 `v-bind` 中 -->
6 <div v-bind:id="数据属性名称 | 过滤器名称"></div>
7 <div v-bind:id="数据属性名称 | 过滤器名称(参数值)"></div>
```

5.1.3 案例演示

- 需求：

1. 实现过滤敏感字符，如当文本中有 tmd、sb 都将进行过滤掉
2. 过滤器传入多个参数，实现求和运算

- 实现：

1. 新建 vue-04-过滤器和插件 目录，安装 vue.js 模块
2. 在 vue-04-过滤器和插件 目录下创建 01-过滤器.html

```
1 <body>
2
3 <div id="app">
4   <h3>过滤器接收多个参数：</h3>
5   <p>{{content | contentFilter}}</p>
6   <input type="text" :value="content | sensitive">
7   <h3>过滤器接收多个参数：</h3>
8   <p>{{vueScore | add(javaScore, pythonScore)}}</p>
9 </div>
10
11 <script src="node_modules/vue/dist/vue.js" type="text/javascript"></script>
12 <script type="text/javascript">
13
14   /*定义全局过滤器: 过滤敏感数据*/
15   /*Vue.filter('contentFilter',function (value) {
16     if (!value) return ''
17     return value.toUpperCase().replace('TMD','***').replace('SB','***')
18   })*/*
19
20   new Vue({
21     el: '#app',
22     data: {
23       content: '小伙子，TMD就是个SB',
24       vueScore: 80,
25       javaScore: 95,
26       pythonScore: 90
27     },
28     //局部过滤器
29     filters: { //不要少了 s
30       contentFilter (value) { //value是调用时 | 左边的那个属性值
31         if (!value) return ''
```

```
32     return value.toUpperCase().replace('TMD','***').replace('SB','***')
33   },
34   add (num1, num2, num3) { //num1是调用时 | 左边的那个属性值
35     return num1 + num2 + num3
36   }
37 }
38 })
39 </script>
40 </body>
```

5.2 自定义插件

5.2.1 插件的作用

1. 插件通常会为 Vue 添加全局功能，一般是添加全局方法/全局指令/过滤器等
2. Vue 插件有一个公开方法 `install`，通过 `install` 方法给 Vue 添加全局功能
3. 通过全局方法 `Vue.use()` 使用插件，它需要在你调用 `new Vue()` 启动应用之前完成。

5.2.2 案例演示

1. 开发插件, 在 `vue-04-过滤器和插件` 目录下创建 `js` 目录, 在 `js` 目录建一个 `plugins.js` 文件

```
1 (function(){
2   // 声明 MyPlugin 插件对象
3   const MyPlugin = {}
4   MyPlugin.install = function (Vue, options) {
5     // 1. 添加全局方法
6     Vue.myGlobalMethod = function () {
7       alert('MyPlugin插件: 全局方法生效')
8     }
9
10    // 2. 添加全局指令
11    Vue.directive('my-directive', {
12      inserted: function (el, binding) {
13        el.innerHTML = "MyPlugin插件 my-directive:" + binding.value
14      }
15    })
16
17    // 3. 添加实例方法
18    Vue.prototype.$myMethod = function (methodOption) {
19      alert('Vue 实例方法生效 : ' + methodOption)
20    }
21  }
22  // 将插件添加到 window 对象中
23  window.MyPlugin = MyPlugin
24
25 })() // 不要少了括号(), 让它立即执行
```

2. 使用插件, 在 vue-04-过滤器和插件 目录下创建 02-自定义插件.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>14-自定义插件</title>
6 </head>
7 <body>
8   <div id="app">
9     <!-- 引用指令时不要少了 v- -->
10    <span v-my-directive="content"></span>
11  </div>
12  <script src="../js/vue.js" type="text/javascript"></script>
13  <script src="js/plugins.js" type="text/javascript"></script>
14  <script type="text/javascript">
15    // 1.引入自定义插件 MyPlugin
16    // 如果报错:Uncaught ReferenceError: MyPlugin is not defined
17    // 解决方法: 查看 plugins.js 是否引入, 如果引入还是报错, 检查 js 语法, 特别是最后一行不要少了括号 ()
18    Vue.use(MyPlugin)
19
20    // 2. 创建 Vue 实例, 模板中使用引用全局指令 v-my-directive="content"
21    var vm = new Vue({
22      el: '#app',
23      data: {
24        content: 'hello'
25      }
26    })
27
28    // 3. 调用自定义的全局方法, 所以是 Vue 调用, 不是 vm
29    Vue.myGlobalMethod()
30
31    // 4. 调用 Vue 实例方法, 所以是 vm 调用, 不是 Vue
32    vm.$myMethod('mengxuegu')
33  </script>
34 </body>
35 </body>
36 </html>
```

3. 访问页面的效果:

1. alert 弹出: MyPlugin插件: 全局方法生效

localhost:63342 显示

Vue 实例方法生效 : mengxuegu

确定

2. alert 弹出: Vue 实例方法生效 : mengxuegu

localhost:63342 显示

MyPlugin插件: 全局方法生效

确定

3. 页面渲染出: MyPlugin插件 my-directive:hello

MyPlugin插件 my-directive:hello

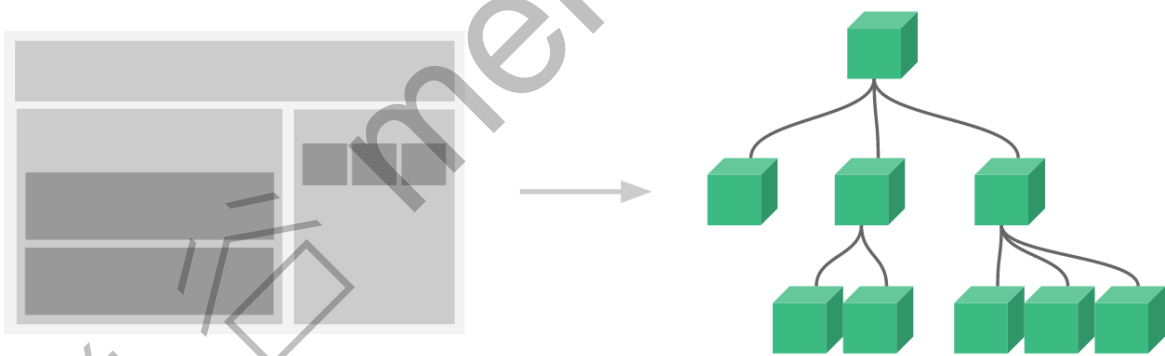
第六章 Vue 组件化开发

6.1 组件的概念

组件 (component) 是 Vue.js 最强大的功能之一。

Vue 中的组件化开发就是把网页的重复代码抽取出来，封装成一个个可复用的视图组件，然后将这些视图组件拼接到一块就构成了一个完整的系统。这种方式非常灵活，可以极大的提高我们开发和维护的效率。

通常一套系统会以一棵嵌套的组件树的形式来组织：



例如：项目可能会有头部、底部、页侧边栏、内容区等组件，每个组件又包含了其它的像导航链接、博文之类的组件。

- 组件就是对局部视图的封装，每个组件包含了：
 - HTML 结构
 - CSS 样式
 - JavaScript 行为
 - data 数据
 - methods 行为
- 提高开发效率，增强可维护性，更好的去解决软件上的高耦合、低内聚、无重用的3大代码问题

- Vue 中的组件思想借鉴于 React
- 目前主流的前端框架：Angular、React、Vue 都是组件化开发思想

6.2 组件的基本使用

为了能在模板中使用，这些组件必须先注册以便 Vue 能够识别。

有两种组件的注册类型：**全局注册**和**局部注册**

6.2.1 全局注册

6.2.1.1 介绍

一般把网页中特殊的公共部分注册为全局组件：轮播图、分页、通用导航栏

- 全局注册之后，可以在任何新创建的 Vue 实例 (new Vue) 的模板中使用
- 简单格式：

```
1 Vue.component('组件名',{
2   template: '定义组件模板',
3   data: function(){ //data 选项在组件中必须是一个函数
4     return {}
5   }
6   //其他选项：methods
7 })
```

说明：

- 组件名：
 - 可使用驼峰(camelCase)或者横线分隔(kebab-case)命名方式
 - 但 DOM 中只能使用**横线分隔**方式进行引用组件
 - 官方强烈推荐组件名字母全小写且必须包含一个连字符
- template：定义组件的视图模板
- data：在组件中必须是一个函数

6.2.1.2 示例

1. 创建 vue-05-组件化 目录，安装 vue 模块，创建 01-全局注册.html

```
1 <body>
2   <div id="app">
3     <!-- 通过组件名直接使用, 不能使用驼峰形式 -->
4     <component-a></component-a>
5   </div>
6   <script src="node_modules/vue/dist/vue.js" type="text/javascript"></script>
7   <script type="text/javascript">
8     /*
```

```
9      1. 全局组件注册:
10      它们注册之后可以用在任何新创建的 Vue 根实例 (new Vue) 的模板中
11
12      参数1：组件名
13      1. 可使用驼峰(camelCase)或者横线分隔(kebab-case)命名方式
14      2. DOM 中只能使用横线分隔方式进行引用组件
15      */
16      Vue.component('component-a', {
17        // template 选项指定此组件的模板代码
18        template: '<div><h1>头部组件 - {{ name }}</h1></div>',
19        // data 必须是函数
20        data: function () {
21          return {
22            name: '全局组件'
23          }
24        }
25      })
26
27      new Vue({
28        el: '#app'
29      })
30    </script>
31  </body>
```

6.2.2 局部注册（子组件）

6.2.2.1 介绍

一般把一些非通用部分注册为局部组件，一般只适用于当前项目的。

- 格式：

```
1  1. JS 对象来定义组件:
2  var ComponentA = { data: function() {}, template: '组件模板A' }
3  var ComponentA = { data: function() {}, template: '组件模板A' }
4
5  2. 在Vue实例中的 components 选项中引用组件：
6  new Vue({
7    el: '#app',
8    data: {},
9    components: { // 组件选项
10      'component-a': ComponentA // key：组件名，value: 引用的组件对象
11      'component-b': ComponentB
12    }
13  })
```

6.2.2.2 示例

```
1 <body>
2   <div id="app">
3     <!-- 通过组件名直接使用, 不能使用驼峰形式 -->
4     <component-b ></component-b>
5   </div>
6   <script src="node_modules/vue/dist/vue.js" type="text/javascript"></script>
7   <script type="text/javascript">
8     // 定义局部组件对象
9     var ComponentB = {
10       template: '<div> 这是 : {{ name }}</div>',
11       data: function () {
12         return {
13           name: '局部组件'
14         }
15       }
16     }
17
18     new Vue({
19       el: '#app',
20       components: { // 局部组件
21         'component-b': ComponentB
22       }
23     })
24   </script>
25 </body>
```

6.2.3 总结

- 组件是可复用的 Vue 实例，不需要手动实例化
- 与 `new Vue` 接收相同的选项，例如 `data`、`computed`、`watch`、`methods` 等
- 组件的 `data` 选项必须是一个函数

6.3 多个组件示例

- 效果演示

头部组件

- 客户管理
- 帐单管理
- 供应商管理

底部组件

- 可将 组件注册 抽取在一个一个的 js 文件中方便管理
- 在 `vue-05-组件化` 目录下创建 `component` 目录存放组件: `Header.js` `Main.js` `Footer.js`

```
1 Header.js 文件内容
2 Vue.component('app-header',{
3   // template 选项指定此组件的模板代码
4   template: '<div class="header"><h1>头部组件</h1></div>'
5 })
6
7 Main.js 文件内容
8 Vue.component('app-main',{
9   // template 选项指定此组件的模板代码
10  template: '<div class="main"><ul><li>客户管理</li><li>帐单管理</li><li>供应商管理</li></ul><h3></h3></div>'
11 })
12
13 Footer.js 文件内容
14 Vue.component('app-footer',{
15   // template 选项指定此组件的模板代码
16   template: '<div class="footer"><h1>底部组件</h1></div>'
17 })
```

- 页面引入组件 js 文件后, 进行使用

```
1 <body>
2   <div id="app">
3     <!-- 通过组件名直接使用 -->
4     <app-header></app-header>
5     <app-main></app-main>
6     <app-footer></app-footer>
```

```

7   </div>
8   <script src="node_modules/vue/dist/vue.js" type="text/javascript"></script>
9   <!-- 要在 vue.js 下面引入组件 -->
10  <script src="component/Footer.js" type="text/javascript"></script>
11  <script src="component/Header.js" type="text/javascript"></script>
12  <script src="component/Main.js" type="text/javascript"></script>
13  <script type="text/javascript">
14    new Vue({
15      el: '#app'
16    })
17  </script>
18 </body>

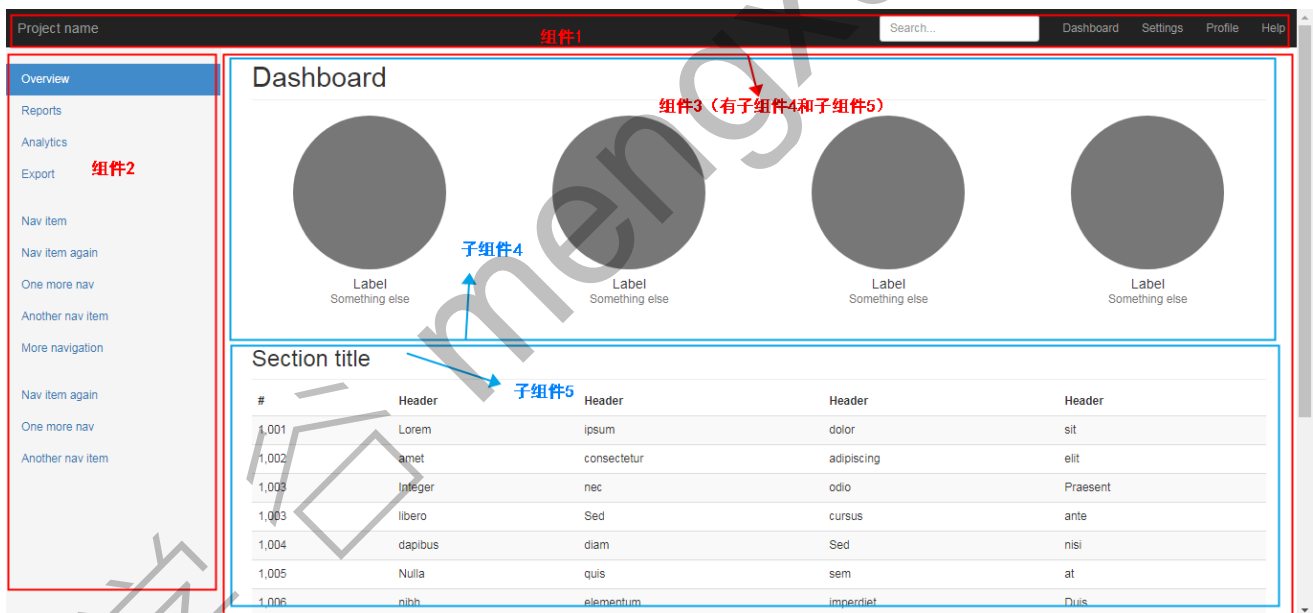
```

6.4 Bootstrap 首页组件化

6.4.1 分析首页

- 分析首页可拆分为多少个组件

html页面位于：01-配套源码\bootstrap



共拆分为5个组件

```

17 <body>
18 <!--头部导航区域--> AppNavbar.js
19 <nav class="navbar navbar-inverse navbar-fixed-top">
43
44 <!--核心区域:分左右两边-->
45 <div class="container-fluid">
46 <div class="row">
47 AppLeaf.js
48 <!--左边菜单栏区域-->
49 <div class="col-sm-3 col-md-2 sidebar">
69
70 <!--右边主页面区域:分上下两个区域-->
71 <div class="col-sm-9 col-sm-offset-3 col-md-10
    col-md-offset-2 main">
72
73 <!--右边上半区域-->
74 <h1 class="page-header">Dashboard</h1>
75 <div class="row placeholders"> Dashboard.js
76
77 <!--右边下半区域-->
78 <h2 class="sub-header">Section title</h2>
79 <div class="table-responsive"> HomeList.js
80
226 </div>
227 </div>
228 </div>
229 </body>

```

6.4.2 头部导航组件注册

1. 在 vue-05-组件化 目录下创建 03-bootstrap 目录
2. 将网盘中 01-配套源码\bootstrap 目录的所有文件复制到 03-bootstrap 目录下
3. 在 index.html 的 <body> 标签下添加一个 <div id="app"> Vue管理入口
4. 在 index.html 引入 vue.js 和 创建 Vue 实例

```

1 <script src="../node_modules/vue/dist/vue.js"></script>
2 <script>
3
4   new Vue({
5     el: '#app'
6   })
7
8 </script>

```

5. 注册和引用头部导航组件 AppNavbar

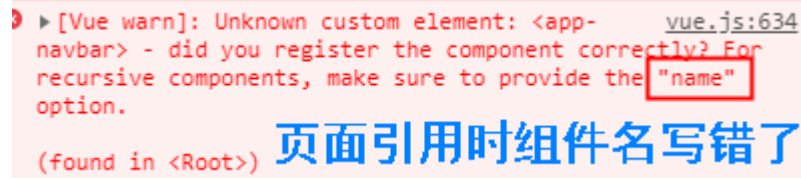
- 左上角显示 梦学谷 ,通过 data 选项函数指定 message 属性显示
- 输入框中失去焦点后 alert('失去焦点'), 通过 methods 选项
- 在 头部导航区域采用组件形式 <navbar></navbar>

```
1 <body>
2   <div id="app">
3     <!--头部导航区域-->
4     <navbar></navbar>
5
6   </div>
7   <script src="../../node_modules/vue/dist/vue.js"></script>
8   <script>
9     //组件对象定义
10    const AppNavbar = {
11      template: `<nav class="navbar navbar-inverse navbar-fixed-top">
12        <div class="container-fluid">
13          <div class="navbar-header">
14            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-
15target="#navbar" aria-expanded="false" aria-controls="navbar">
16              <span class="sr-only">Toggle navigation</span>
17              <span class="icon-bar"></span>
18              <span class="icon-bar"></span>
19              <span class="icon-bar"></span>
20            </button>
21            <a class="navbar-brand" href="#">{{ projectName }}</a>
22          </div>
23          <div id="navbar" class="navbar-collapse collapse">
24            <ul class="nav navbar-nav navbar-right">
25              <li><a href="#">Dashboard</a></li>
26              <li><a href="#">Settings</a></li>
27              <li><a href="#">Profile</a></li>
28              <li><a href="#">Help</a></li>
29            </ul>
30            <form class="navbar-form navbar-right">
31              <input type="text" class="form-control" placeholder="Search..." @blur="search">
32            </form>
33          </div>
34        </nav>`
35      data: function() {
36        return {
37          projectName: '梦学谷'
38        }
39      },
40      methods: {
41        search() {
42          alert('失去焦点')
43        }
44      }
45    }
46
47    new Vue({
48      el: '#app',
49      components: {
50        'app-navbar': AppNavbar
51      }
52    })
```

```
53 </script>
54 </body>
```

6. 浏览器访问 index.html

- 如果如果控制台报如下错,则 `<app-navbar></app-navbar>` 标签名与组件名不相同



7. 抽取定义的 AppNavbar 对象到 AppNavbar.js 中

- 把 `const AppNavbar = {.....}` 部分剪切进 `AppNavbar.js` 文件中, `AppNavbar` 对象添加到 `window` 域

```
1 ;(function () {
2   //粘贴到这里
3   window.AppNavbar = {
4     //
5   }
6 })()
```

