

1- Vue.js 介绍

1.1 Vue.js是什么?

- Vue (读音 /vju:/, 类似于 view) 是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是, Vue 被设计为可以自底向上逐层应用。
- Vue 的核心库只关注视图层, 不仅易于上手, 还便于与第三方库或既有项目整合。另一方面, 当与现代化的工具链以及各种支持类库结合使用时, Vue 也完全能够为复杂的单页应用提供驱动。自底向上逐层应用: 作为渐进式框架要实现的目标就是方便项目增量开发(即插即用)。

官方网站: <https://cn.vuejs.org/>

1.2 为什么使用Vue.js

Vue是一个渐进式的框架, 将Vue作为你应用的一部分嵌入其中, 带来更丰富的交互体验。

解耦视图和数据, 可复用的组件。

2- Vue.js 初体验

2.1 安装Vue的方式

方式一：直接CDN引入

1、在html页面使用script引入vue.js的库即可使用。

```
1 | 远程CDN
2 | <script src="https://cdn.jsdelivr.net/npm/vue@2.6.12/dist/vue.js"></script>
3 | 本地
4 | <script src="vue.min.js"></script>
```

方式二：NPM安装

```
1 | vue-CLI脚手架: 使用vue.js官方提供的CLI脚本架创建vue.js工程.
```

2.2 入门程序

创建一个01-Vuejs初始化目录, 并且在目录下创建 01_vue入门程序.html 文件.

流程步骤

- 1、定义html, 引入vue.js
- 2、定义app div, 此区域作为vue的接管区域
- 3、定义Vue实例, 接管app区域。
- 4、定义model (数据对象)
- 5、在app中展示数据