- 将 `template` 提取出来

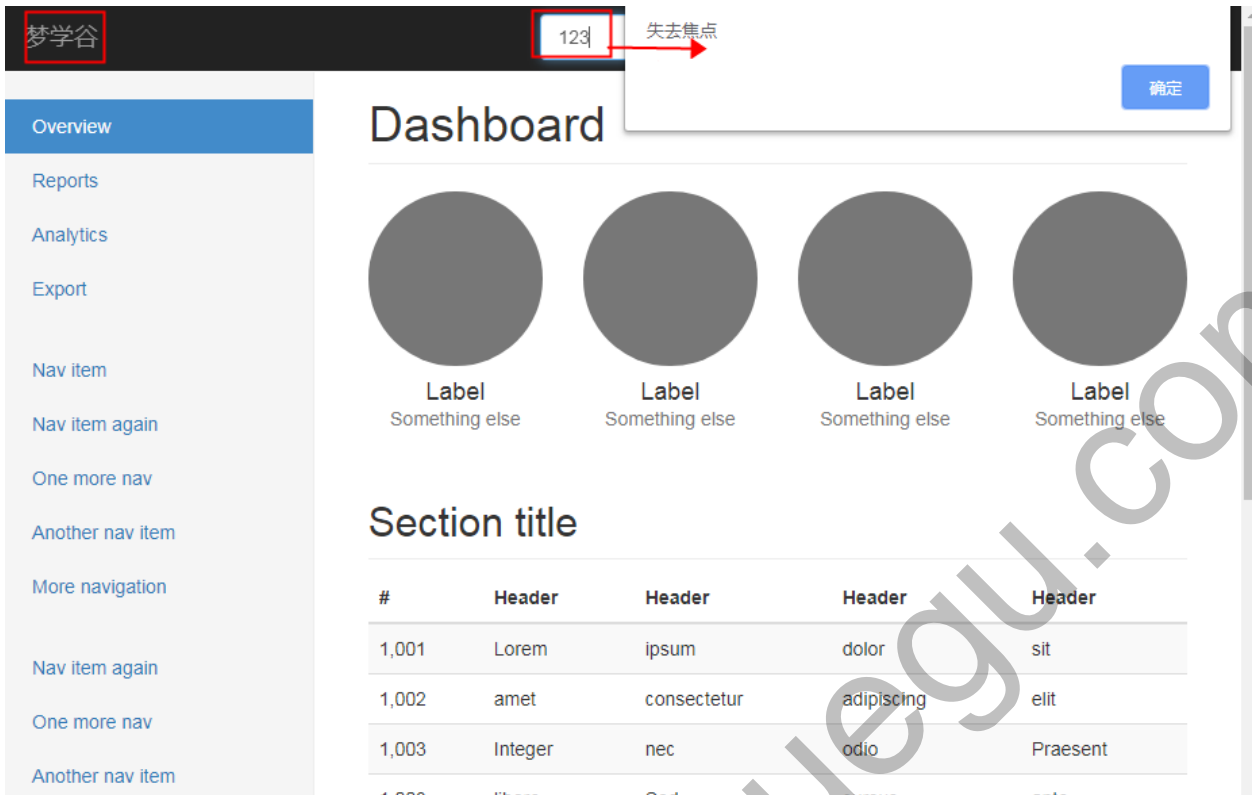
```
1 ;(function () {
2   const template = `<nav class="navbar navbar-inverse navbar-fixed-top">
3     <div class="container-fluid">
4       <div class="navbar-header">
5         <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-
6           target="#navbar" aria-expanded="false" aria-controls="navbar">
7           <span class="sr-only">Toggle navigation</span>
8           <span class="icon-bar"></span>
9           <span class="icon-bar"></span>
10          </span>
11          <a class="navbar-brand" href="#">{{ projectName }}</a>
12        </div>
13        <div id="navbar" class="navbar-collapse collapse">
14          <ul class="nav navbar-nav navbar-right">
15            <li><a href="#">Dashboard</a></li>
16            <li><a href="#">Settings</a></li>
17            <li><a href="#">Profile</a></li>
18            <li><a href="#">Help</a></li>
19          </ul>
20          <form class="navbar-form navbar-right">
21            <input type="text" class="form-control" placeholder="Search..." @blur="search">
22          </form>
23        </div>
24      </div>
25    </nav>`
```

```
26
27 window.Navbar = { // 添加 window 域中,html 页面才可以进行获取
28   template, // 等价于 template: template,
29   data: function() {
30     return {
31       projectName: '梦学谷'
32     }
33   },
34   methods: {
35     search() {
36       alert('失去焦点')
37     }
38   }
39 }
40 })()
```

8. 在 index.html 中引入 AppNavbar.js

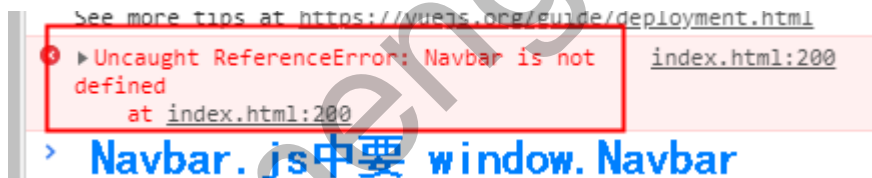
```
1 <body>
2   <div id="app">
3     <!-- 头部导航区域 -->
4     <app-navbar></app-navbar>
5
6     <script src="../../node_modules/vue/dist/vue.js"></script>
7     <!-- 注意要在 vue.js 下面引入 -->
8     <script src="components/AppNavbar.js"></script>
9     <script>
10      new Vue({
11        el: '#app',
12        components: {
13          AppNavbar // 等价于 AppNavbar: AppNavbar
14        }
15      })
16    </script>
17  </div>
18 </body>
```

9. 效果



10. 如果报错解决方式，

- 在 index.html 中要引入 AppNavbar.js
- AppNavbar.js 要将 AppNavbar 放到 window 域中



6.4.3 左侧导航组件注册

1. AppLeaf.js 文件中定义 AppLeaf 组件对象

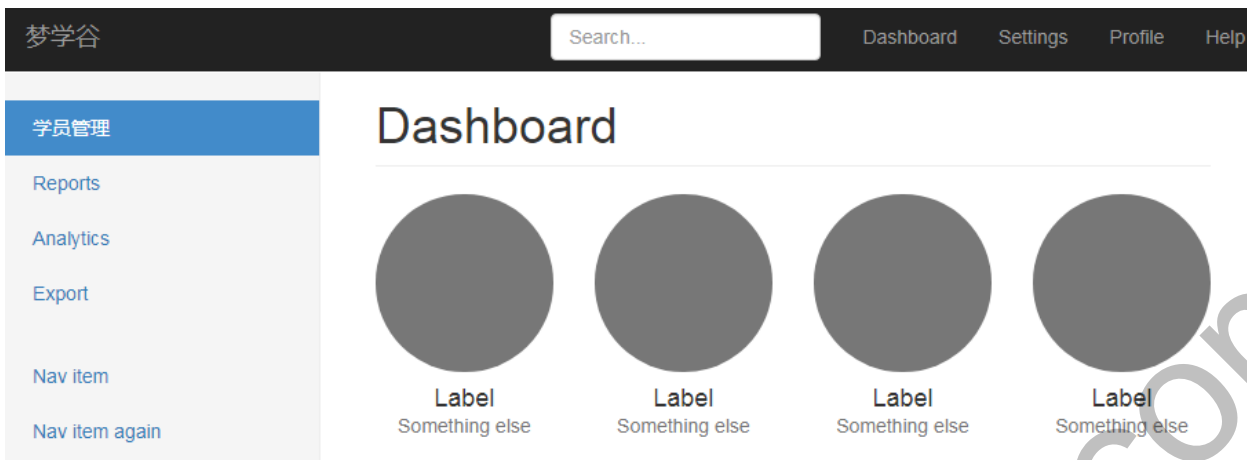


```
16 <ul class="nav nav-sidebar">
17   <li><a href="">Nav item again</a></li>
18   <li><a href="">One more nav</a></li>
19   <li><a href="">Another nav item</a></li>
20 </ul>
21 </div>`
22
23 window.AppLeaf = {
24   template,
25   data: function() {
26     return {
27       message: '学员管理'
28     }
29   }
30 }
31 })
```

2. 在 index.html 中引入 AppLeaf.js , Vue实例中添加 AppLeaf 组件, 主页面通过 <app-leaf/> 组件渲染

```
1 <body>
2   <div id="app">
3     <!--头部导航区域-->
4     <app-navbar></app-navbar>
5
6     <!--核心区域:分左右两边-->
7     <div class="container-fluid">
8       <div class="row">
9         <!--左边菜单栏区域-->
10        <app-leaf></app-leaf>
11
12      </div>
13    </div>
14
15  </div>
16  <script src="../node_modules/vue/dist/vue.js"></script>
17  <!-- 注意要在 vue.js 下面引入 -->
18  <script src="components/AppNavbar.js"></script>
19  <script src="components/AppLeaf.js"></script>
20  <script>
21    new Vue({
22      el: '#app',
23      components: {
24        AppNavbar, // 等价于 AppNavbar: AppNavbar
25        AppLeaf
26      }
27    })
28  </script>
29 </body>
```

3. 效果



6.4.4 右侧主页面组件注册

6.4.4.1 主页面组件化

1. 在 `home\AppHome.js` 中定义 `AppHome` 组件对象

```
1 ;(function(){
2   const template = `<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
3     <!--右边上半区域-->
4     <h1 class="page-header">Dashboard</h1>
5     <div class="row placeholders">
6       <div class="col-xs-6 col-sm-3 placeholder">
7         
10        <h4>Label</h4>
11        <span class="text-muted">Something else</span>
12      </div>
13      <div class="col-xs-6 col-sm-3 placeholder">
14        
17        <h4>Label</h4>
18        <span class="text-muted">Something else</span>
19      </div>
20      <div class="col-xs-6 col-sm-3 placeholder">
21        
24        <h4>Label</h4>
25        <span class="text-muted">Something else</span>
26      </div>
27    </div>
28  </div>`
29 })
```

```
25     </div>
26   </div>
27   <!--右边下半区域-->
28   <h2 class="sub-header">Section title</h2>
29   <div class="table-responsive">
30     <table class="table table-striped">
31       <thead>
32         <tr>
33           <th>#</th>
34           <th>Header</th>
35           <th>Header</th>
36           <th>Header</th>
37           <th>Header</th>
38         </tr>
39       </thead>
40       <tbody>
41         <tr>
42           <td>1,001</td>
43           <td>Lorem</td>
44           <td>ipsum</td>
45           <td>dolor</td>
46           <td>sit</td>
47         </tr>
48       </tbody>
49     </table>
50   </div>
51 </div>`
52 window.AppHome = {
53   template
54 }
55 })()
```

2. 在 index.html 中引入 components/home/AppHome.js , Vue实例添加 AppHome 组件,
主页面通过 <app-home/> 组件渲染

```
1  <body>
2    <div id="app">
3
4      <!--头部导航区域-->
5      <app-navbar></app-navbar>
6
7      <!--核心区域:分左右两边-->
8      <div class="container-fluid">
9        <div class="row">
10         <!--左边菜单栏区域-->
11         <app-leaf></app-leaf>
12         <!--右边主页面区域: 分上下两个区域-->
13         <app-home></app-home>
14       </div>
15     </div>
16   </div>
17
```

```
18 </div>
19 <script src="../../node_modules/vue/dist/vue.js"></script>
20 <!-- 注意要在 vue.js 下面引入 -->
21 <script src="components/AppNavbar.js"></script>
22 <script src="components/AppLeaf.js"></script>
23 <script src="components/home/AppHome.js"></script>
24 <script>
25   new Vue({
26     el: '#app',
27     components: {
28       AppNavbar, // 等价于 AppNavbar: AppNavbar
29       AppLeaf,
30       AppHome
31     }
32   })
33 </script>
34 </body>
```

6.4.4.2 Dashboard 子组件化

- 将 AppHome.js 中的 右边上半区域 中的 <div> 部分剪切到 Dashboard 组件中，Dashboard.js 内容如下：

```
1 ;(function () {
2   const template = `<div class="row placeholders">
3     <div class="col-xs-6 col-sm-3 placeholder">
4       
7       <h4>Label</h4>
8       <span class="text-muted">Something else</span>
9     </div>
10    <div class="col-xs-6 col-sm-3 placeholder">
11      
14      <h4>Label</h4>
15      <span class="text-muted">Something else</span>
16    </div>
17    <div class="col-xs-6 col-sm-3 placeholder">
18      
21      <h4>Label</h4>
22      <span class="text-muted">Something else</span>
```

```
22     </div>
23     </div>`
24
25     window.Dashboard = {
26       template
27     }
28
29   })()
```

2. 在 AppHome.js 中引入 Dashboard 组件作为 AppHome 的子组件

- 在 AppHome 对象的 components 选项中注册 Dashboard 组件
- 在 AppHome 对象的 template 模板中引入组件 <dashboard/>

```
1  ;(function(){
2    const template = `<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
3      <!--右边上半区域-->
4      <h1 class="page-header">Dashboard</h1>
5      <dashboard></dashboard>
6      <!--右边下半区域-->
7      <h2 class="sub-header">Section title</h2>
8      <div class="table-responsive">
9        <table class="table table-striped">
10         <thead>
11           <tr>
12             <th>#</th>
13             <th>Header</th>
14             <th>Header</th>
15             <th>Header</th>
16             <th>Header</th>
17           </tr>
18         </thead>
19         <tbody>
20           <tr>
21             <td>1,001</td>
22             <td>Lorem</td>
23             <td>ipsum</td>
24             <td>dolor</td>
25             <td>sit</td>
26           </tr>
27         </tbody>
28       </table>
29     </div>
30   </div>`
31
32   window.AppHome = {
33     template,
34     components: { // 子组件
35       Dashboard
36     }
37   }
38 })()
```

3. 在 index.html 中引入 Dashboard.js , 注意：Dashboard.js 要在 AppHome.js 前面引入

```
1 <script src="../../node_modules/vue/dist/vue.js"></script>
2 <!-- 注意要在 vue.js 下面引入 -->
3 <script src="components/Navbar.js"></script>
4 <script src="components/Leaf.js"></script>
5 <!-- 注意：Dashboard.js 要在 Home.js 前面引入 -->
6 <script src="components/Home/Dashboard.js"></script>
7 <script src="components/Home/Home.js"></script>
```

4. 访问页面如果控制台报错

- o 在 index.html 中 Dashboard.js 要在 AppHome.js 前面引入
- o Dashboard 单词不要写错了

6.4.4.3 查询列表子组件化

1. 将 AppHome.js 中的 右边下半区域 中的 <div> 部分剪切到 HomeList.js 组件中，HomeList.js 内容如下：

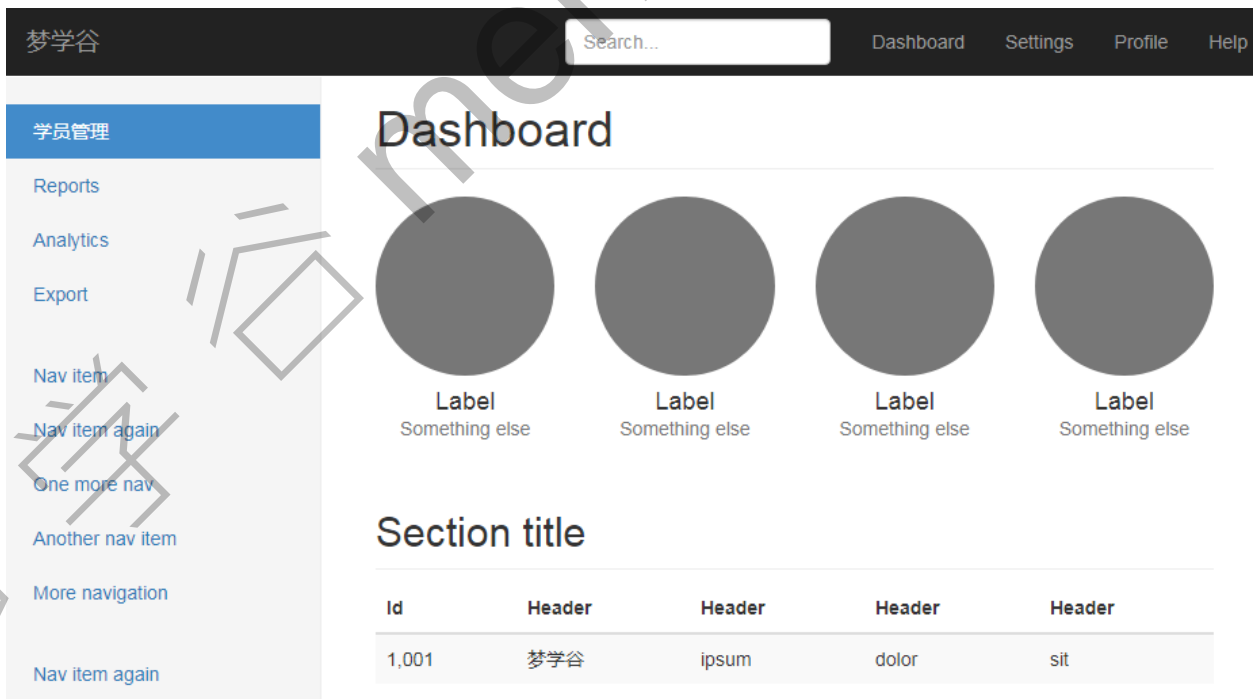
```
1 ;(function () {
2   const template = `<div class="table-responsive">
3     <table class="table table-striped">
4       <thead>
5         <tr>
6           <th>Id</th>
7           <th>Header</th>
8           <th>Header</th>
9           <th>Header</th>
10          <th>Header</th>
11        </tr>
12      </thead>
13      <tbody>
14        <tr>
15          <td>1,001</td>
16          <td>{{ name }}</td>
17          <td>ipsum</td>
18          <td>dolor</td>
19          <td>sit</td>
20        </tr>
21      </tbody>
22    </table>
23  </div>`
24   window.HomeList = {
25     template,
26     data () {
27       return {
28         name: '梦学谷'
29       }
30     }
31   }
32 })()
```

2. 在 AppHome.js 中引入 HomeList 组件作为 Home 的子组件

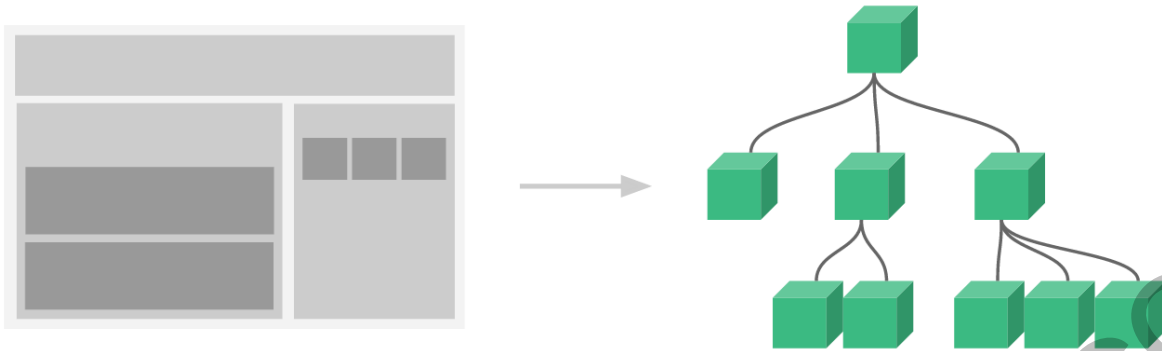
- 在 AppHome 对象的 components 选项中注册 HomeList 组件
- 在 AppHome 对象的 template 模板中引入组件 `<home-list/>`，注意是横线分隔方式引用组件

```
1  ;(function(){
2    const template = `<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
3      <!--右边上半区域-->
4      <h1 class="page-header">Dashboard</h1>
5      <dashboard></dashboard>
6      <!--右边下半区域-->
7      <h2 class="sub-header">Section title</h2>
8      <home-list></home-list>
9    </div>`
10
11    window.Home = {
12      template,
13      components: { // 子组件
14        Dashboard,
15        HomeList
16      }
17    }
18
19  })()
```

3. 效果



6.4.5 极致组件化



6.4.5.1 根组件提取 App.js

- 将 `index.html` 中 `<div id="#app">` 标签体中的代码提取出来，变成一个根组件存入 `App.js` 中，

提取的内容如下：

```
1 <app-navbar></app-navbar>
2 <!--核心区域:分左右两边-->
3 <div class="container-fluid">
4   <div class="row">
5     <!--左边菜单栏区域-->
6     <app-leaf></app-leaf>
7     <app-home></app-home>
8   </div>
9 </div>
```

- 在 `03-bootstrap` 目录下创建 `App.js` 文件
- 将 `<div id="#app">` 标签体中的代码剪切到 `App.js` 文件中作为模板页面

注意：template 模板中必须需要的根元素，所以要在提取的内容外层加上 `<div></div>`，一定要不要少了，不然报以下错误：

```
[Vue warn]: Error compiling template: vue.js:634
Component template should contain exactly one root element. If you are using v-if on multiple
elements, use v-else-if to chain them instead.
2 |   <navbar></navbar>
3 |   <!--核心区域:分左右两边-->
```

```
1 ;(function() {
2   // 不要少了最外层的根元素 div
3   const template = `<div>
4     <app-navbar></app-navbar>
5     <!--核心区域:分左右两边-->
6     <div class="container-fluid">
7       <div class="row">
8         <!--左边菜单栏区域-->
9         <app-leaf></app-leaf>
10        <app-home></app-home>
```

```
11     </div>
12   </div>
13 </div>`
14   window.App = {
15     template
16   }
17 })
```

6.4.5.2 剪切组件对象

- 当前 App.js 的 template 中引用了 AppNavbar、AppLeaf 和 AppHome 组件，所以我们要将 index.html 中的 components 选项中的组件对象剪切到 App 组件对象中。

App.js 代码如下：

```
1 ;(function() {
2   // 不要少了最外层的根元素 div
3   const template = `<div>
4     <navbar></navbar>
5     <!--核心区域:分左右两边-->
6     <div class="container-fluid">
7       <div class="row">
8         <!--左边菜单栏区域-->
9         <leaf></leaf>
10        <home></home>
11      </div>
12    </div>
13  </div>`
14   window.App = {
15     template,
16     components: {
17       AppNavbar, // 等价于 Navbar: Navbar
18       AppLeaf,
19       AppHome
20     }
21   }
22 })
```

6.4.5.3 index.html 引入 App.js

- 在 index.html 的引入 App.js 文件，并在 Vue 实例中的 components 选项中引入 App 组件，然后在 <div id="app"> 下引用 </app>

```
1 <body>
2   <!-- 留一个组件的出口，此处要被子组件替换 -->
3   <div id="app">
4     <app></app>
5   </div>
6   <script src="../node_modules/vue/dist/vue.js"></script>
7   <!-- 注意要在 vue.js 下面引入 -->
8   <script src="components/Navbar.js"></script>
```



```
9 <script src="components/Leaf.js"></script>
10 <!-- 注意：Dashboard.js 和 HomeList.js 要在 Home.js 前面引入 -->
11 <script src="components/Home/Dashboard.js"></script>
12 <script src="components/Home/HomeList.js"></script>
13 <script src="components/Home/Home.js"></script>
14 <script src="App.js"></script>
15 <script>
16   new Vue({
17     el: '#app',
18     components: {
19       App //等价于 App: App
20     }
21   })
22 </script>
23 </body>
```

2. 在 `<div id="app">` 下通过 `</app>` 引用 `App` 组件不是很好, 因为页面代码中会多出一个 `div`。

更好的方式是, 在 `<div id="app">` 下无须使用 `</app>` 引用 `App` 组件,

可以通过 `Vue` 根实例的 `template` 选项引用组件 `<app></app>` 后, 然后会把 `template` 中的渲染结果替换掉 `#app` 标签。

```
1 <body>
2 <!-- 留一个组件的出口，此处要被子组件替换 -->
3 <div id="app">
4 </div>
5 <script src="../../node_modules/vue/dist/vue.js"></script>
6 <!-- 注意要在 vue.js 下面引入 -->
7 <script src="components/Navbar.js"></script>
8 <script src="components/Leaf.js"></script>
9 <!-- 注意：Dashboard.js 和 HomeList.js 要在 Home.js 前面引入 -->
10 <script src="components/Home/Dashboard.js"></script>
11 <script src="components/Home/HomeList.js"></script>
12 <script src="components/Home/Home.js"></script>
13 <script src="App.js"></script>
14 <script>
15   new Vue({
16     el: '#app',
17     //Vue根实例中有template选项引用了组件后，然后会把template中的渲染结果替换掉 #app 标识的元素。
18     template: '<app></app>',
19     components: {
20       App //等价于 App: App
21     }
22   })
23 </script>
24 </body>
```

3. 当前 `index.html` 文件中还有 JS 代码, 可以将这些 JS 代码提取出来放到 `main.js` 中

- 在 `03-bootstrap` 目录下创建一个 `main.js` 文件

- 剪切 `Vue` 实例化代码到 `main.js` 中

```
1 new Vue({
2   el: '#app',
3   // Vue 根实例中有 template 选项引用了组件后，然后会把 template 中的渲染结果替换掉 #app 标签
4   template: '<app></app>',
5   components: {
6     App //等价于 App: App
7   }
8 })
```

- `index.html` 文件中引入 `main.js`

```
1 <body>
2   <!-- 留一个组件的出口，此处要被子组件替换 -->
3   <div id="app">
4   </div>
5   <script src="../../node_modules/vue/dist/vue.js"></script>
6   <!-- 注意要在 vue.js 下面引入 -->
7   <script src="components/Navbar.js"></script>
8   <script src="components/Leaf.js"></script>
9   <!-- 注意：Dashboard.js 和 HomeList.js 要在 Home.js 前面引入 -->
10  <script src="components/Home/Dashboard.js"></script>
11  <script src="components/Home/HomeList.js"></script>
12  <script src="components/Home/Home.js"></script>
13  <script src="App.js"></script>
14  <script src="main.js"></script>
15 </body>
```

6.5 组件化注意事项

- 组件可以理解为特殊的 Vue 实例，不需要手动实例化，管理自己的 `template` 模板
- 组件的 `template` 必须有且只有一个根节点
- 组件的 `data` 选项必须是函数，且函数体返回一个对象
- 组件与组件之间是相互独立的，可以配置自己的一些选项资源 `data`、`methods`、`computed` 等等
- 思想：组件自己管理自己，不影响别人

6.6 Vue 父子组件间通信

6.6.1 组件间通信方式

1. `props` 父组件向子组件传递数据
2. `$emit` 自定义事件
3. `slot` 插槽分发内容

6.6.2 组件间通信规则

1. 不要在子组件中直接修改父组件传递的数据
2. 数据初始化时，应当看初始化的数据是否用于多个组件中，如果需要被用于多个组件中，则初始化在父组件中；如果只在一个组件中使用，那就初始化在这个要使用的组件中。
3. 数据初始化在哪个组件，更新数据的方法(函数)就应该定义在哪个组件。

6.6.3 `props` 向子组件传递数据

6.6.3.1 声明组件对象中定义 `props`

1. 在声明组件对象中使用 `props` 选项指定

```
1 const MyComponent = {  
2   template: '<div></div>',  
3   props: 此处值有以下3种方式,  
4   components: {  
5   }  
6 }
```

- 方式1：指定传递属性名，注意是 数组形式

```
1 props: ['id', 'name', 'salary', 'isPublished', 'commentIds', 'author', 'getEmp']
```

- 方式2：指定传递属性名和数据类型，注意是 对象形式

```
1 props: {  
2   id: Number,  
3   name: String,  
4   salary: Number,  
5   isPublished: Boolean,  
6   commentIds: Array,  
7   author: Object,  
8   getEmp: Function  
9 }
```

- 方式3：指定属性名、数据类型、必要性、默认值

```
1 props: {  
2   name: {  
3     type: String,  
4     required: true,  
5     default: 'mxg'  
6   }  
7 }
```

6.6.3.2 引用组件时动态赋值

在引用组件时, 通过 `v-bind` 动态赋值

```
1 <my-component v-bind:id="2" :name="meng" :salary="9999" :is-published="true" :comment-ids="[1, 2]"  
  :author="{name: 'alan'}" :get-emp="getEmp" >  
2 </my-component>
```

6.6.3.3 传递数据注意

1. `props` 只用于父组件向子组件传递数据
2. 所有标签属性都会成为组件对象的属性, 模板页面可以直接引用
3. 问题:
 - a. 如果需要向非子后代传递数据, 必须多层逐层传递
 - b. 兄弟组件间也不能直接 `props` 通信, 必须借助父组件才可以

6.6.4 props 案例-列表渲染

需求: 实现 DashBoard 和 HomeList 组件中渲染数据功能

6.6.4.1 复制 bootstrap 案例

1. 复制 `vue-05-组件化\03-bootstrap` 项目为 `vue-06-组件间通信`
2. NPM 安装 `vue` 模块
3. 更改 `vue-06-组件间通信\index.html` 中 `vue.js` 的路径

```
1 <script src="node_modules/vue/dist/vue.js"></script>
```

4. 测试访问 `index.html` 正常, 再进行下面操作

6.6.4.2 AppHome.js

```
1 ;(function () {
2   const template = `<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
3
4     <!--右边上半区域-->
5     <h1 class="page-header">Dashboard</h1>
6     <dashboard :hobbies="hobbies"></dashboard>
7
8     <!--右边下半区域-->
9     <h2 class="sub-header">员工管理</h2>
10    <home-list :emp-list="empList"></home-list>
11  </div>`
12
13  window.AppHome = {
14    template, // template: template,
15    data () {
16      return {
17        hobbies: ['看书', '台球', '睡觉', '撸代码'],
18        empList: [
19          {id: 1, name: '小梦1', salary: 80001},
20          {id: 2, name: '小梦2', salary: 80002},
21          {id: 3, name: '小梦3', salary: 80003},
22          {id: 4, name: '小梦4', salary: 80004}
23        ]
24      }
25    },
26    components: { //Dashboard 作为AppHome 的子组件
27      Dashboard, // Dashboard: Dashboard
28      HomeList // HomeList:HomeList
29    },
30  }
31 })()
```

6.6.4.3 Dashboard.js

```
1 ;(function () {
2   const template = `
3     <div class="row placeholders">
4       <div v-for="(hobby, index) in hobbies" :key="index" class="col-xs-6 col-sm-3 placeholder">
5         
8         <h4>{{ hobby }}</h4>
9         <span class="text-muted">Something else</span>
10       </div>
11     </div>`
12
13   window.Dashboard = {
14     // 声明当前子组件接收父组件的属性
15     props: ['hobbies'],
16     template // template: template
17   }
18 })()
```

6.6.4.4 HomeList.js

将 HomeList.js 中列表 `<tr></tr>` 抽取到一个新的 `Item` 组件中

```
1 ;(function () {
2   const template = `<div class="table-responsive">
3     <table class="table table-striped">
4       <thead>
5         <tr>
6           <th>ID</th>
7           <th>姓名</th>
8           <th>工资</th>
9         </tr>
10      </thead>
11      <tbody>
12        <Item v-for="(emp, index) in empList" :key="emp.id" :emp="emp"/>
13      </tbody>
14    </table>
15  </div>`
16
17  window.HomeList = {
18    //声明组件接收父组件传递的属性
19    props: {
20      empList: Array
21    },
22    template, // template: template
23    components: {
24      Item
25    }
26  }
27 })()
```

6.6.4.5 Item.js

新增的 `Item.js` 位于 `vue-06-组件化通信\components\home` 目录下

```
1 ;(function () {
2   const template = `
3     <tr>
4       <td>{{emp.id}}</td>
5       <td>{{emp.name}}</td>
6       <td>{{emp.salary}}</td>
7     </tr>
8   `
9   window.Item = {
10     props: {
11       emp: { // 指定属性名/数据类型/是否必须
12         type: Object,
13         required: true
14       }
15     },
16   }
```

```
16     template
17   }
18 })()
```

6.6.4.6 index.js 引入 item.js

注意：引入时的顺序和位置

```
1  <body>
2    <div id="app">
3    </div>
4    <script src="node_modules/vue/dist/vue.js"></script>
5    <script src="components/AppNavbar.js"></script>
6    <script src="components/AppLeaf.js"></script>
7    <script src="components/home/Dashboard.js"></script>
8    <!--引入的位置-->
9    <script src="components/home/Item.js"></script>
10   <script src="components/home/HomeList.js"></script>
11   <script src="components/home/AppHome.js"></script>
12   <script src="App.js"></script>
13   <script src="main.js"></script>
14
15 </body>
```

6.6.5 props 案例-删除员工功能实现

6.6.5.1 AppHome.js

1. 因为删除 emp 是对 empList 做更新操作，而 empList 是初始化在当前这个组件里，所以删除的函数要定义在 AppHome.js 这个组件中
2. 向子组件传递 deleteEmp 函数 :deleteEmp="deleteEmp"

```
1  ;(function () {
2    const template = `<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
3
4    <!-- 右边上半区域-->
5    <h1 class="page-header">Dashboard</h1>
6    <dashboard :hobbies="hobbies"></dashboard>
7
8    <!-- 右边下半区域-->
9    <h2 class="sub-header">员工管理</h2>
10   <home-list :emp-list="empList" :deleteEmp="deleteEmp"></home-list>
11   </div>`
12
13   window.AppHome = {
14     template, // template: template,
15     data() {
16       return {
17         hobbies: ['看书', '台球', '睡觉', '撸代码'],
```

```
18     empList: [  
19       { id: 1, name: '小梦1', salary: 80001 },  
20       { id: 2, name: '小梦2', salary: 80002 },  
21       { id: 3, name: '小梦3', salary: 80003 },  
22       { id: 4, name: '小梦4', salary: 80004 }  
23     ]  
24   }  
25 },  
26  
27   methods: {  
28     // 删除指定下标的数据  
29     // 因为删除 emp 会对 empList 做更新操作，  
30     // 而 empList 是初始化在当前这个组件里，所以删除的函数要定义在这个组件中  
31     deleteEmp(index) {  
32       this.empList.splice(index, 1)  
33     }  
34   },  
35  
36   components: { //Dashboard 作为AppHome 的子组件  
37     Dashboard, // Dashboard: Dashboard  
38     HomeList // HomeList: HomeList  
39   },  
40 }  
41 })()
```

6.6.5.2 HomeList.js

1. 模板中添加 `<th>操作</th>`
2. 父组件传递的 `deleteEmp` 函数在 `Item.js` 才会使用，所以需要通过 `HomeList.js` 逐层传递给 `Item.js`
3. 将 `index` 索引值 传递给 `Item.js` 中的组件

```
1  ;(function () {  
2    const template = `<div class="table-responsive">  
3      <table class="table table-striped">  
4        <thead>  
5          <tr>  
6            <th>ID</th>  
7            <th>姓名</th>  
8            <th>工资</th>  
9            <th>操作</th>  
10         </tr>  
11        </thead>  
12        <tbody>  
13          <Item v-for="(emp, index) in empList" :key="emp.id" :emp="emp" :deleteEmp="deleteEmp"  
14          :index="index"/>  
15        </tbody>  
16      </table>  
17    </div>`  
18    window.HomeList = {  
19      //声明组件接收父组件传递的属性  
20      props: {
```



```
21     empList: Array,  
22     deleteEmp: Function // 逐层传递  
23   },  
24  
25   template, // template: template  
26   components: {  
27     Item  
28   }  
29 }  
30 })()
```

6.6.5.4 Item.js

1. props 声明接收父组件传递的 deleteEmp 和 index 属性值
2. 为 删除 按钮添加点击事件 `删除`
3. 定义 deleteltem 处理函数，其中调用 AppHome.js 传递过来 deleteEmp 函数。

```
1 ;(function () {  
2   const template = `  
3     <tr>  
4       <td>{{emp.id}}</td>  
5       <td>{{emp.name}}</td>  
6       <td>{{emp.salary}}</td>  
7       <td>  
8         <a href="#" @click="deleteltem">删除</a>  
9       </td>  
10    </tr>  
11  `;  
12  window.Item = {  
13    props: {  
14      emp: { // 指定属性名/数据类型/是否必须  
15        type: Object,  
16        required: true  
17      },  
18      deleteEmp: Function,  
19      index: Number  
20    },  
21  
22    methods: {  
23      // 删除员工  
24      deleteltem () {  
25        if (window.confirm(`确定删除${this.emp.name}的记录吗?`)) {  
26          // 移除索引为index的一条记录，  
27          // 注意：不要少了 this  
28          this.deleteEmp(this.index)  
29        }  
30      }  
31    },  
32    template  
33  }  
34 })()
```

6.6.6 自定义事件

作用：通过 自定义事件 来代替 props 传入函数形式

6.6.6.1 绑定自定义事件

在父组件中定义事件监听函数，并引用子组件标签上 `v-on` 绑定事件监听。

```
1 // 通过 v-on 绑定
2 // @自定义事件名=事件监听函数
3 // 在子组件 dashboard 中触发 delete_hobby 事件来调用 deleteHobby 函数
4 <dashboard @delete_hobby="deleteHobby"></dashboard>
```

6.6.6.2 触发监听事件函数执行

在子组件中触发父组件的监听事件函数调用

```
1 // 子组件触发事件函数调用
2 // this.$emit(自定义事件名, data)
3 this.$emit('delete_emp', index)
```

6.6.6.3 自定义事件注意

1. 自定义事件只用于子组件向父组件发送消息(数据)
2. 隔代组件或兄弟组件间通信此种方式不合适

6.6.7 自定义事件案例-删除仪表盘

6.6.7.1 Dashboard.js

在子组件中触发父组件的监听事件函数执行删除操作

```
1 ;(function () {
2   const template = `
3     <div class="row placeholders">
4       <div v-for="(hobby, index) in hobbies" :key="index" class="col-xs-6 col-sm-3 placeholder">
5         
8         <h4>{{ hobby }}</h4>
9         <span class="text-muted">
10           <a href="#" @click="deleteHobby">删除</a>
11         </span>
12       </div>
13     </div>`
14
15   window.Dashboard = {
```

```
14 // 声明当前组件接收的属性
15 props: ['hobbies'],
16
17 methods: {
18   // 删除爱好
19   deleteHobby(index) {
20     // 触发父组件中 delete_hobby 事件进行删除操作
21     this.$emit('delete_hobby', index)
22   }
23 },
24
25 template // template: template
26 }
27 })()
```

6.6.7.2 AppHome.js

自定义事件删除函数deleteHobby

在引用 dashboard 组件标签上自定义事件, 绑定事件监听函数 @delete_hobby="deleteHobby"

```
1 ;(function () {
2   const template = `<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
3
4     <!--右边上半区域-->
5     <h1 class="page-header">Dashboard</h1>
6     <!--通过自定义事件实现删除功能:
7       @自定义事件名=事件监听函数
8       在子组件 dashboard 中触发 delete_hobby 事件来调用 deleteHobby 函数
9     -->
10    <dashboard :hobbies="hobbies" @delete_hobby="deleteHobby"></dashboard>
11
12    <!--右边下半区域-->
13    <h2 class="sub-header">员工管理</h2>
14    <home-list :emp-list="empList" :deleteEmp="deleteEmp"></home-list>
15    </div>`
16
17   window.AppHome = {
18     template, // template: template,
19     data() {
20       return {
21         hobbies: ['看书', '台球', '睡觉', '撸代码'],
22         empList: [
23           { id: 1, name: '小梦1', salary: 80001 },
24           { id: 2, name: '小梦2', salary: 80002 },
25           { id: 3, name: '小梦3', salary: 80003 },
26           { id: 4, name: '小梦4', salary: 80004 }
27         ]
28       }
29     },
30
31     methods: {
32       // 删除指定下标的数据
```

```
33 // 因为删除 emp 会对 empList 做更新操作，
34 // 而 empList 是初始化在当前这个组件里，所以删除的函数要定义在这个组件中
35 deleteEmp(index) {
36   this.empList.splice(index, 1)
37 },
38 //删除爱好
39 deleteHobby(index) {
40   this.hobbies.splice(index, 1)
41 }
42 },
43
44 components: { //Dashboard 作为AppHome 的子组件
45   Dashboard, // Dashboard: Dashboard
46   HomeList // HomeList:HomeList
47 },
48 }
49 })
```

注意: 删除员工有垮组件传递函数,不推荐使用自定义事件来实现删除

6.6.8 slot 插槽

作用: 主要用于父组件向子组件传递 标签+数据 , (而上面prop和自定事件只是传递数据)

场景 : 一般是某个位置需要经常动态切换显示效果 (如饿了么)

6.6.8.1 子组件定义插槽

在子组件中定义插槽, 当父组件向指定插槽传递标签数据后, 插槽处就被渲染, 否则插槽处不会被渲染.

```
1 <div>
2   <!-- name属性值指定唯一插槽名，父组件通过此名指定标签数据-->
3   <slot name="aaa">不确定的标签结构 1</slot>
4   <div>组件确定的标签结构</div>
5   <slot name="bbb">不确定的标签结构 2</slot>
6 </div>
```

6.6.8.2 父组件传递标签数据

```
1 <child>
2   <!--slot属性值对应子组件中的插槽的name属性值-->
3   <div slot="aaa">向 name=aaa 的插槽处插入此标签数据</div>
4   <div slot="bbb">向 name=bbb 的插槽处插入此标签数据</div>
5 </child>
```

6.6.8.3 插槽注意事项

1. 只能用于父组件向子组件传递 标签+数据

2. 传递的插槽标签中的数据处理都需要定义所在父组件中

6.6.9 slot 插槽案例

需求：将Dashboard 的标题通过插槽显示

6.6.9.1 AppHome.js

将 Dashboard 的标题标签定义为插槽 `<slot name="dashboard"></slot>`

```
1 ;(function () {
2   const template = `<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
3
4     <!-- 右边上半区域 -->
5     <!-- <h1 class="page-header">Dashboard</h1> -->
6     <!-- 定义插槽 -->
7     <slot name="dashboard"></slot>
8     <!-- 通过自定义事件实现删除功能:
9         @自定义事件名=事件监听函数
10        在子组件 dashboard 中触发 delete_hobby 事件来调用 deleteHobby 函数
11    -->
12    <dashboard :hobbies="hobbies" @delete_hobby="deleteHobby"></dashboard>
13
14    <!-- 右边下半区域 -->
15    <h2 class="sub-header">员工管理</h2>
16    <home-list :emp-list="empList" :deleteEmp="deleteEmp"></home-list>
17  </div>`
```

6.6.9.2 App.js

向子组件中定义的插槽处传递标签数据。其中标题使用 title 数据绑定

```
1 ;(function () {
2   // 组件模板中，必须包含有且只有一个根元素
3   const template = `<div id="#app">
4     <!-- 头部导航区域 -->
5     <app-navbar></app-navbar>
6
7     <!-- 核心区域: 分左右两边 -->
8     <div class="container-fluid">
9       <div class="row">
10
11         <!-- 左边菜单栏区域 -->
12         <app-leaf></app-leaf>
13
14         <!-- 右边主页面区域: 分上下两个区域 -->
15         <app-home>
16           <h1 slot="dashboard" class="page-header">{{title}}</h1>
17         </app-home>
18       </div>
19     </div>
20  </div>`
```

```
21
22
23 window.App = {
24   data() {
25     return {
26       title: '仪表盘'
27     }
28   },
29   template,
30   components: {
31     AppNavbar, // 等价于 AppNavbar: AppNavbar
32     AppLeaf, // AppLeaf: AppLeaf
33     AppHome
34   }
35 }
36 }()
```

6.7 非父子组件间通信 PubSubJS（补充知识）

注意：此部分内容是后面新增补充的新教程内容

6.7.1 介绍

Vue.js 可通过 PubSubJS 库来实现非父子组件之间的通信，使用 PubSubJS 的消息发布与订阅模式，来进行数据的传递。

理解：订阅信息 ==== 绑定事件监听，发布消息 ==== 触发事件。

注意：但是必须先执行订阅事件 subscribe，然后才能 publish 发布事件。

6.7.2 订阅消息(绑定事件监听)

先在 created 钩子函数中订阅消息

```
1 // event接收的是消息名称，data是接收发布时传递的数据
2 PubSub.subscribe('消息名称（相当于事件名）', function(event, data) {
3   // 事件回调处理
4 })
```

6.7.3 发布消息(触发事件)