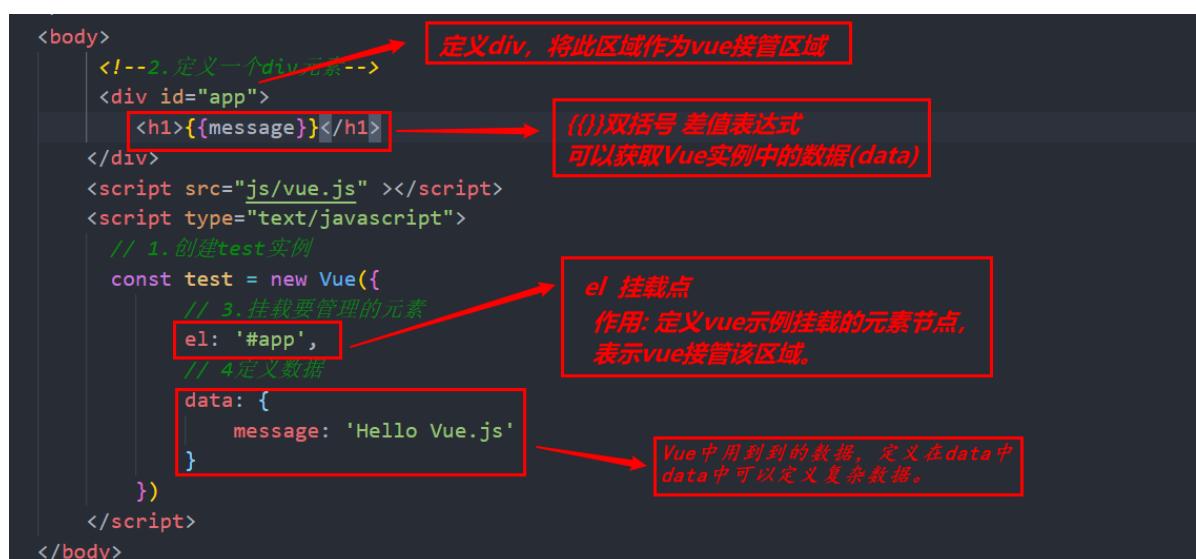
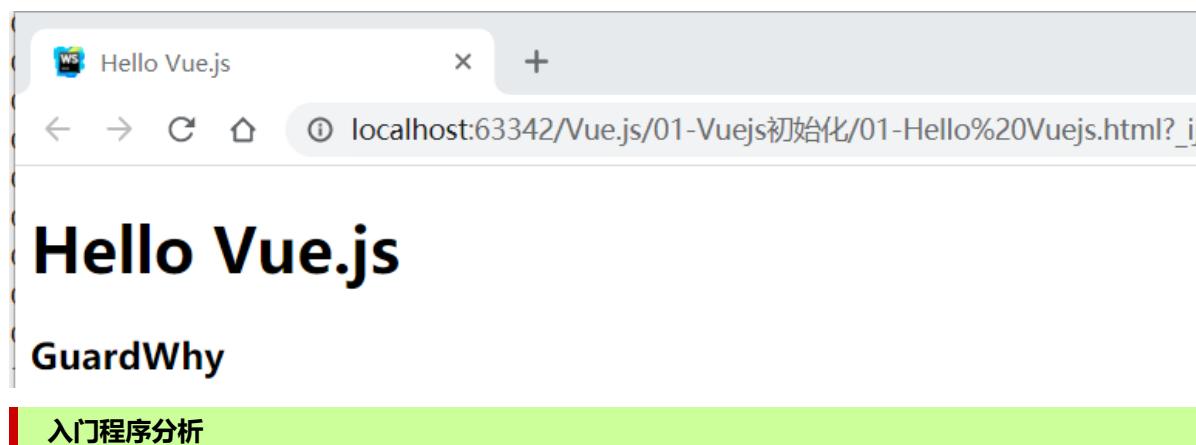
```
1 | <!DOCTYPE html>
2 | <html lang="en">
3 | <head>
4 |   <meta charset="UTF-8">
5 |   <title>Hello vue.js</title>
6 | </head>
```

```

7 <body>
8     <!--2. 定义一个div元素-->
9     <div id="app">
10        <h1>{{message}}</h1>
11        <h3>{{name}}</h3>
12    </div>
13
14    <script src="js/vue.js"></script>
15    <script type="text/javascript">
16        // 1. 创建test实例
17        const test = new Vue({
18            // 3. 挂载要管理的元素
19            el: '#app',
20            // 4. 定义数据
21            data: {
22                message: 'Hello vue.js',
23                name: 'guardwhy'
24            }
25        })
26    </script>
27 </body>
28 </html>

```

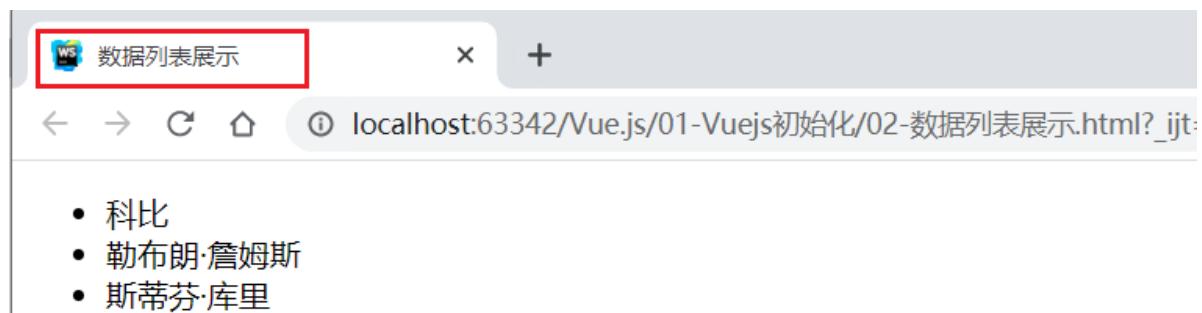
执行结果



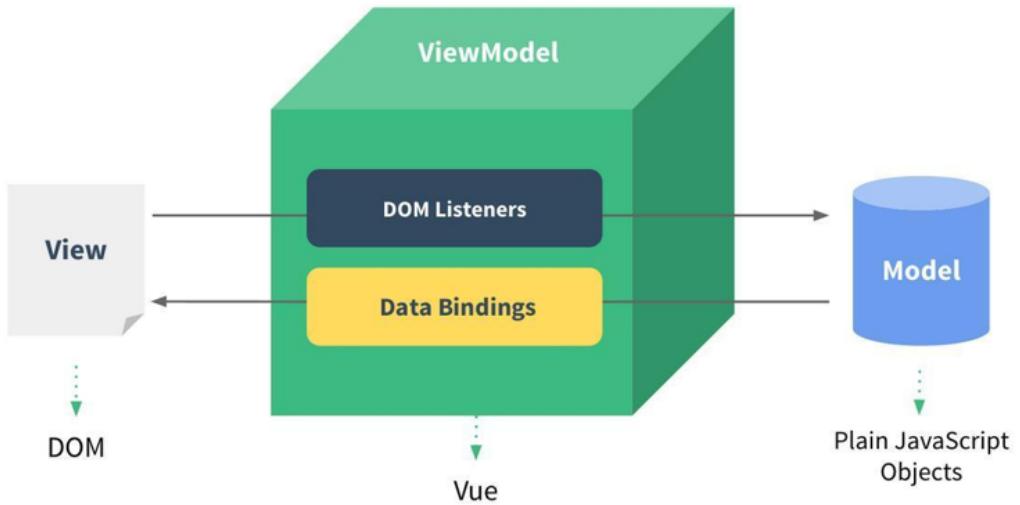
2.3 数据列表展示

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>数据列表展示</title>
6  </head>
7  <body>
8      <div id="test">
9          <ul>
10         <!--遍历操作-->
11         <li v-for="item in starts">{{item}}</li>
12     </ul>
13  </div>
14  <!--引入vue.js文件-->
15  <script src="js/vue.js"></script>
16  <script type="text/javascript">
17      // 创建对象
18      const test = new Vue({
19          el: '#test',
20          data : {
21              message: '你好啊',
22              starts: ['kobe', 'LeBron James', 'Stephen Curry']
23          }
24      })
25  </script>
26  </body>
27  </html>
```

执行结果



2.4 什么是MVVM?



MVVM (Model-View-Viewmodel) 是一种软件架构模式。MVVM有助于将图形用户界面的开发与业务逻辑或后端逻辑（数据模型）的开发分离开来，这是通过置标语言或GUI代码实现的。MVVM的视图模型是一个值转换器，这意味着视图模型负责从模型中暴露（转换）数据对象，以便轻松管理和呈现对象。在这方面，视图模型比视图做得更多，并且处理大部分视图的显示逻辑。视图模型可以实现中介者模式，组织对视图所支持的用例集的后端逻辑的访问。

View层(视图层)

在我们前端开发中，通常就是DOM层。主要的作用是给用户展示各种信息。

Model层(数据层)

数据可能是我们固定的死数据，更多的是来自我们服务器，从网络上请求下来的数据。

在我们计数器的案例中，就是后面抽取出来的obj，当然，里面的数据可能没有这么简单。

ViewModel层(视图模型层)

视图模型层是View和Model沟通的桥梁。一方面它实现了Data Binding，也就是数据绑定，将Model的改变实时的反应到View中。

另一方面它实现了DOM Listener，也就是DOM监听，当DOM发生一些事件(点击、滚动、touch等)时，可以监听到，并在需要的情况下改变对应的Data。

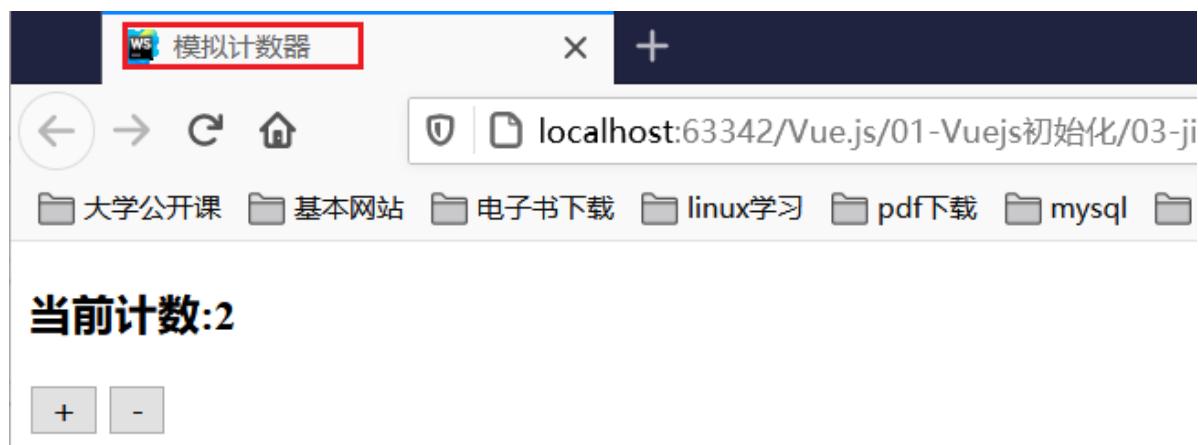
```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>模拟计数器</title>
6      <style>
7          /*
8              MVVM: 前端的架构模式
9              M: Model 负责存储数据
10             V: View 负责页面展示
11             VM: ViewModel 负责业务处理(MVM模式的核心)
12         */
13     </style>
14 </head>
15 <body>
16     <div id="app">
17         <!-- View视图部分!!-->

```