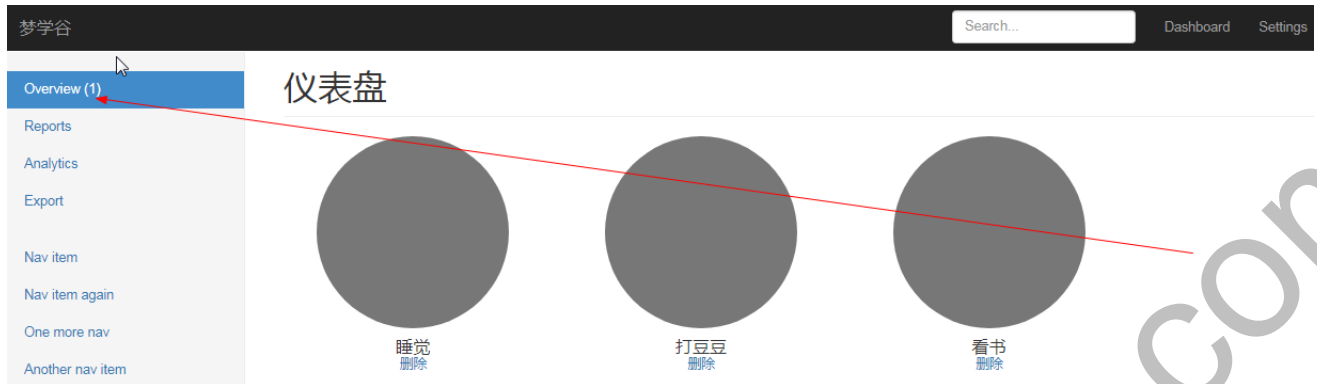
```
1 PubSub.publish('消息名称（相当于事件名）', data)
```

6.7.4 案例

复制 vue-06-组件间通信 为 vue-06-组件间通信2

需求：在右侧删除爱好数据成功后，在左侧导航栏上显示已删除的总数，没有删除则不显示数量。

分析：左侧导航与右侧爱好仪表盘是非父子组件，采用 PubSubJS 消息订阅发布实现功能



6.7.4.2 安装 pubsub-js

```
1 npm install pubsub-js
```

6.7.4.2 index.html 引入 pubsub-js 库

```
1 <script src="./node_modules/vue/dist/vue.js"></script>
2 <script src="./node_modules/pubsub-js/src/pubsub.js"></script>
```

6.7.4.3 AppLeaf.js 订阅消息

- 初始化数量，在 created 钩子中订阅消息，监听当删除爱好后，进行统计已删除总数

```
1 ;(function () {
2   const template = `<div class="col-sm-3 col-md-2 sidebar">
3     <ul class="nav nav-sidebar">
4       <li class="active">
5         <a href="#">Overview
6         <!--是否显示和显示总删除数-->
7         <span v-show="delNum">{{ delNum }}</span>
8       </li>
9     </ul>
10    <li><a href="#">Reports</a></li>
11    <li><a href="#">Analytics</a></li>
12    <li><a href="#">Export</a></li>
13  </ul>
14  </div>`
15
16  window.AppLeaf = {
17    template,
18    data() {
19      return {
20        delNum: 0 //初始化数量
21      }
22    },
23    created() { //
```

```
24 //一开始初始化实例时，就进行订阅消息
25 PubSub.subscribe('changeNum', (event, num) => { // 箭头函数
26   // 删除成功
27   this.delNum = this.delNum + num
28 })
29 },
30 }
31 })()
```

6.7.4.4 AppHome.js 发布消息

```
1  methods: {
2    . . .
3    . . .
4    // 删除爱好
5    deleteHobby (index) {
6      this.hobbies.splice(index, 1)
7      // 删除成功，发布消息，导航统计数据
8      PubSub.publish('changeNum', 1)
9    }
10  },
```

6.7.5 优点

不管是父子之间还是非父子之间通信 PubSubJS 都可以实现

6.8 单文件组件

6.8.1 当前存在的问题

上面定义组件时存在的问题:

- **全局定义 (Global definitions)** 强制要求每个 component 中的命名不得重复
- **字符串模板 (String templates)** 缺乏语法高亮，在 HTML 有多行的时候，需要用到丑陋的 \
- **不支持 CSS (No CSS support)** 意味着当 HTML 和 JavaScript 组件化时，CSS 明显被遗漏
- **没有构建步骤 (No build step)** 限制只能使用 HTML 和 ES5 JavaScript, 而不能使用预处理器，如 Pug (formerly Jade) 和 Babel

文件扩展名为 `.vue` 的 **single-file components(单文件组件)** 为以上所有问题提供了解决方法，并且还可以使用 webpack 或 Browserify 等构建工具。

6.8.2 单文件组件模板格式


```
1 <template>
2 // 组件的模块
3 </template>
4
5 <script>
6 // 组件的JS
7 </script>
8
9 <style>
10 // 组件的样式
11 </style>
```

第七章 生命周期和 Ajax 服务端通信

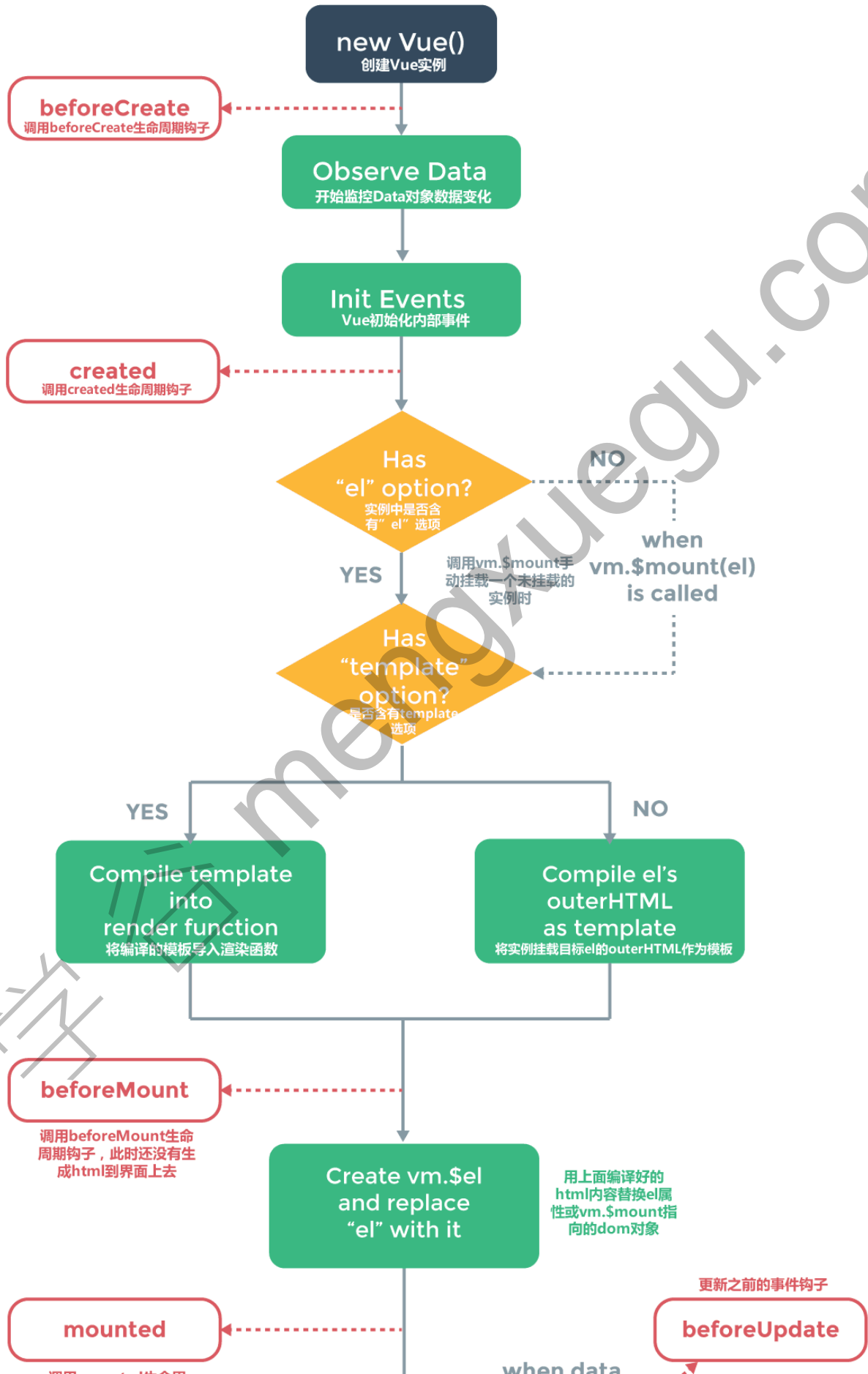
7.1 Vue 实例生命周期

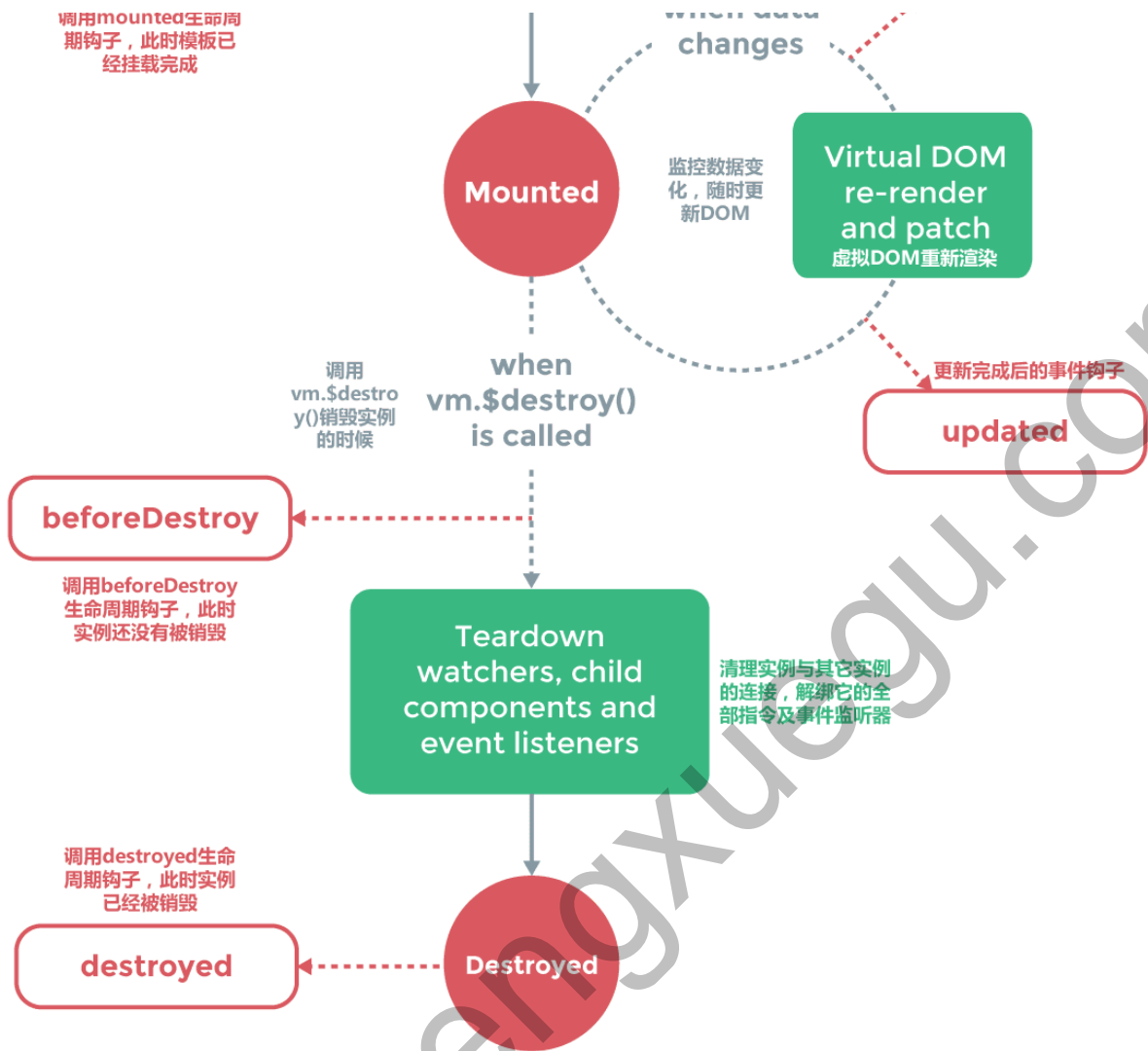
7.1.1 生命周期钩子函数

每个 Vue 实例在被创建时都要经过一系列的初始化过程

- 生命周期分为三大阶段：初始化显示、更新显示、销毁Vue实例
 - 初始化阶段的钩子函数：
 - beforeCreate() 实例创建前：数据和模板均未获取到
 - created()** 实例创建后：最早可访问到 data 数据，但模板未获取到
 - beforeMount() 数据挂载前：模板已获取到，但是数据未挂载到模板上。
 - mounted()** 数据挂载后：数据已挂载到模板中
 - 更新阶段的钩子函数：
 - beforeUpdate() 模板更新前：data 改变后，更新数据模板前调用
 - updated() 模板更新后：将 data 渲染到数据模板中
 - 销毁阶段的钩子函数：
 - beforeDestroy()** 实例销毁前
 - destroyed() 实例销毁后

7.1.2 生命周期图示





7.1.3 示例演示

- 创建 `vue-06-lifecycle&ajax` 目录，在它下面新建 `01-生命钩子.html` 文件，测试Vue实例生命周期
- 安装 Vue 模块
- 示例代码：

```

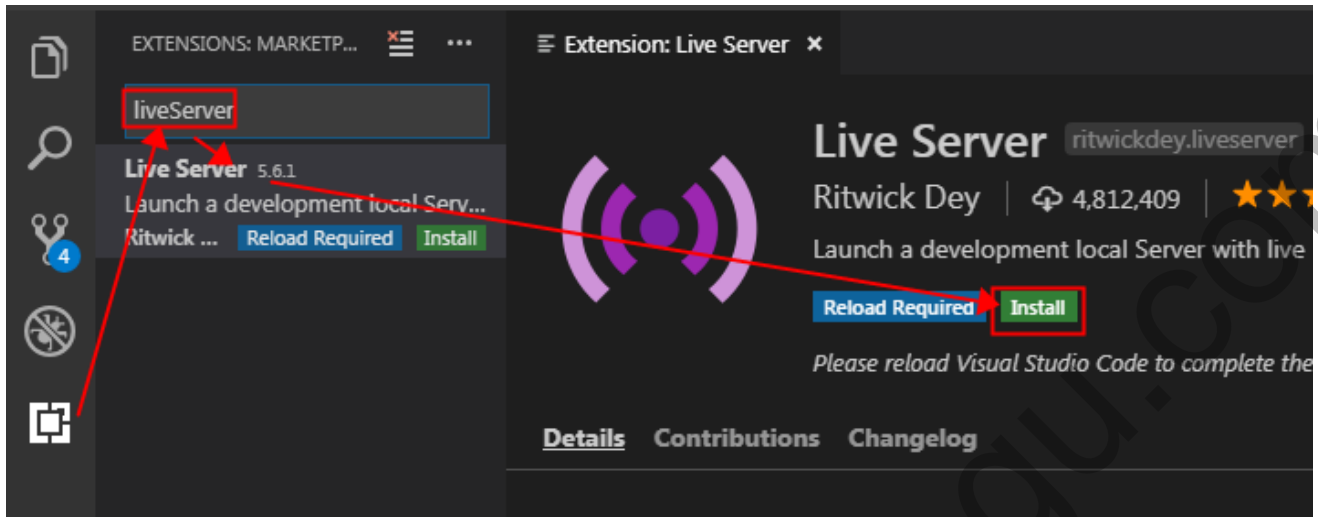
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <title>生命周期</title>
8 </head>
9 <body>
10  <div id="app">
11    <h1> {{ message }} </h1>
12  </div>
13  <script src="../node_modules/vue/dist/vue.js" ></script>
14  <script>
15    var vm = new Vue({

```

```
16     el: '#app',
17     data: {
18       message: '梦学谷'
19     },
20
21     beforeCreate() {
22       console.log('beforeCreate()', this.$el, this.$data)
23     },
24
25     created() {
26       // 已初始化 data 数据，但数据未挂载到模板中
27       console.log('created()', this.$el, this.$data)
28     },
29
30     beforeMount() {
31       // 模板已获取到，但是数据未挂载到模板上
32       console.log('beforeMount()', this.$el, this.$data)
33     },
34
35     mounted() {
36       // 编译完成，数据已挂载到模板中
37       console.log('mounted()', this.$el, this.$data)
38     },
39
40     beforeUpdate() {
41       // 当 data 改变后，去更新模板中的数据前调用
42       // 注意：浏览器问题，需使用 this.$el.innerHTML 获取更新前的 Dom 模板数据
43       console.log('beforeUpdate()', this.$el.innerHTML, this.$data)
44     },
45
46     updated() {
47       // data 被 Vue 渲染之后的 Dom 数据模板
48       console.log('updated()', this.$el.innerHTML, this.$data)
49     },
50
51     beforeDestroy() {
52       // 销毁实例前调用
53       console.log('beforeDestroy()')
54     },
55
56     destroyed() {
57       // 销毁实例后调用
58       console.log('created()')
59     }
60   }).$mount('#app') // 实例中未使用 el 选项，可使用$mount()手动挂载 Dom
61
62   // vm.$destroy() 销毁 Vue 实例
63
64   </script>
65 </body>
66 </html>
```

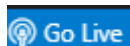
7.2 liveServer 服务端插件

1. 在 VS Code 中安装 liveServer 服务端插件，用于 Ajax 接口调用。



2. 启动服务器：

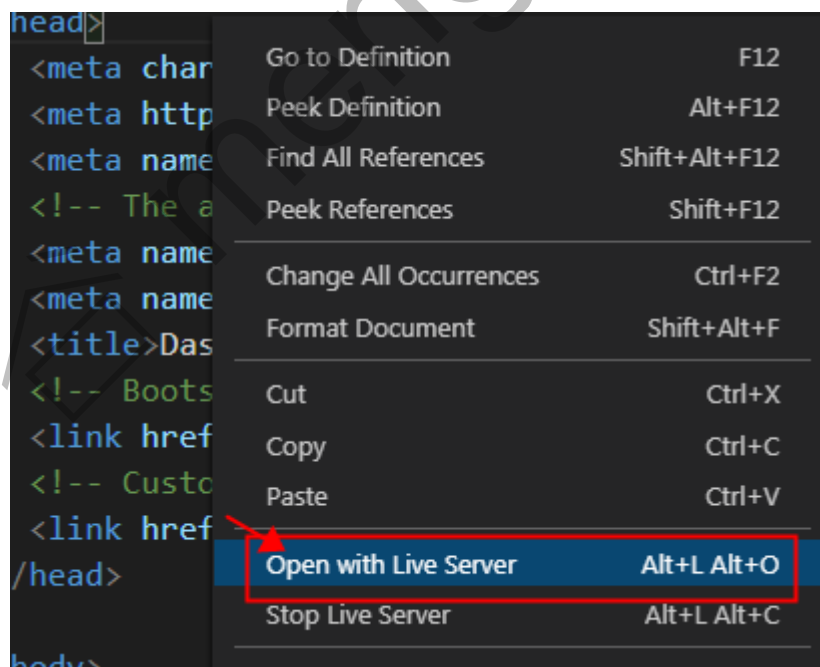
方式1：任意打开一个 html 页面，在最下面可看到



点击它可启动，默认端口5500

方式2：在 html 页面单击右键，点击如下，访问页面

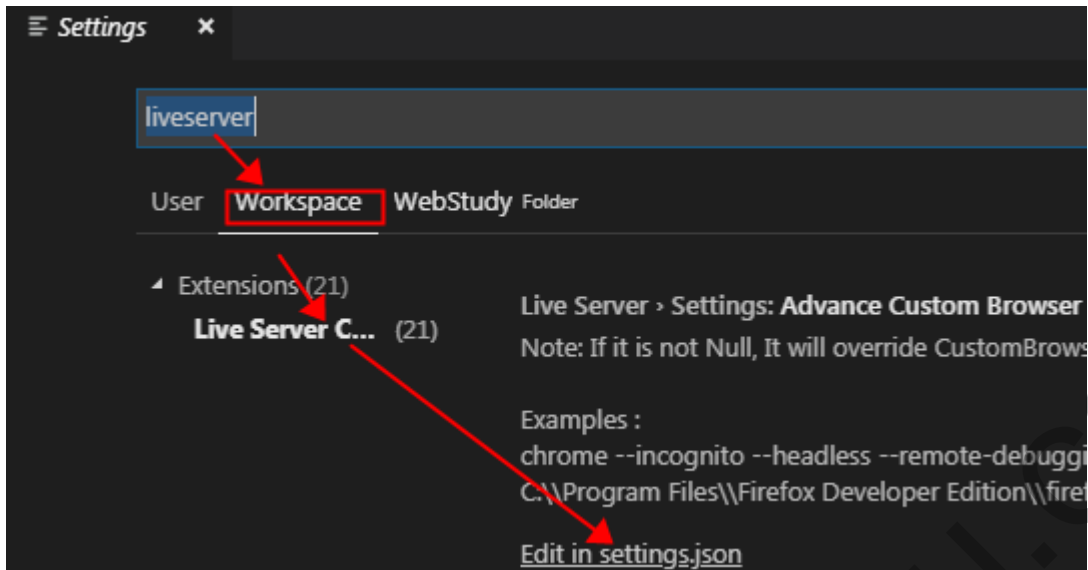
注意：如果之前启动过服务器，则使用 方式2 启动html页面，不能使用 方式1 会端口占用



3. 更改 liveServer 默认端口号：

按 `Ctrl + ,`，打开 Settings，如下操作，打开 settings.json，

再json文件中添加 `"liveServer.settings.port": 8080,`



7.3 Vue 中常用的 ajax 库

目前 Vue 官方没有内置任何 ajax 请求方法

7.3.1 vue-resource

- 在 vue1.x 版本中, 被广泛使用的非官方 Vue 插件 vue-resource

7.3.2 axios

- 在 vue 2+ 版本中, 官方推荐使用非常棒的第三方 ajax 请求库

使用: 结合生命钩子获取数据, 渲染数据

7.4 vue-resource 的使用

7.4.1 参考文档

<https://github.com/pagekit/vue-resource/blob/develop/docs/http.md>

7.4.2 安装

```
1 npm install vue-resource
```

7.4.3 示例演示

- 在 vue-07-lifecycle&ajax 目录下新建 db.json 文件存入模拟数据

```
1 [
2   {"name": "张三1", "age": 18},
3   {"name": "张三2", "age": 18}
4 ]
```

2. 在 `vue-07-lifecycle&ajax` 目录下，新建 `02-vue-resource.html` 文件

```
1 <body>
2   <div id="app">
3     <h1>{{ repData }}</h1>
4   </div>
5   <script src="./node_modules/vue/dist/vue.js"></script>
6   <!-- 放到 vue.js 下面 -->
7   <script src="./node_modules/vue-resource/dist/vue-resource.js"></script>
8   <script>
9     // 使用插件
10    Vue.use(VueResource)
11
12    new Vue({
13      el: '#app',
14      data: {
15        repData: []
16      },
17      created() {
18        this.$http.get('http://127.0.0.1:5500/vue-07-lifecycle&ajax/db.json').then((response) => {
19          // 如果要使用Vue实例的this，此处需要使用箭头函数
20          // success callback
21          console.log(response.data) // 响应数据
22          this.repData = response.data
23        }, (response) => {
24          // error callback
25          console.log(response.statusText) // 错误信息
26        })
27      },
28    })
29  </script>
30 </body>
```

7.5 axios 的使用

7.5.1 参考文档

<https://github.com/axios/axios/blob/master/README.md>

7.5.2 安装

```
1 npm install axios
```

7.5.3 示例演示

在 `vue-06-lifecycle&ajax` 目录下，新建 `03-axios.html` 文件

```
1 <body>
2   <div id="app">
3     <h1>{{ repData }}</h1>
4   </div>
5   <script src="./node_modules/vue/dist/vue.js"></script>
6   <script src="./node_modules/axios/dist/axios.js"></script>
7   <script>
8     new Vue({
9       el: '#app',
10      data: {
11        repData: []
12      },
13      created() {
14        // 发送 ajax 请求
15        axios.get('http://127.0.0.1:5500/vue-07-lifecycle&ajax/db.json').then(response => {
16          console.log(response.data) // 得到返回结果数据
17          this.repData = response.data
18        }).catch(error => {
19          console.log(error.message)
20        })
21      },
22    })
23  </script>
24 </body>
```

7.6 组件化 axios 通信

- 将 `vue-06-组件化通信` 复制到 `vue-07-lifecycle&ajax` 目录下，重命名为 `04-bootstrap-ajax`。
- 进入 `04-bootstrap-ajax` 命令行，安装 `axios`

```
1 npm install axios
```

- `index.html` 引入 `axios.js` 文件，注意引入位置

```
1 <body>
2   <div id="app">
3   </div>
4   <script src="node_modules/vue/dist/vue.js"></script>
5   <script src="node_modules/axios/dist/axios.js"></script>
6   <!-- 在上面引入 axios.js -->
7   <script src="components/AppNavbar.js"></script>
8   <script src="components/AppLeaf.js"></script>
9   <script src="components/home/Dashboard.js"></script>
10  <script src="components/home/Item.js"></script>
```



```
11 <script src="components/home/HomeList.js"></script>
12 <script src="components/home/AppHome.js"></script>
13 <script src="App.js"></script>
14 <script src="main.js"></script>
15 </body>
```

- 在 04-bootstrap-ajax 目录下新建 emp.json 文件存入模拟数据

```
1 [
2   {"id": 1, "name": "张三1", "salary": 9899},
3   {"id": 2, "name": "张三2", "salary": 9999},
4   {"id": 3, "name": "张三3", "salary": 9099},
5   {"id": 4, "name": "张三4", "salary": 9199}
6 ]
```

- 实现 axios 异步渲染数据, 重构 AppHome.js

```
1 ;(function () {
2   const template = `<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
3
4     <!--右边上半区域-->
5     <!-- <h1 class="page-header">Dashboard</h1-->
6     <!--定义插槽-->
7     <slot name="dashboard"></slot>
8     <!--通过自定义事件实现删除功能:
9       @自定义事件名=事件监听函数
10      在子组件 dashboard 中触发 delete_hobby 事件来调用 deleteHobby 函数
11    -->
12     <dashboard :hobbies="hobbies" @delete_hobby="deleteHobby"></dashboard>
13
14     <!--右边下半区域-->
15     <h2 class="sub-header">员工管理</h2>
16     <home-list :emp-list="empList" :deleteEmp="deleteEmp"></home-list>
17   </div>`
18
19   window.AppHome = {
20     template, // template: template,
21     data() {
22       return {
23         hobbies: ['看书', '台球', '睡觉', '撸代码'],
24         empList: []
25       }
26     },
27
28     // 生命周期钩子
29     created() {
30       axios.get('http://127.0.0.1:5500/vue-07-lifecycle&ajax/04-bootstrap-ajax/emp.json').then(response => {
31         console.log(response.data) // 得到返回结果数据
32         this.empList = response.data
33       }).catch(error => {
34         console.log(error.message)
35       })
36     }
37   }
38 })
```

```
36    },
37
38    methods: {
39      // 删除指定下标的数据
40      // 因为删除 emp 会对 empList 做更新操作，
41      // 而 empList 是初始化在当前这个组件里，所以删除的函数要定义在这个组件中
42      deleteEmp(index) {
43        this.empList.splice(index, 1)
44      },
45      //删除爱好
46      deleteHobby(index) {
47        this.hobbies.splice(index, 1)
48      }
49    },
50
51    components: { //Dashboard 作为AppHome 的子组件
52      Dashboard, // Dashboard: Dashboard
53      HomeList // HomeList:HomeList
54    },
55  }
56 })
```

• 效果：



第八章 Vue-Router 路由

8.1 什么是路由

Vue Router 是 [Vue.js](#) 官方的路由管理器。它和 Vue.js 的核心深度集成，让构建单页面应用变得非常简单。

通过根据不同的请求路径，切换显示不同组件进行渲染页面。

8.2 基本路由使用

创建 `vue-08-router` 目录，该目录下新建 `01-vue-router-demo.html` 文件

8.2.1 安装路由

注意：要安装 vue 模块

```
1 npm install vue-router
```

8.2.2 引入 vue-router.js

```
1 <script src="./node_modules/vue/dist/vue.js"></script>
2 <script src="./node_modules/vue-router/dist/vue-router.js"></script>
```

8.2.3 HTML 路由切换

```
1 <div id="app">
2   <div class="header">
3     <h1>头部</h1>
4   </div>
5   <div class="left">
6     <p>
7       <ul>
8         <!-- 方式1: 传统方式 -->
9         <li><a href="#/foo">Go Foo</a></li>
10        <li><a href="#/bar">Go Bar</a></li>
11        <!-- 方式2: 官方推荐 -->
12        <!-- <router-link> 默认会被渲染成一个 `<a>` 标签, -->
13        <!-- 通过传入 `to` 属性指定跳转链接, 不用像上面加 `#` 号 -->
14        <li><router-link to="/foo">Go to Foo</router-link></li>
15        <li><router-link to="/bar">Go to Bar</router-link></li>
16      </ul>
17    </p>
18  </div>
19  <div class="main">
20    <!-- 路由出口: 路由匹配到的组件将渲染在这里 -->
21    <router-view></router-view>
22  </div>
23 </div>
```

8.2.4 JS 配置路由

```
1 <script src="./node_modules/vue/dist/vue.js"></script>
2 <script src="./node_modules/vue-router/dist/vue-router.js"></script>
3 <script>
4   // 1. 定义组件
5   var Foo = {
6     template: '<div> foo 组件 </div>'
7   }
8   var Bar = {
9     template: '<div> bar 组件 </div>'
10  }
11
12  // 2. 配置路由表：当点击特定的 url 时，显示对应的那个组件。
13  const router = new VueRouter({
14    routes: [ //配置每个路由映射一个组件
15      { path: '/foo', component: Foo },
16      { path: '/bar', component: Bar }
17    ]
18  })
19
20  // 3. 注入路由到实例中
21  new Vue({
22    el: '#app',
23    router // router: router
24  })
25 </script>
```

8.3 路由案例实战

8.3.1 拷贝项目

1. 拷贝 vue-07-lifecycle&ajax\04-bootstrap-ajax 项目到 vue-08-router 下，
重命名为：02-bootstrap-ajax-router
2. 进入 02-bootstrap-ajax-router 命令行，通过 NPM 安装 vue-router

```
1 npm install vue-router
```

3. 修改 axios 请求地址

因为改了目录名称，所以要修改 vue-08-router\02-bootstrap-ajax-router\components\home\AppHome.js 的URL，不然请求失败。

```
1 created() {  
2   axios.get('http://127.0.0.1:5500/vue-08-router/02-bootstrap-ajax-router/emp.json')  
3     .then(response => { //function (response) {  
4       console.log(response.data, this);  
5       this.empList = response.data  
6     })  
7     .catch(error => { //function (error) {  
8       alert(error.message)  
9     })  
10  },
```

4. 测试

是否可以正常访问: <http://127.0.0.1:5500/vue-08-router/02-bootstrap-ajax-router/index.html>

8.3.2 News 组件

在 vue-08-router\02-bootstrap-ajax-router\components\home 目录下新建 News.js 文件

News 模板位于: 01-配套源码\bootstarp\news.html

```
1 ;(function () {  
2   const template = `  
3     <!--右边主页面区域-->  
4     <div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">  
5       <div class="header clearfix">  
6         <nav>  
7           <ul class="nav nav-pills">  
8             <li class="active"><a href="#">体育</a></li>  
9             <li><a href="#">科技</a></li>  
10          </ul>  
11        </nav>  
12        <hr>  
13      </div>  
14  
15      <!--体育栏目-->  
16      <div>  
17        <ul>  
18          <li>  
19            <a href="#">世界杯开赛啦</a>  
20          </li>  
21          <li>  
22            <a href="#">NBA开赛倒计时</a>  
23          </li>  
24        </ul>  
25      <!--详情-->  
26      <div class="jumbotron">  
27        <h2>世界杯开赛啦</h2>  
28        <p>世界杯于今晚8点举行开幕式.....</p>  
29      </div>  
30    </div>
```

```
31      <!--科技栏目-->
32      <div>
33          <ul>
34              <li>
35                  <span>5G时代到来了 </span>
36                  <button class="btn btn-default btn-xs">查看(Push)</button>&nbsp;  
37                  <button class="btn btn-default btn-xs">查看(replace)</button>
38              </li>
39              <li>
40                  <span>互联网大洗牌</span>
41                  <button class="btn btn-default btn-xs">查看(Push)</button>&nbsp;  
42                  <button class="btn btn-default btn-xs">查看(replace)</button>
43              </li>
44          </ul>
45      <!--详情-->
46      <div class="jumbotron">
47          <h2>世界杯开赛啦</h2>
48          <p>世界杯于今晚8点举行开幕式.....</p>
49      </div>
50  </div>
51 </div>
52 `
53 window.News = {
54   template
55 }
56
57 })()
```

8.3.3 About 组件

在 vue-08-router\02-bootstrap-ajax-router\components\home 目录下新建 About.js 文件

```
1 ;(function () {
2   const template = `<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
3     <h1>梦学谷-陪你学习，伴你梦想！</h1>
4     <input />
5   </div>`
6
7   window.About = {
8     template
9   }
10 })()
```

8.3.4 配置路由

在 vue-08-router\02-bootstrap-ajax-router 目录下新建 router.js 路由配置文件

```
1 ;(function () {  
2   window.router = new VueRouter({  
3     routes: [  
4       {path: '/', component: AppHome},  
5       {path: '/news', component: News},  
6       {path: '/about', component: About}  
7     ]  
8   })  
9 })()
```

8.3.5 注入路由到实例中

在 main.js 中的 Vue 实例中引入 router

```
1 ;(function () {  
2   new Vue({  
3     el: '#app',  
4     // Vue实例中的template选项中引用了组件后，会将这个组件的渲染结果替换掉 #app 标签的元素  
5     // template: '<app> </app>',  
6     template: '<app/>',  
7     router, // 引用路由配置  
8     components: {  
9       App // 等价于 App: App  
10    }  
11  })  
12 })()
```

8.3.6 配置路由渲染组件出口

在 App.js 中配置

```
1 ;(function () {  
2   // 组件模板中，必须包含有且只有一个根元素  
3   const template = `<div id="#app">  
4     <!--头部导航区域-->  
5     <app-navbar></app-navbar>  
6  
7     <!--核心区域:分左右两边-->  
8     <div class="container-fluid">  
9       <div class="row">  
10  
11         <!--左边菜单栏区域-->  
12         <app-leaf></app-leaf>  
13  
14         <!--右边主页面区域: 分上下两个区域  
15         <app-home>  
16           <h1 slot="dashboard" class="page-header">{{title}}</h1>  
17         </app-home>  
18         -->  
19       <!-- 配置路由渲染组件出口 -->
```

```
20     <router-view>
21       <h1 slot="dashboard" class="page-header">{{title}}</h1>
22     </router-view>
23
24   </div>
25 </div>
26 </div>
27 `
28
29 window.App = {
30   template,
31   data () {
32     return {
33       title: '仪表盘'
34     }
35   },
36   components: {
37     AppNavbar, // 等价于 AppNavbar: AppNavbar
38     AppLeaf, // AppLeaf: AppLeaf
39     AppHome
40   }
41 }
42 })()
```

8.3.7 修改跳转链接

在 AppLeaf.js 中修改跳转链接

```
1 ;(function () {
2
3   window.AppLeaf = {
4     template: `<div class="col-sm-3 col-md-2 sidebar">
5       <ul class="nav nav-sidebar">
6         <li class="active">
7           <router-link to="/">首页</router-link>
8         </li>
9         <li>
10          <router-link to="/news">新闻管理</router-link>
11        </li>
12        <li>
13          <router-link to="/about">关于我们</router-link>
14        </li>
15      </ul>
16    </div>`
17   }
18
19 })()
```


8.3.8 引入 js 文件

在 index.html 中引入 `vue-router.js`、`Bar.js`、`Foo.js`、`router.js`，注意引入的位置顺序

```
1 <body>
2   <div id="app">
3   </div>
4   <script src="/node_modules/vue/dist/vue.js"></script>
5   <!-- vue-router.js要引入在 vue.js 下方-->
6   <script src="node_modules/vue-router/dist/vue-router.js"></script>
7   <script src="/node_modules/axios/dist/axios.js"></script>
8   <script src="components/AppNavbar.js"></script>
9   <script src="components/AppLeaf.js"></script>
10  <script src="components/home/Dashboard.js"></script>
11  <!-- 注意引入的位置 -->
12  <script src="components/home/Item.js"></script>
13  <script src="components/home/HomeList.js"></script>
14  <script src="components/home/AppHome.js"></script>
15  <!-- 引入 News.js About.js router.js-->
16  <script src="components/home/News.js"></script>
17  <script src="components/home/About.js"></script>
18  <script src="router.js"></script>
19  <script src="App.js"></script>
20  <script src="main.js"></script>
21 </body>
```

8.3.9 启动测试

访问：<http://127.0.0.1:5500/vue-08-router/02-bootstrap-ajax-router/index.html>

点击左侧菜单，右侧显示不同的组件效果

8.4 样式匹配-高亮显示导航

问题：你会发现不管点击左侧导航哪个链接，当前都是在第一个导航处显示高亮背景。

按 F12 查看源代码，发现高亮样式在 `<li class="active">` 标签上，应该点击哪个作用在哪个上面

参考路由 API 文档：<https://router.vuejs.org/zh/api/>

8.4.1 tag

`<router-link>` 默认渲染后生成 `<a>` 标签。

- 可在 `<router-link>` 上使用 `tag` 属性，指定渲染后生成其他标签。

```
1 <router-link to="/foo" tag="li">foo</router-link>
2 <!-- 渲染结果 -->
3 <li>foo</li>
```

8.4.2 active-class

`<router-link>` 渲染后生成标签上默认有 CSS 类名：`router-link-active`。

- 可在 `<router-link>` 上使用 `active-class` 属性，指定渲染后生成其他类名。
- 可以通过路由的构造选项 `linkActiveClass` 来全局配置，不用在每个 `<router-link>` 使用 `active-class` 指定生成的类名。

8.4.3 exact

默认情况下，路由地址 `/`、`/foo`、`/bar` 都以 `/` 开头，它们都会去匹配 `/` 地址的 CSS 类名。

- 可在 `<router-link>` 上使用 `exact` 属性开启 CSS 类名精确匹配。

```
1 <!-- 这个链接只会在地址为 / 的时候被激活, -->
2 <router-link to="/" exact>
```

8.4.4 实现高亮显示导航链接

- 重构：AppLeaf.js

```
1 ;(function () {
2
3   window.AppLeaf = {
4     template: `<div class="col-sm-3 col-md-2 sidebar">
5       <ul class="nav nav-sidebar">
6         <!--
7           1. 因为 .active 样式生成在 li 标签上而不 a 标签上,
8           可以通过在 router-link 标签上使用 tag 属性, 让它渲染为 li 标签
9           2. 可在每个 router-link 标签上使用 active-class 属性, 指定渲染后标签上面的 CSS 类名,
10            或者在 VueRouter 实例(router.js)中使用选项 linkActiveClass 全局配置生成的 CSS 类名
11           3. 通过上面,发现不管点击哪一个链接, / 路由(首页)一直有高亮背景,
12            因为 /foo 都会去模糊匹配 / 和 /foo, 所以 / 一直有高亮
13            可在router-link 标签上使用 exact 属性, 使用开启精确匹配
14         -->
15
16         <router-link to="/" tag="li" exact>
17           <a>首页</a>
18         </router-link>
19         <router-link to="/news" tag="li" >
20           <a>新闻管理</a>
21         </router-link>
22         <router-link to="/about" tag="li">
23           <a>关于我们</a>
24         </router-link>
```

```
25     </ul>
26   </div>`
27 }
28
29 })()
```

- 重构 router.js

```
1 ;(function () {
2   window.router = new VueRouter({
3     // 全局配置 router-link 标签生成的 CSS 类名
4     linkActiveClass: 'active',
5     routes: [
6       {path: '/', component: AppHome},
7       {path: '/news', component: News},
8       {path: '/about', component: About}
9     ]
10  })
11 })()
```

- 重新访问，点击哪个就哪个有背景色

← → ↻ ⓘ 127.0.0.1:5500/vue-08-router/02-bootstrap-ajax-router/index.html#/news

java 梦学谷教育 阿里云 xiaomi 前端 在线教育 遵循 semver 标准 Android

梦学谷

首页

体育

科技

新闻管理

关于我们

- 世界杯开赛啦
- NBA开赛倒计时

8.5 嵌套路由

8.5.1 演示效果



8.5.2 子路由组件

Sport.js 体育栏目组件

Tech.js 科技栏目组件

8.5.3 配置嵌套路由

```
1 {
2   path: '/news',
3   component: News,
4   children: [
5     // 当匹配到 /news/sport 请求时，
6     // 组件 Sport 会被渲染在 News 组件中的 <router-view> 中
7     {
8       path: '/news/sport',
9       component: Sport
10    },
11    // 简写方式，等价于 /news/tech 路径，注意前面没有 /，有 / 就是根目录了
12    {
13      path: 'tech',
14      component: Tech
15    },
16    // 点击新闻管理默认选中 新闻，
17    // 就是/news后面没有子路径时，redirect 重定向到 体育
18    {
19      path: '',
20      redirect: '/news/sport'
21    }
22  ]
23 }
```

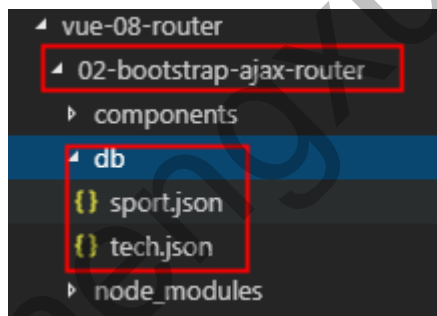
8.5.4 路由跳转链接

```
1 <ul class="nav nav-pills">
2   <router-link to="/news/sport" tag="li">
3     <a >体育</a>
4   </router-link>
5   <router-link to="/news/tech" tag="li">
6     <a >科技</a>
7   </router-link>
8 </ul>
9
10 <!--定义路由出口-->
11 <router-view></router-view>
```

8.6 嵌套路由案例-新闻管理

新闻管理模块中体育和科技栏目，点击不同栏目下方显示不同内容，将体育和科技为 Sport 和 Tech 两个组件，所以将 News.js 中的模板内容区域分别抽取到 Sport.js 和 Tech.js 文件中

- 拷贝 Vue\01-配套源码 下的 db 目录到 vue-08-router\02-bootstrap-ajax-router\ 下



8.6.1 Sport 组件

在 vue-08-router\02-bootstrap-ajax-router\components\home 目录下新建 Sport.js 文件

```
1 ;(function () {
2   const template = `
3     <div><!--注意：不要少了根元素-->
4     <ul>
5       <li v-for="(sport, index) in sportArr" :key="sport.id">
6         <a href="#"> {{sport.title}} </a>
7       </li>
8     </ul>
9
10    <!--详情-->
11    <div class="jumbotron">
12      <h2>世界杯开赛啦</h2>
13      <p>世界杯于今晚8点举行开幕式.....</p>
14    </div>
15  </div>
16  `
17  window.Sport = {
```

```
18   data() {
19     return {
20       sportArr: [],
21     }
22   },
23   //钩子异步加载数据
24   created() {
25     this.getSportArr()
26   },
27   methods: {
28     getSportArr() {
29       axios.get('http://127.0.0.1:5500/vue-08-router/02-bootstrap-ajax-router/db/sport.json')
30       .then(response => { //function (response) {
31         console.log(response.data, this);
32         this.sportArr = response.data
33       })
34       .catch(error => { //function (error) {
35         alert(error.message)
36       })
37     }
38   },
39   template
40 }
41
42 })()
```

8.6.2 Tech 组件

在 vue-08-router\02-bootstrap-ajax-router\components\home 目录下新建 Tech.js 文件

```
1  ;(function () {
2    const template = `
3      <div>
4        <ul>
5          <li v-for="(tech, index) in techArr" :key="tech.id">
6            <span> {{tech.title}} </span>
7            <button class="btn btn-default btn-xs">查看(Push)</button>&nbsp;
8            <button class="btn btn-default btn-xs">查看(replace)</button>
9          </li>
10         </ul>
11         <!--详情-->
12         <div class="jumbotron">
13           <h2>世界杯开赛啦</h2>
14           <p>世界杯于今晚8点举行开幕式.....</p>
15         </div>
16       </div>
17     `
18     window.Tech = {
19       data () {
20         return {
```

```
21     techArr: []
22   }
23 },
24
25   //钩子异步加载数据
26   created() {
27     this.getTechArr()
28   },
29   methods: {
30     getTechArr() {
31       axios.get('http://127.0.0.1:5500/vue-08-router/02-bootstrap-ajax-router/db/tech.json')
32       .then(response => { //function (response) {
33         console.log(response.data, this);
34         this.techArr = response.data
35       })
36       .catch(error => { //function (error) {
37         alert(error.message)
38       })
39     }
40   },
41   template
42 }
43
44 })()
```

8.6.3 配置嵌套路由

在 `vue-08-router\02-bootstrap-ajax-router\router.js` 中配置嵌套路由

```
1 ;(function () {
2   window.router = new VueRouter({
3     // 全局配置 router-link 标签生成的 CSS 类名
4     linkActiveClass: 'active',
5     routes: [
6       {
7         path: '/',
8         component: AppHome
9       },
10      {
11        path: '/news',
12        component: News,
13        children: [
14          // 当匹配到 /news/sport 请求时，
15          // 组件 Sport 会被渲染在 News 组件中的 <router-view> 中
16          {
17            path: '/news/sport',
18            component: Sport
19          },
20          // 简写方式，等价于 /news/tech 路径，注意前面没有 / ,有 / 就是根目录了
21        ]
22      }
23    ]
24  })
25 })()
```

```
22     path: 'tech',
23     component: Tech
24   },
25   //点击新闻管理默认选中 新闻，
26   // 就是/news后面没有子路径时, redirect 重定向到 体育
27   {
28     path: '',
29     redirect: '/news/sport'
30   }
31 ]
32 },
33 {
34   path: '/about',
35   component: About
36 }
37 ]
38 })
39 })()
```

8.6.4 跳转链接和路由渲染出口

在 `vue-08-router\02-bootstrap-ajax-router\components\home\News.js` 中配置

```
1 ;(function () {
2   const template = `
3     <div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
4       <div class="header clearfix">
5         <nav>
6           <ul class="nav nav-pills">
7             <router-link to="/news/sport" tag="li">
8               <a>体育</a>
9             </router-link>
10            <router-link to="/news/tech" tag="li">
11              <a>科技</a>
12            </router-link>
13          </ul>
14        </nav>
15        <hr>
16      </div>
17      <!--定义路由出口-->
18      <router-view></router-view>
19    </div>
20    `
21    window.News = {
22      template
23    }
24  })()
```

8.6.5 index.html 引入 js

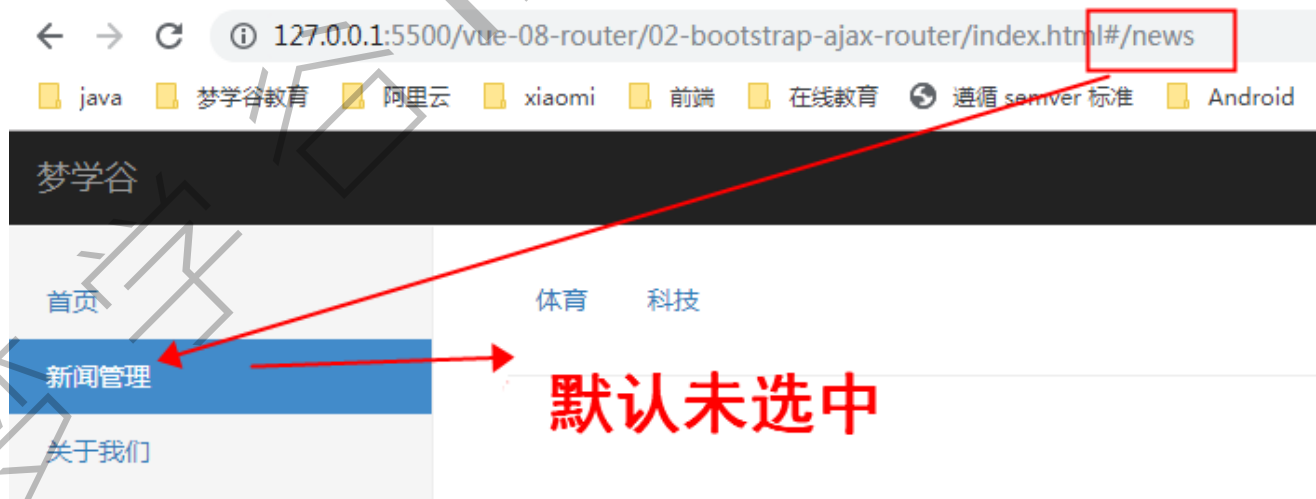
在 index.html 中引入 Sport.js 、 Tech.js ， 注意引入的位置顺序

```
1 <body>
2 <div id="app">
3 </div>
4 <script src="./node_modules/vue/dist/vue.js"></script>
5 <script src="node_modules/vue-router/dist/vue-router.js"></script>
6 <script src="./node_modules/axios/dist/axios.js"></script>
7 <script src="components/AppNavbar.js"></script>
8 <script src="components/AppLeaf.js"></script>
9 <script src="components/home/Dashboard.js"></script>
10 <script src="components/home/Item.js"></script>
11 <script src="components/home/HomeList.js"></script>
12 <script src="components/home/AppHome.js"></script>
13 <script src="components/home/News.js"></script>
14
15 <script src="components/home/Sport.js"></script>
16 <script src="components/home/Tech.js"></script>
17
18 <script src="components/home/About.js"></script>
19 <script src="router.js"></script>
20 <script src="App.js"></script>
21 <script src="main.js"></script>
22 </body>
```

8.6.6 启动测试

访问：<http://127.0.0.1:5500/vue-08-router/02-bootstrap-ajax-router/index.html>

点击新闻管理，右侧没有默认选中



8.6.7 配置默认选中

在 vue-08-router\02-bootstrap-ajax-router\router.js 的 news 子路由中配置默认重定向到新闻

```
{ path: '', redirect: '/news/sport'}
```

```
1 ;(function(){
2   window.router = new VueRouter({
3     // 全局配置 router-link 标签生成的 CSS 类名
4     linkActiveClass: 'active',
5     routes: [
6       { path: '/', component: AppHome },
7       {
8         path: '/news', component: News,
9         children: [
10           // 当匹配到 /news/sport 请求时，
11           // 组件 Sport 会被渲染在 News 组件中的 <router-view> 中
12           { path: '/news/sport', component: Sport },
13           // 简写方式，等价于 /news/tech 路径，注意前面没有 /
14           { path: 'tech', component: Tech },
15           // 点击新闻管理默认选中 新闻，
16           // 就是/news后面没有子路径时，redirect 重定向到 体育
17           { path: '', redirect: '/news/sport' }
18         ]
19       },
20       { path: '/about', component: About }
21     ]
22   })
23 })()
```

再重新测试下



8.7 缓存路由组件与案例

8.7.1 场景与作用

1. 默认情况下，当路由组件被切换后组件实例会销毁，当切换回来时实例会重新创建。
2. 如果可以缓存路由组件实例，切换后不用重新加载数据，可以提高用户体验。

8.7.2 实现缓存路由组件

`<keep-alive>` 可缓存渲染的路由组件实例

```
1 <keep-alive>
2   <router-view></router-view>
3 </keep-alive>
```

8.7.3 案例演示

1. 在 vue-08-router\02-bootstrap-ajax-router\components\home>About.js 的模板中添加一个input输入框，

```
1 ;(function () {
2   const template = `<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
3     <h1>梦学谷-陪你学习，伴你梦想！</h1>
4     <input type="text"/>
5   </div>`
6
7   window.About = {
8     template
9   }
10 })()
```

2. 输入框输入内容后，来回切换组件，实现输入框内容不会被清空。

在 vue-08-router\02-bootstrap-ajax-router\App.js 中配置

```
1 <!-- 配置路由渲染组件出口 -->
2 <keep-alive>
3   <router-view>
4     <h1 slot="dashboard" class="page-header">{{title}}</h1>
5   </router-view>
6 </keep-alive>
```

8.8 路由组件传递数据

8.8.1 传递数据步骤

1. 路由配置

```
1 {  
2   path: '/news/sport',  
3   component: Sport,  
4   children: [  
5     {  
6       path: '/news/sport/detail/:id', // :id 路径变量占位符  
7       component: SportDetail  
8     }  
9   ]  
10 },
```

2. 路由跳转路径

```
1 <!--  
2 要动态拼接值, 则 to 属性值是 JS 表达式,  
3 要写 JS 表达式, 则使用 v-bind 方式绑定属性  
4 注意 + 前面有单引号 "  
5 -->  
6 <router-link :to="'/news/sport/detail/' + sport.id">  
7   {{sport.title}}  
8 </router-link>
```

3. 在路由组件中读取请求参数

```
1 this.$route.params.id
```

8.8.2 体育栏目案例

需求: 在 体育 栏目下点击标题后, 在下方显示详情

8.8.2.1 配置路由

详情显示是属于 Sport 组件的, 在 vue-08-router\02-bootstrap-ajax-router\router.js 中配置

```
1 {  
2   path: '/news',  
3   component: News,  
4   children: [  
5     // 当匹配到 /news/sport 请求时,  
6     // 组件 Sport 会被渲染在 News 组件中的 <router-view> 中  
7     {  
8       path: '/news/sport',  
9       component: Sport,  
10      children: [  
11        { // 配置详情请求路径  
12          path: '/news/sport/detail/:id', // :id 路径变量占位符  
13          component: SportDetail  
14        }  
15      ]  
16     }  
17   ]  
18 }
```

```
15         ]  
16     },  
17     ... 省略  
18 },
```

8.8.2.2 Sport 组件指定路径和渲染出口

在 vue-08-router\02-bootstrap-ajax-router\components\home\Sport.js 中配置

```
1 ;(function () {  
2     const template = `  
3         <div><!--注意：不要少了根元素-->  
4             <ul>  
5                 <li v-for="(sport, index) in sportArr" :key="sport.id">  
6                     <!--<a href="#"> {{sport.title}} </a-->  
7                     <!--  
8                         要动态拼接值, 则 to 属性值是 JS 表达式,  
9                         要写 JS 表达式, 则使用 v-bind 方式绑定属性  
10                        注意 + 前面有单引号 "  
11                    -->  
12                    <router-link :to="/news/sport/detail/" + sport.id">  
13                        {{sport.title}}  
14                    </router-link>  
15                </li>  
16            </ul>  
17  
18            <!--详情-->  
19            <router-view></router-view>  
20        </div>  
21    `;  
22    window.Sport = {  
23        data() {  
24            return {  
25                sportArr: [],  
26            }  
27        },  
28        //钩子异步加载数据  
29        created() {  
30            this.getSportArr()  
31        },  
32        methods: {  
33            getSportArr() {  
34                axios.get('http://127.0.0.1:5500/vue-08-router/02-bootstrap-ajax-router/db/sport.json')  
35                    .then(response => { //function (response) {  
36                        console.log(response.data, this);  
37                        this.sportArr = response.data  
38                    })  
39                    .catch(error => { //function (error) {  
40                        alert(error.message)  
41                    })  
42                }  
43            },  
44            template
```

```
45   }  
46  
47   })()
```

8.8.2.3 SportDetail 详情组件

通过 `this.$route.params.id` 在 SportDetail 模板中可获取路径变量中的 id 值，

在 vue-08-router\02-bootstrap-ajax-router\components\home\SportDetail.js 配置如下

```
1  ;(function () {  
2    const template = `  
3    <div class="jumbotron">  
4      <h1>ID: {{ id }}</h1>  
5      <h2>{{sportDetail.title}}</h2>  
6      <p>{{sportDetail.content}}</p>  
7    </div>  
8    `;  
9    window.SportDetail = {  
10     template,  
11     data () {  
12       return {  
13         id: null,  
14         sportDetail: {}  
15       }  
16     },  
17     created() {  
18       // 注意：  
19       // 1. 是 $route，最后没有 r 字母  
20       // 2. created 钩子只会调用1次，当切换标题列表的路由时，此钩子不会再次调用，  
21       // 所以对应 ID 不会被更新，可以使用 watch 监听 $route 路由的变化。  
22       this.getRportById()  
23     },  
24  
25     methods: {  
26       getRportById() {  
27         // 将路径变量值赋值给id  
28         this.id = this.$route.params.id 0  
29         // 1. 异步获取所有数据  
30         axios.get('http://127.0.0.1:5500/vue-08-router/02-bootstrap-ajax-router/db/sport.json')  
31         .then(response => {  
32           const sportArr = response.data  
33           // 2. 通过 id 获取指定数据  
34           // find 会将满足条件的数据返回，仅返回一条  
35           this.sportDetail = sportArr.find((detail) => {  
36             // this 要代表当前组件实现，则使用箭头函数  
37             return detail.id === this.id  
38           })  
39         })  
40         .catch(error => {  
41           alert(error.message)  
42         })  
43       }  
44     }  
45   }  
46  
47   })()
```

```
44     },
45
46     watch: { // 是对象，不是函数噢
47       // 使用 watch 监听 $route 路由的变化, 获取 ID 值
48       '$route': function () {
49         this.getRportById()
50         console.log('$router改变了', this.id)
51       }
52     }
53
54   }
55
56 })
```

8.8.2.4 index.html 引入 js

- 引入 SportDetail.js

```
1  <body>
2    <div id="app">
3    </div>
4    <script src="./node_modules/vue/dist/vue.js"></script>
5    <script src="node_modules/vue-router/dist/vue-router.js"></script>
6    <script src="./node_modules/axios/dist/axios.js"></script>
7    <script src="components/AppNavbar.js"></script>
8    <script src="components/AppLeaf.js"></script>
9    <script src="components/home/Dashboard.js"></script>
10   <script src="components/home/Item.js"></script>
11   <script src="components/home/HomeList.js"></script>
12   <script src="components/home/AppHome.js"></script>
13   <script src="components/home/News.js"></script>
14   <!-- 引入 SportDetail.js -->
15   <script src="components/home/SportDetail.js"></script>
16   <script src="components/home/Sport.js"></script>
17   <script src="components/home/Tech.js"></script>
18   <script src="components/home/About.js"></script>
19   <script src="router.js"></script>
20   <script src="App.js"></script>
21   <script src="main.js"></script>
22 </body>
```

8.9 程式路由导航

8.9.1 声明式与程式路由

声明式(直接通过 <code><a></code> 标签href指定链接跳转)	编程式(采用 js 代码链接跳转,如 localhost.href)
<code><router-link :to="..."></code>	<code>router.push(...)</code>

8.9.2 编程式路由导航 API

```
1 this.$router.push(path)  相当于点击路由链接(后退1步,会返回当前路由界面)
2 this.$router.replace(path) 用新路由替换当前路由(后退1步,不可返回到当前路由界面)
3 this.$router.back()    后退回上一个记录路由
4 this.$router.go(n)     参数 n 指定步数
5 this.$router.go(-1)    后退回上一个记录路由
6 this.$router.go(1)     向前进下一个记录路由
```

8.9.3 科技栏目案例

8.9.3.1 配置路由

```
1 {
2   path: 'tech',
3   component: Tech,
4   children: [
5     { // 点击 栏目标题列表, 查看详情
6       path: '/news/tech/detail/:id',
7       component: TechDetail
8     }
9   ]
10 },
```

8.9.3.2 Tech 绑定点击事件和渲染出口

- 测试JS函数中push/replace处理路由跳转 和 back/ge后退

```
1 ;(function () {
2   const template = `
3     <div>
4       <ul>
5         <li v-for="(tech, index) in techArr" :key="tech.id">
6           <span> {{tech.title}} </span>
7           <button class="btn btn-default btn-xs" @click="pushTech(tech.id)">查看(Push)</button>&nbsp;
8           <button class="btn btn-default btn-xs" @click="replaceTech(tech.id)">查看(replace)</button>
9         </li>
10      </ul>
11      <button @click="$router.back()">后退</button>
12      <!--详情-->
13      <router-view></router-view>
14    </div>
15  `
16   window.Tech = {
```



```
17   data() {
18     return {
19       techArr: []
20     }
21   },
22
23   //钩子异步加载数据
24   created() {
25     this.getTechArr()
26   },
27   methods: {
28     pushTech(id) {
29       // push 可后退回来
30       // 是 $router, 有字母 r 路由器。用的反单引号 `` 拼接
31       this.$router.push(`/news/tech/detail/${id}`)
32     },
33     replaceTech(id) {
34       // replace 不可后退回来
35       this.$router.replace(`/news/tech/detail/${id}`)
36     },
37
38     getTechArr() {
39       axios.get('http://127.0.0.1:5500/vue-08-router/02-bootstrap-ajax-router/db/tech.json')
40       .then(response => { //function (response) {
41         console.log(response.data, this);
42         this.techArr = response.data
43       })
44       .catch(error => { //function (error) {
45         alert(error.message)
46       })
47     }
48   },
49   template
50 }
51
52 })()
```

8.9.3.3 TechDetail 详情组件

```
1  ;(function () {
2    const template = `
3      <div class="jumbotron">
4        <h2>{{techDetail.title}}</h2>
5        <p>{{techDetail.content}}</p>
6      </div>
7    `
8    window.TechDetail = {
9      template,
10     data() {
11       return {
12         techDetail: {}
13       }
14     }
15   }
```

```
14     },
15
16     created () {
17         // 不要少 this
18         this.getTechById()
19     },
20
21     methods: {
22         // 通过 id 获取详情
23         getTechById () {
24             // 1. 获取路径变量 id 值
25             const id = this.$route.params.id
26             // 2. 获取所有数据
27             axios.get('http://127.0.0.1:5500/vue-08-router/02-bootstrap-ajax-router/db/tech.json')
28                 .then(response => {
29                     const techArr = response.data
30                     // 3. 查找指定id的数据
31                     this.techDetail = techArr.find(tech => {
32                         return tech.id === id
33                     })
34                 })
35                 .catch(error => {
36                     alert(error.message)
37                 })
38         }
39     },
40
41     watch: { // 是对象，不是函数噢
42         // 使用 watch 监听 $route 路由的变化, 获取 ID 值
43         '$route': function () {
44             this.getTechById()
45         }
46     }
47 }
48
49 })()
```

8.9.3.4 index.html 引入 js

- 引入TechDetail.js

```
1 <body>
2 <div id="app">
3 </div>
4 <script src="./node_modules/vue/dist/vue.js"></script>
5 <script src="node_modules/vue-router/dist/vue-router.js"></script>
6 <script src="./node_modules/axios/dist/axios.js"></script>
7 <script src="components/AppNavbar.js"></script>
8 <script src="components/AppLeaf.js"></script>
9 <script src="components/home/Dashboard.js"></script>
10 <script src="components/home/Item.js"></script>
11 <script src="components/home/HomeList.js"></script>
12 <script src="components/home/AppHome.js"></script>
```

```
13 <script src="components/home/News.js"></script>
14 <script src="components/home/SportDetail.js"></script>
15 <!-- 引入 TechDetail.js -->
16 <script src="components/home/TechDetail.js"></script>
17 <script src="components/home/Sport.js"></script>
18 <script src="components/home/Tech.js"></script>
19 <script src="components/home/About.js"></script>
20 <script src="router.js"></script>
21 <script src="App.js"></script>
22 <script src="main.js"></script>
23 </body>
```

第九章 Webpack和Vue-Loader打包资源

Webpack 文档：04-Webpack入门到精通教程.pdf

第十章 Vue-CLI 3.x 脚手架构建项目

10.1 什么是 Vue-CLI

1. Vue-CLI 是 Vue 官方提供的, 用来搭建项目脚手架的工具, 它是 Vue.js 开发的标准工具, 它已经集成了 webpack, 内置好了很多常用配置, 使得我们在使用 Vue 开发项目时更加的标准化。
2. 作用: 通过 Vue-CLI 下载模板项目。
3. 官方文档: <https://cli.vuejs.org/zh/>
4. github站点: <https://github.com/vuejs/vue-cli>

10.2 Vue CLI 安装

Vue CLI 环境要求:

需要 Node.js 8.9+ (推荐 8.11.0+)

1. 全局安装 Vue-CLI

```
1 npm install -g @vue/cli
```

2. 安装成功后, 在命令行可以使用 `vue` 命令, 比如查看当前安装的版本:

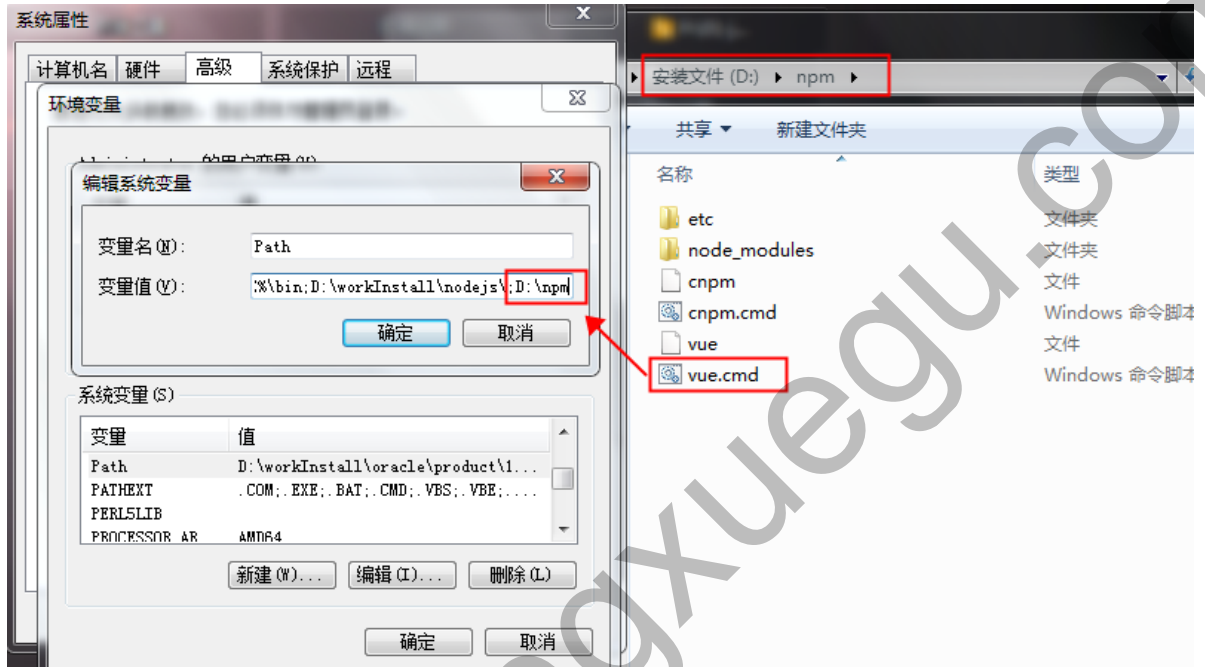
```
1 vue --version
2 # 或者 大写V
3 vue -V
```

- 如果执行上面后, 命令行提示 'vue' 不是内部或外部命令

```
C:\Users\Administrator>vue
'vue' 不是内部或外部命令，也不是可运行的程序
或批处理文件。
```

○ 解决方法：配置环境变量

1. 查看全局安装目录 `npm root -g`
2. 在 我的电脑 进入全局安装目录下，找到 `vue.cmd`
3. 右键计算机，属性—》高级系统设置—》环境变量，将 `vue.cmd` 的路径加入环境变量，点击“确定”



4. 重启命令行窗口，`vue -V` 执行正常。

10.3 基于 Vue-CLI 创建项目

运行 `vue create` 命令来创建一个新项目

```
1 vue create 项目名称
```

10.3.1 创建默认项目

1. 在 `D:\StudentProject\WebStudy` 目录下打开命令行窗口，输入以下命令进行新建项目，项目名是 `vue-cli-demo1`

```
1 vue create vue-cli-demo1
```

```
D:\StudentProject\WebStudy>vue create vue-cli-demo1
```

```
如果淘宝npm相关提示，输入 y 即可
Vue CLI v
? Please pick a preset: <Use arrow keys>
> default <babel, eslint>
Manually select features
```

上下键选择

- 会提示选择默认 (default) 还是手动 (Manually) ,默认的配置非常适合快速创建一个新项目的原型，而手动设置则提供了更多的选项。
- 这里选择 default , 等待一会 (如有提示等待，一直回车执行下去就行)。

```
added 50 packages from 27 contributors, updated 2 packages, moved 7
audited 24203 packages in 66.85s
found 0 vulnerabilities

Running completion hooks...

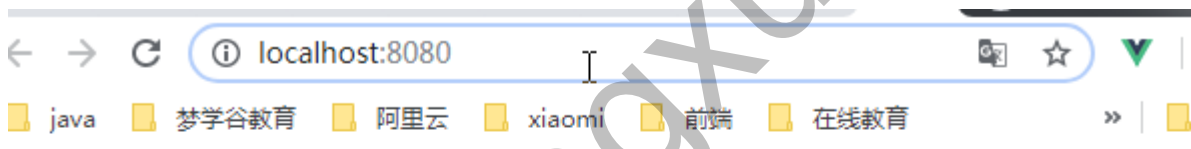
Generating README.md...

Successfully created project vue-cli-demo1.
Get started with the following commands:

$ cd vue-cli-demo1
$ npm run serve

D:\StudentProject\WebStudy>
```

2. 进入： `cd vue-cli-demo1`
3. 运行： `npm run serve`
4. 访问： <http://localhost:8080/>



Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

10.3.2 创建自定义项目

1. 在 `D:\WebProject\webStudy` 目录下打开命令行窗口，输入以下命令进行新建项目，
项目名是 `vue-cli-demo2`

```
1 vue create vue-cli-demo2
```

```
D:\StudentProject\WebStudy\vue-cli-demo1>cd ..  
D:\StudentProject\WebStudy>vue create vue-cli-demo2  
  
Vue CLI v.  
? Please pick a preset:  
  default <babel, eslint>  
> Manually select features
```

选择 Manually select features 手动选择自定义配置进行创建项目

2. 如下图, 根据项目需求, 选择哪些配置选项

- 注意: 按 空格键 是选中或取消, a 键是全选

```
? Please pick a preset: Manually select features  
? Check the features needed for your project: <Press <space> to select, <a> to toggle all, <i> to invert selection>  
>(*) Babel  
  < > TypeScript  
  < > Progressive Web App (PWA) Support  
  < > Router  
  < > Vuex  
  < > CSS Pre-processors  
  <(*) Linter / Formatter  
  < > Unit Testing  
  < > E2E Testing
```

- 1 Babel, (常用) 解决兼容性问题, 支持 ES6 的代码转译成浏览器能识别的代码
- 2 TypeScript, 是一种给 JavaScript 添加特性的语言扩展, 增加了很多功能, 微软开发的
- 3 Progressive Web App (PWA) Support, 渐进式的 Web 应用程序支持
- 4 Router, (常用) 是 vue-router 路由。
- 5 Vuex, 是 Vue.js 应用程序的状态管理模式+库 (常用)。
- 6 CSS Pre-processors, (常用) 支持 CSS 预处理器, Sass/Less 预处理器。
- 7 Linter / Formatter, (常用) 支持代码风格检查和格式化。
- 8 Unit Testing, 支持单元测试。
- 9 E2E Testing, 支持 E2E 测试。

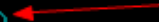
- 选择对应配置选项:

```
? Check the features needed for your project: <Press <space> to select, <a> to toggle all, <i> to invert selection>  
? Check the features needed for your project:  
(*) Babel  
  < > TypeScript  
  < > Progressive Web App (PWA) Support  
(*) Router  
  < > Vuex  
(*) CSS Pre-processors  
(*) Linter / Formatter  
  < > Unit Testing  
  < > E2E Testing
```

- 选择后按回车键, 会提示: 是否使用 history 模式的路由, 按回车即可

```
? Use history mode for router? <Requires proper server setup for index fallback in production> <Y/n> _
```

- 选择CSS预处理器

```
? Pick a CSS pre-processor <PostCSS, Autoprefixer and CSS Modules are supported by default>:  
> Sass/SCSS <with dart-sass>   
  Sass/SCSS <with node-sass>  
  Less  
  Stylus
```

- 选择ESLint + Prettier

```
? Pick a linter / formatter config:  
  ESLint with error prevention only  
  ESLint + Airbnb config  
  ESLint + Standard config  
> ESLint + Prettier 检查与格式化代码
```

- 选择语法检查方式，这里我选择：保存就检测

```
? Pick a linter / formatter config: Prettier  
? Pick additional lint features: <Press <space> to select, <a> to toggle all, <i>  
> to invert selection  
>(*) Lint on save 保存就检测  
< > Lint and fix on commit fix和commit时候检查
```

- 会提示：把babel,postcss,eslint这些配置放哪，我选择：放在独立文件中，然后回来即可

```
? Where do you prefer placing config for Babel, PostCSS, ESLint, etc.? <Use arrow keys>  
> In dedicated config files 分别使用独立文件放置  
  In package.json 放package.json里
```

- 会提示：是否将当前项目设置的配置保存为预配置，方便后面创建项目时，继续使用这套配置？

按回车保存即可，下次创建项目时，就会多有一个选项(vue-cli-demo2)

```
? Save this as a preset for future projects? <y/N> _  
是否记录本次的配置，方便后面创建工程时，继续使用这套配置  
如果要删除 preset (预配置)，在 C:\Users\你的用户名 目录下的 .vuerc 文件中删除
```

- 确定后，等待下载依赖模块

```
Creating project in D:\StudentProject\WebStudy\vue-cli-demo2.  
Initializing git repository...  
Installing CLI plugins. This might take a while...  
[.....] - fetchMetadata: sill pacote range manifest for url-parse@
```

```
added 1167 packages from 916 contributors and audited 23913 packages in 176.445s  
found 0 vulnerabilities  
  
  Invoking generators...  
  Installing additional dependencies...  
  
added 53 packages from 47 contributors, updated 2 packages, moved 9 packages and  
audited 26381 packages in 55.753s  
found 0 vulnerabilities  
  
  Running completion hooks...  
  
  Generating README.md...  
  
Successfully created project vue-cli-demo2.  
Get started with the following commands:  
  
$ cd vue-cli-demo2  
$ npm run serve  
  
D:\StudentProject\WebStudy>
```

○ 运行

```
1 cd vue-cli-demo2  
2 npm run serve
```


[Home](#) | [About](#)



Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

Installed CLI Plugins

[babel](#) [eslint](#)

Essential Links

[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#) [News](#)

10.4 CLI 服务命令

参考：<https://cli.vuejs.org/zh/guide/cli-service.html>

CLI 服务 (`@vue/cli-service`) 是一个开发环境依赖。针对绝大部分应用优化过的内部的 webpack 配置；

- 在一个 Vue CLI 项目中，`@vue/cli-service` 模块安装了一个名为 `vue-cli-service` 的命令。
在 `package.json` 中的 `scripts` 指定 `vue-cli-service` 相关命令。

```
1 "scripts": {  
2   "serve": "vue-cli-service serve --open",  
3   "build": "vue-cli-service build",  
4   "lint": "vue-cli-service lint"  
5 }
```

- `serve` 启动一个开发环境服务器(基于 [webpack-dev-server](#))
 - 修改组件代码后，会自动热模块替换
- `build` 会项目根目录下自动创建一个 `dist/` 目录，打包后的文件都在其中

- 打包后的 js 会自动生成后缀为 .js 和 .map 的文件
 - js文件: 是经过压缩加密的, 如果运行时报错, 输出的错误信息无法准确定位到哪里的代码报错。
 - map文件: 文件比较大, 代码未加密, 可以准确的输出是哪一行哪一列有错。
- lint 使用 Eslint 进行检查并修复代码的规范
 - 比如: 将 main.js 中的格式多加个空格, 执行 npm run lint 后它会自动的去掉多余空格。
- 执行命令方式:

```
1 npm run serve
2 npm run build
3 npm run lint
```

10.5 脚手架项目结构

```
1 |-- node_modules: 存放下载依赖的文件夹
2 |-- public: 存放不会变动静态的文件, 它与src/assets的区别在于, public目录中的文件不被webpack打包处理, 会原样拷贝到dist目录下
3   |-- index.html: 主页面文件
4   |-- favicon.ico: 在浏览器上显示的图标
5 |-- src: 源码文件夹
6   |-- assets: 存放组件中的静态资源
7   |-- components: 存放一些公共组件
8   |-- views: 存放所有的路由组件
9   |-- App.vue: 应用根主组件
10  |-- main.js: 应用入口 js
11 |-- .browserslistrc: 指定了项目可兼容的目标浏览器范围, 对应是package.json 的 browserslist选项
12 |-- .eslintrc.js: eslint相关配置
13 |-- .gitignore: git 版本管制忽略的配置
14 |-- babel.config.js: babel 的配置,即ES6语法编译配置
15 |-- package-lock.json: 用于记录当前状态下实际安装的各个包的具体来源和版本号等, 保证其他人在 npm install 项目时大家的依赖能保证一致。
16 |-- package.json: 项目基本信息,包依赖配置信息等
17 |-- postcss.config.js: postcss一种对css编译的工具, 类似babel对js的处理
18 |-- README.md: 项目描述说明的 readme 文件
```

关于 browsers 的配置如下:

代码	含义
last 2 versions	每一个主流浏览器的最后5个版本
last 2 Chrome versions	谷歌浏览器的最后两个版本
> 2%	市场占有率大于2%
> 2% in US	美国市场占有率大于2%
ie 6-8	ie浏览器6-8
Firefox > 20	火狐版本>20
Firefox < 20	火狐<20
since 2013	2013年之后发布的所有版本
iOS 7	指定IOS 7浏览器

10.6 自定义配置

在项目根目录下创建 `vue.config.js` 文件。

`vue.config.js` 基本常用配置（其他的具体看文档：<https://cli.vuejs.org/zh/config/#vue-config-js>）

```
1 module.exports = {
2   devServer: {
3     port: 8001, // 端口号，如果端口被占用，会自动提升 1
4     open: true, // 启动服务自动打开浏览器
5     https: false, // 协议
6     host: "localhost", // 主机名，也可以 127.0.0.1 或 做真机测试时候 0.0.0.0
7   },
8
9   lintOnSave: false, // 默认 true，警告仅仅会被输出到命令行，且不会使得编译失败。
10  outputDir: "dist", // 默认是 dist，存放打包文件的目录，
11  assetsDir: "assets", // 存放生成的静态资源 (js、css、img、fonts) 的 (相对于 outputDir) 目录
12  indexPath: "out/index.html", // 默认 index.html，指定生成的 index.html 的输出路径 (相对于 outputDir)。
13  productionSourceMap: false, // 打包时，不生成 .map 文件，加快打包构建
14 };
```

10.7 ESLint 插件

10.7.1 什么是 ESLint

ESLint 是一个语法规则和代码风格的检查工具，可以用来保证写出语法正确、风格统一的代码。

如果我们开启了 ESLint，也就意味着要接受它非常苛刻的语法检查，包括空格不能少些或多些，必须单引不能双引，语句后不可以写分号等等，这些规则其实是可以设置的。

我们作为前端的初学者，最好先关闭这种校验，否则会浪费很多精力在语法的规范性上。如果以后做真正的企业级开发，建议开启。Vue-Cli 中默认是开启的

10.7.2 ESLint 配置介绍

在项目根目录下的 `package.json` 文件中有 `eslintConfig` 选项中配置, 或者 `.eslintrc.js` 配置

```
1 "eslintConfig": { // eslint配置
2   "root": true, // 用来告诉eslint找当前配置文件
3   "env": { // 指定你想启用的环境，下面的配置指定为node环境
4     "node": true
5   },
6   "extends": [
7     "plugin:vue/essential", // 格式化代码插件
8     "eslint:recommended" // 启用推荐规则
9   ],
10  "rules": { //新增自定义规则
11    "semi": [1, 'never'], // 禁止用分号
12    "indent": [2, 2] // 缩进采用2个空格
13  },
14  "parserOptions": {
15    "parser": "babel-eslint" //用来指定eslint解析器的
16  }
17 },
```

10.7.3 自定义语法规则

参考：

Vue Github文档：<https://github.com/vuejs/vue-docs-zh-cn/blob/master/vue-cli-plugin-eslint/README.md>

ESLint 官方规则：<https://cn.eslint.org/docs/rules/>

1. 语法规则写在 `package.json` 或 `.eslintrc.js` 文件中 `rules` 选项中, 语法如下：

```
1 rules: {
2   "规则名": [错误等级值, 规则配置]
3 }
```

规则值：

```
1 "off" 或者 0 // 关闭规则
2 "warn" 或者 1 // 打开规则，作为警告（信息打印黄色字体）
3 "error" 或者 2 // 打开规则，作为错误（信息打印红色字体）
```

2. 常用 ESLint 规则配置参数

```
1 "no-alert": 0, //禁止使用alert confirm prompt
2 "no-array-constructor": 2, //禁止使用数组构造器
3 "no-bitwise": 0, //禁止使用按位运算符
4 "no-caller": 1, //禁止使用arguments.caller或arguments.callee
5 "no-catch-shadow": 2, //禁止catch子句参数与外部作用域变量同名
6 "no-class-assign": 2, //禁止给类赋值
7 "no-cond-assign": 2, //禁止在条件表达式中使用赋值语句
8 "no-console": 2, //禁止使用console
9 "no-const-assign": 2, //禁止修改const声明的变量
10 "no-constant-condition": 2, //禁止在条件中使用常量表达式 if(true) if(1)
11 "no-continue": 0, //禁止使用continue
12 "no-control-regex": 2, //禁止在正则表达式中使用控制字符
13 "no-debugger": 2, //禁止使用debugger
14 "no-delete-var": 2, //不能对var声明的变量使用delete操作符
15 "no-div-regex": 1, //不能使用看起来像除法的正则表达式 /=foo/
16 "no-dupe-keys": 2, //在创建对象字面量时不允许键重复 {a:1,a:1}
17 "no-dupe-args": 2, //函数参数不能重复
18 "no-duplicate-case": 2, //switch中的case标签不能重复
19 "no-else-return": 2, //如果if语句里面有return,后面不能跟else语句
20 "no-empty": 2, //块语句中的内容不能为空
21 "no-empty-character-class": 2, //正则表达式中的[]内容不能为空
22 "no-empty-label": 2, //禁止使用空label
23 "no-eq-null": 2, //禁止对null使用==或!=运算符
24 "no-eval": 1, //禁止使用eval
25 "no-ex-assign": 2, //禁止给catch语句中的异常参数赋值
26 "no-extend-native": 2, //禁止扩展native对象
27 "no-extra-bind": 2, //禁止不必要的函数绑定
28 "no-extra-boolean-cast": 2, //禁止不必要的bool转换
29 "no-extra-parens": 2, //禁止非必要的括号
30 "no-extra-semi": 2, //禁止多余的冒号
31 "no-fallthrough": 1, //禁止switch穿透
32 "no-floating-decimal": 2, //禁止省略浮点数中的0 .5 3.
33 "no-func-assign": 2, //禁止重复的函数声明
34 "no-implicit-coercion": 1, //禁止隐式转换
35 "no-implied-eval": 2, //禁止使用隐式eval
36 "no-inline-comments": 0, //禁止行内备注
37 "no-inner-declarations": [2, "functions"], //禁止在块语句中使用声明（变量或函数）
38 "no-invalid-regexp": 2, //禁止无效的正则表达式
39 "no-invalid-this": 2, //禁止无效的this，只能用在构造器，类，对象字面量
40 "no-irregular-whitespace": 2, //不能有不规则的空格
41 "no-iterator": 2, //禁止使用__iterator__ 属性
42 "no-label-var": 2, //label名不能与var声明的变量名相同
43 "no-labels": 2, //禁止标签声明
44 "no-lone-blocks": 2, //禁止不必要的嵌套块
45 "no-lonely-if": 2, //禁止else语句内只有if语句
46 "no-loop-func": 1, //禁止在循环中使用函数（如果没有引用外部变量不形成闭包就可以）
47 "no-mixed-requires": [0, false], //声明时不能混用声明类型
48 "no-mixed-spaces-and-tabs": [2, false], //禁止混用tab和空格
49 "linebreak-style": [0, "windows"], //换行风格
50 "no-multi-spaces": 1, //不能用多余的空格
51 "no-multi-str": 2, //字符串不能用\换行
52 "no-multiple-empty-lines": [1, {"max": 2}], //空行最多不能超过2行
53 "no-native-reassign": 2, //不能重写native对象
```

```
54 "no-negated-in-lhs": 2, // 操作符的左边不能有!
55 "no-nested-ternary": 0, // 禁止使用嵌套的三目运算
56 "no-new": 1, // 禁止在使用new构造一个实例后不赋值
57 "no-new-func": 1, // 禁止使用new Function
58 "no-new-object": 2, // 禁止使用new Object()
59 "no-new-require": 2, // 禁止使用new require
60 "no-new-wrappers": 2, // 禁止使用new创建包装实例，new String new Boolean new Number
61 "no-obj-calls": 2, // 不能调用内置的全局对象，比如Math() JSON()
62 "no-octal": 2, // 禁止使用八进制数字
63 "no-octal-escape": 2, // 禁止使用八进制转义序列
64 "no-param-reassign": 2, // 禁止给参数重新赋值
65 "no-path-concat": 0, // node中不能使用__dirname或__filename做路径拼接
66 "no-plusplus": 0, // 禁止使用++，--
67 "no-process-env": 0, // 禁止使用process.env
68 "no-process-exit": 0, // 禁止使用process.exit()
69 "no-proto": 2, // 禁止使用__proto__属性
70 "no-redeclare": 2, // 禁止重复声明变量
71 "no-regex-spaces": 2, // 禁止在正则表达式字面量中使用多个空格 /foo bar/
72 "no-restricted-modules": 0, // 如果禁用了指定模块，使用就会报错
73 "no-return-assign": 1, // return 语句中不能有赋值表达式
74 "no-script-url": 0, // 禁止使用javascript:void(0)
75 "no-self-compare": 2, // 不能比较自身
76 "no-sequences": 0, // 禁止使用逗号运算符
77 "no-shadow": 2, // 外部作用域中的变量不能与它所包含的作用域中的变量或参数同名
78 "no-shadow-restricted-names": 2, // 严格模式中规定的限制标识符不能作为声明时的变量名使用
79 "no-spaced-func": 2, // 函数调用时 函数名与()之间不能有空格
80 "no-sparse-arrays": 2, // 禁止稀疏数组，[1,,2]
81 "no-sync": 0, // nodejs 禁止同步方法
82 "no-ternary": 0, // 禁止使用三目运算符
83 "no-trailing-spaces": 1, // 一行结束后面不要有空格
84 "no-this-before-super": 0, // 在调用super()之前不能使用this或super
85 "no-throw-literal": 2, // 禁止抛出字面量错误 throw "error";
86 "no-undef": 1, // 不能有未定义的变量
87 "no-undef-init": 2, // 变量初始化时不能直接给它赋值为undefined
88 "no-undefined": 2, // 不能使用undefined
89 "no-unexpected-multiline": 2, // 避免多行表达式
90 "no-underscore-dangle": 1, // 标识符不能以 开头或结尾
91 "no-unneeded-ternary": 2, // 禁止不必要的嵌套 var isYes = answer === 1 ? true : false;
92 "no-unreachable": 2, // 不能有无法执行的代码
93 "no-unused-expressions": 2, // 禁止无用的表达式
94 "no-unused-vars": [2, {"vars": "all", "args": "after-used"}], // 不能有声明后未被使用的变量或参数
95 "no-use-before-define": 2, // 未定义前不能使用
96 "no-useless-call": 2, // 禁止不必要的call和apply
97 "no-void": 2, // 禁用void操作符
98 "no-var": 0, // 禁用var，用let和const代替
99 "no-warning-comments": [1, {"terms": ["todo", "fixme", "xxx"], "location": "start"}], // 不能有警告备注
100 "no-with": 2, // 禁用with
101 "array-bracket-spacing": [2, "never"], // 是否允许非空数组里面有多余的空格
102 "arrow-parens": 0, // 箭头函数用小括号括起来
103 "arrow-spacing": 0, // =>的前/后括号
104 "accessor-pairs": 0, // 在对象中使用getter/setter
105 "block-scoped-var": 0, // 块语句中使用var
106 "brace-style": [1, "1tbs"], // 大括号风格
```



```
107 "callback-return": 1, //避免多次调用回调什么的
108 "camelcase": 2, //强制驼峰法命名
109 "comma-dangle": [2, "never"], //对象字面量项尾不能有逗号
110 "comma-spacing": 0, //逗号前后的空格
111 "comma-style": [2, "last"], //逗号风格，换行时在行首还是行尾
112 "complexity": [0, 11], //循环复杂度
113 "computed-property-spacing": [0, "never"], //是否允许计算后的键名什么的
114 "consistent-return": 0, //return 后面是否允许省略
115 "consistent-this": [2, "that"], //this别名
116 "constructor-super": 0, //非派生类不能调用super，派生类必须调用super
117 "curly": [2, "all"], //必须使用 if{} 中的 {}
118 "default-case": 2, //switch语句最后必须有default
119 "dot-location": 0, //对象访问符的位置，换行的时候在行首还是行尾
120 "dot-notation": [0, { "allowKeywords": true }], //避免不必要的方括号
121 "eol-last": 0, //文件以单一的换行符结束
122 "eqeqeq": 2, //必须使用全等
123 "func-names": 0, //函数表达式必须有名字
124 "func-style": [0, "declaration"], //函数风格，规定只能使用函数声明/函数表达式
125 "generator-star-spacing": 0, //生成器函数*的前后空格
126 "guard-for-in": 0, //for in循环要用if语句过滤
127 "handle-callback-err": 0, //nodejs 处理错误
128 "id-length": 0, //变量名长度
129 "indent": [2, 4], //缩进风格
130 "init-declarations": 0, //声明时必须赋初值
131 "key-spacing": [0, { "beforeColon": false, "afterColon": true }], //对象字面量中冒号的前后空格
132 "lines-around-comment": 0, //行前/行后备注
133 "max-depth": [0, 4], //嵌套块深度
134 "max-len": [0, 80, 4], //字符串最大长度
135 "max-nested-callbacks": [0, 2], //回调嵌套深度
136 "max-params": [0, 3], //函数最多只能有3个参数
137 "max-statements": [0, 10], //函数内最多有几个声明
138 "new-cap": 2, //函数名首行大写必须使用new方式调用，首行小写必须用不带new方式调用
139 "new-parens": 2, //new时必须加小括号
140 "newline-after-var": 2, //变量声明后是否需要空一行
141 "object-curly-spacing": [0, "never"], //大括号内是否允许不必要的空格
142 "object-shorthand": 0, //强制对象字面量缩写语法
143 "one-var": 1, //连续声明
144 "operator-assignment": [0, "always"], //赋值运算符 += -= 什么的
145 "operator-linebreak": [2, "after"], //换行时运算符在行尾还是行首
146 "padded-blocks": 0, //块语句内行首行尾是否要空行
147 "prefer-const": 0, //首选const
148 "prefer-spread": 0, //首选展开运算
149 "prefer-reflect": 0, //首选Reflect的方法
150 "quotes": [1, "single"], //引号类型 ` ` " "
151 "quote-props": [2, "always"], //对象字面量中的属性名是否强制双引号
152 "radix": 2, //parseInt必须指定第二个参数
153 "id-match": 0, //命名检测
154 "require-yield": 0, //生成器函数必须有yield
155 "semi": [2, "always"], //语句强制分号结尾
156 "semi-spacing": [0, { "before": false, "after": true }], //分号前后空格
157 "sort-vars": 0, //变量声明时排序
158 "space-after-keywords": [0, "always"], //关键字后面是否要空一格
159 "space-before-blocks": [0, "always"], //不以新行开始的块(前面要不要有空格)
```

```
160 "space-before-function-paren": [0, "always"],//函数定义时括号前面要不要有空格
161 "space-in-parens": [0, "never"],//小括号里面要不要有空格
162 "space-infix-ops": 0,//中缀操作符周围要不要有空格
163 "space-return-throw-case": 2,//return throw case后面要不要加空格
164 "space-unary-ops": [0, { "words": true, "nonwords": false }],//一元运算符的前/后要不要加空格
165 "spaced-comment": 0,//注释风格要不要有空格什么的
166 "strict": 2,//使用严格模式
167 "use-isnan": 2,//禁止比较时使用NaN，只能用isNaN()
168 "valid-jsdoc": 0,//jsdoc规则
169 "valid-typeof": 2,//必须使用合法的typeof的值
170 "vars-on-top": 2,//var必须放在作用域顶部
171 "wrap-iife": [2, "inside"],//立即执行函数表达式的小括号风格
172 "wrap-regex": 0,//正则表达式字面量用小括号包起来
173 "yoda": [2, "never"]//禁止尤达条件
```

10.7.4 按规则自动修复代码

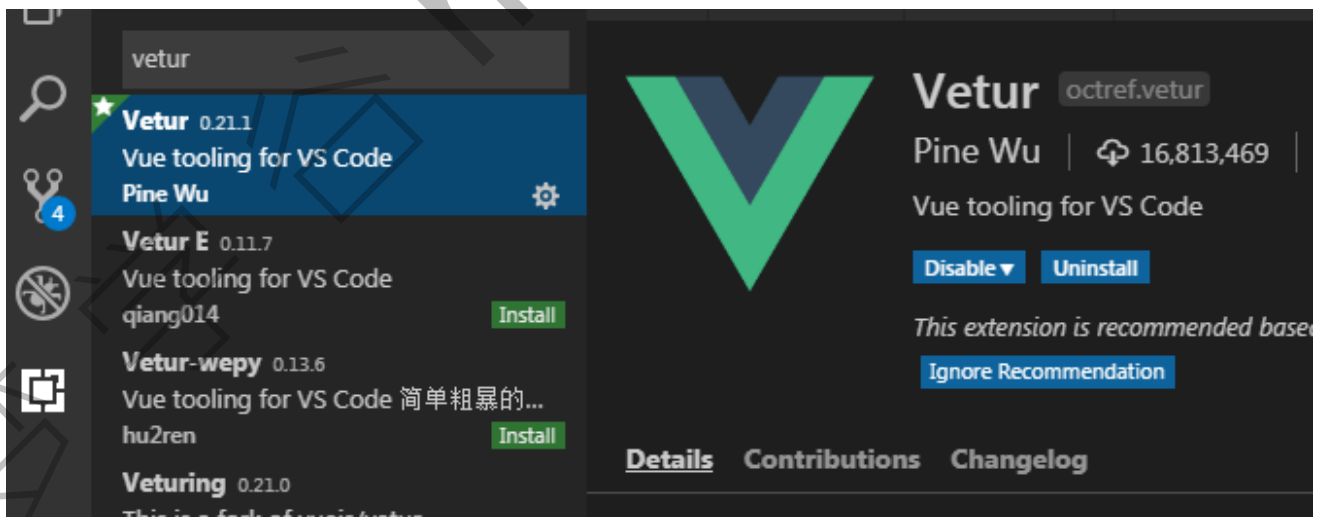
执行下面命令会根据 Eslint 定义的语法规则进行检查，并按语法规则进行自动修复项目中不合规的代码

```
1 npm run lint
```

10.8 创建 .vue 模板代码

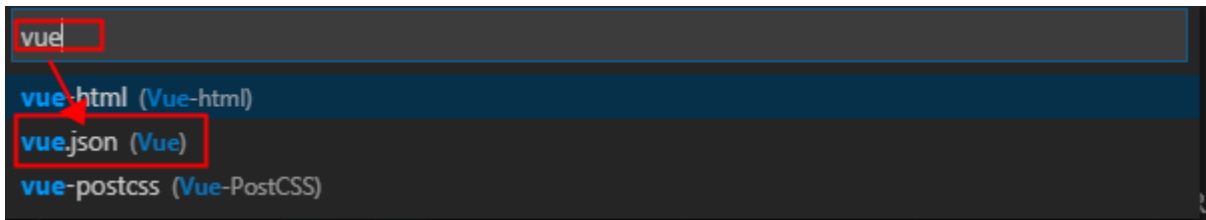
10.8.1 安装插件识别vue文件

插件库中搜索 Vetur 安装，未安装学员记得安装一个



10.8.2 配置模板代码片段

1. 依次选择“File（文件）-> preferences（首选项）-> User Snippets(用户代码片段)”，此时，会弹出一个搜索框，输入vue，如下：



2. 选择 `vue` 后，VS Code 会自动打开一个名字为 `vue.json` 的文件，复制以下内容到这个文件中

`vue.json` 模板内容，请复制网盘下载 `vue.json` 文件中的内容，不要复制pdf文件里的

文件位置：02-参考资源\vue.json

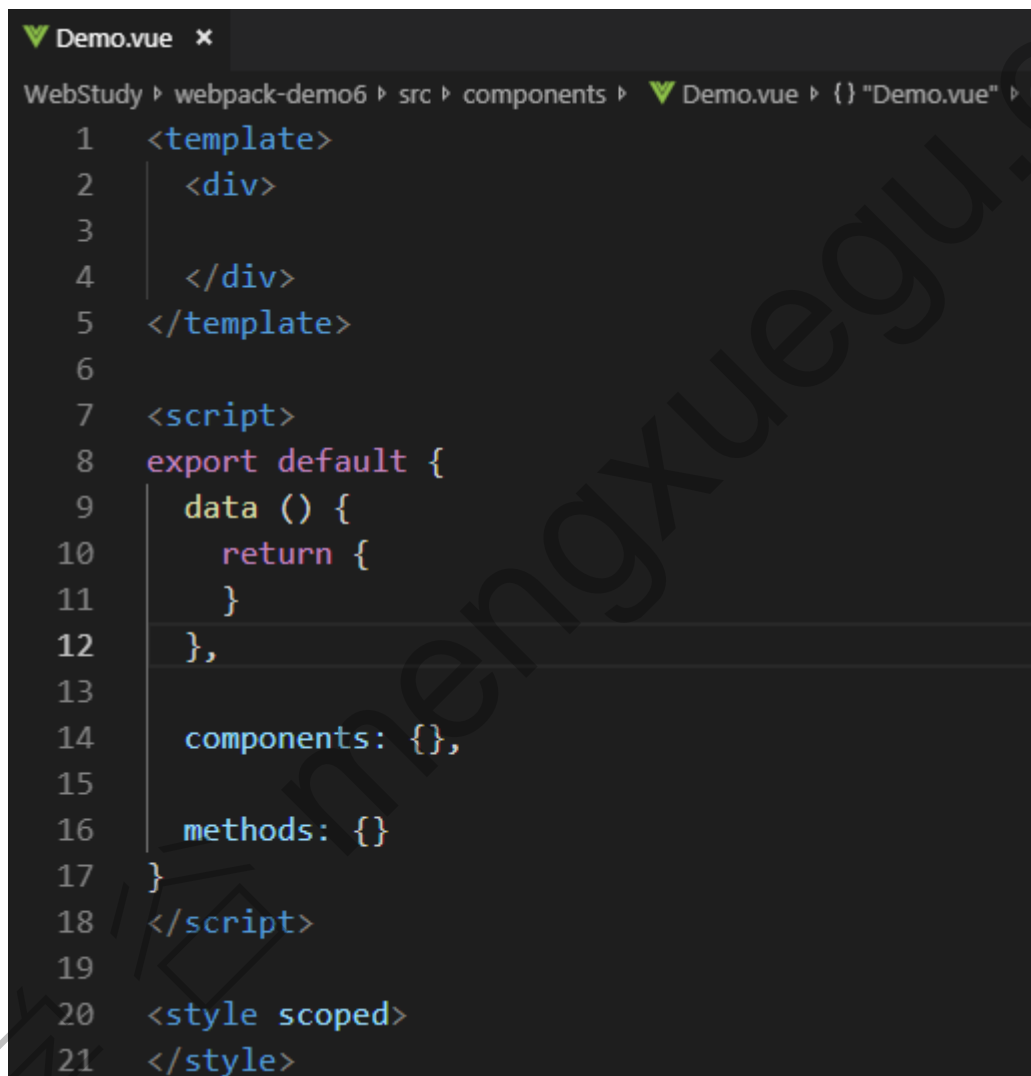
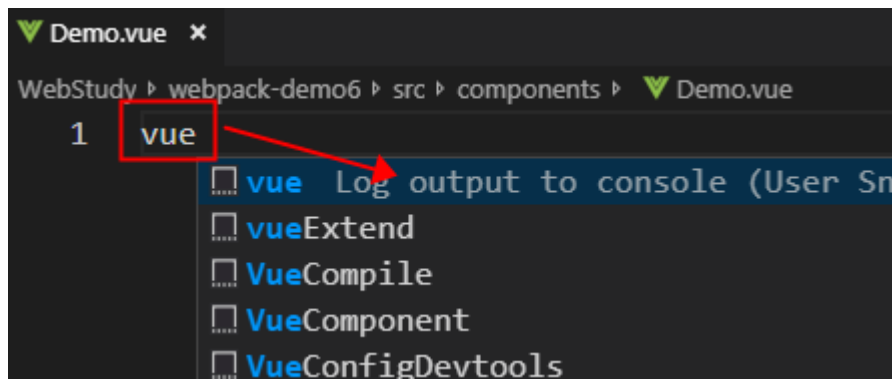
```
1 {  
2   "Print to console": {  
3     "prefix": "vue",  
4     "body": [  
5       "<template>",  
6       " <div>",  
7       "  $0",  
8       " </div>",  
9       "</template>",  
10      "",  
11      "<script>",  
12      "export default {",  
13      "  data () {",  
14      "    return {",  
15      "    }",  
16      "  }",  
17      "",  
18      "  components: {},",  
19      "",  
20      "  methods: {}",  
21      "}",  
22      "</script>",  
23      "",  
24      "<style scoped>",  
25      "</style>",  
26    ],  
27    "description": "Log output to console"  
28  }  
29 }
```

保存后关闭这个文件。

说明：`$0` 生成代码后光标的位置；`prefix` 表示生成对应预设代码的命令

10.8.3 测试

新建一个名为 `Demo.vue` 的文件，输入 `vue`，此时编辑区会有一系列提示，选择 `Log output to console` 这一项



10.9 部署项目

10.9.1 打包项目

进入到打包的项目文件夹下,运行以下命令

```
1 npm run build
```

打包成功后，项目文件夹下会多出 `dist` 目录，那么你可以将 `dist` 目录里构建的内容部署到任何静态文件服务器中。

这里其实是调用了 webpack 来实现打包的。

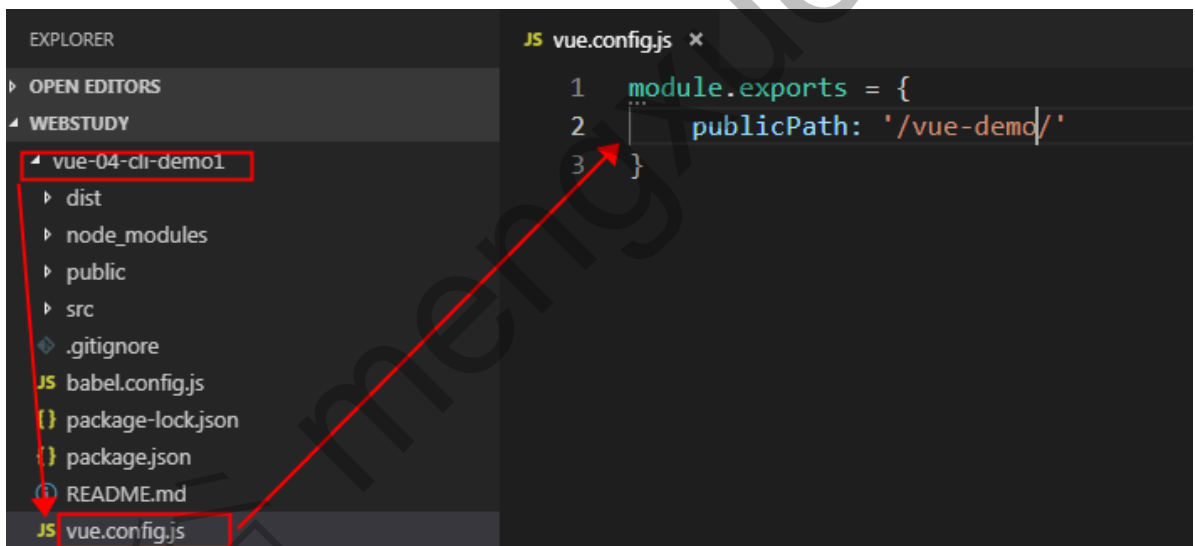
10.9.2 部署项目

- 方式1：使用 Node.js 静态文件服务器 (serve)

```
1 # 安装 server
2 npm install -g serve
3 # 本地部署 dist 目录下的项目
4 serve -s dist
```

- 方式2：使用动态 web 服务器 (tomcat)

1. 在项目的根目录下新建 `vue.config.js` 文件（是根目录，不是src目录）



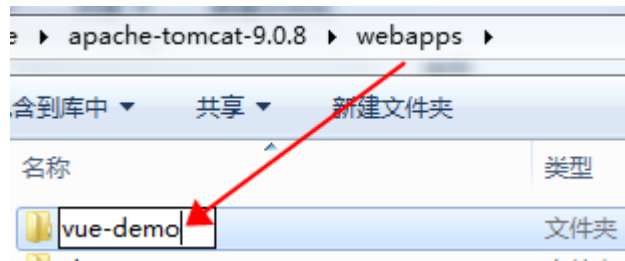
2. 配置打包信息。比如，项目名字为 `vue-demo`

```
1 module.exports = {
2   publicPath: '/vue-demo/'
3 }
```

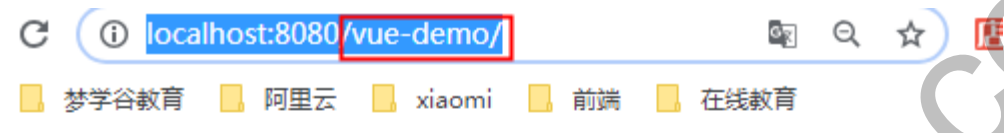
3. 进行重新打包，因为 `dist` 目录下的文件会使用项目名称 `/vue-demo/`

```
1 npm run build
```

4. `dist` 目录拷贝到 `tomcat\webapps` 目录下，然后将 `dist` 目录名 修改为 项目名称 `vue-demo`



5. 打开 `tomcat/bin/startup.bat` 启动服务器（注意之前启动的 8080 端口项目要先停止）
6. 访问：<http://localhost:8080/vue-demo/>



Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

Installed CLI Plugins

[babel](#) [eslint](#)

Essential Links

[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#) [News](#)

Ecosystem

[vue-router](#) [vuex](#) [vue-devtools](#) [vue-loader](#) [awesome-vue](#)