```
18     <h3>当前计数:{{counter}}</h3>
19
20     <!--监听元素的点击事件-->
21     <button v-on:click="add">+</button>
22     <button v-on:click="sub">-</button>
23 </div>
24 <script src="js/vue.js"></script>
25 <script type="text/javascript">
26     // 创建的vue实例,就是VM ViewModel
27     const app = new Vue({
28         el: '#app',
29         // data就是MVVM模式中的 model
30         data: {
31             // 定义当前计数为0
32             counter: 0
33         },
34         // 定义方法
35         methods: {
36             add: function () {
37                 console.log('add被执行');
38                 this.counter++
39             },
40             sub: function () {
41                 console.log('sub被执行了..');
42                 this.counter--
43             }
44         }
45     })
46 </script>
47 </body>
48 </html>
```

执行结果



2.5 计数器的MVVM

```

8     <!-- 创建一个div-->
9      <div id="app">
10     <h3>当前计数:{{counter}}</h3>
11     <!-- 监听元素的点击事件-->
12     <button v-on:click="add">+</button>
13     <button v-on:click="sub">-</button>
14   </div>
15   <script src="js/vue.js"></script>
16   <script type="text/javascript">
17     // 创建对象
18     const app = new Vue({
19       el: '#app',
20       data: {           Model
21         // 定义当前计数为0
22         counter: 0
23       },
24       // 定义方法
25       methods: {
26         add: function () {
27           console.log('add被执行');
28           this.counter++;
29         },
30         sub: function () {
31           console.log('sub被执行了...');
32           this.counter--;
33         }
34       }
35     })
36   </script>
37
38

```

计数器中就有严格的MVVM思想，View依然是的DOM，Model就是抽离出来的obj，ViewModel就是创建的Vue对象实例。

它们之间如何工作呢？

- 首先ViewModel通过Data Binding让obj中的数据实时的在DOM中显示。
- 其次ViewModel通过DOM Listener来监听DOM事件，并且通过methods中的操作，来改变obj中的数据。

3-Vue常用的指令

3.1 Mustache语法

{}: 插值表达式

通常用来获取Vue实例中定义的数据(data)，属性节点中不能够使用插值表达式。

el: 挂载点

类型: String | HTMLElement

定义 Vue实例挂载的元素节点,表示vue接管该区域。

1、Vue的作用范围是什么？

Vue会管理el选项命中的元素,及其内部元素。

2、el选择挂载点时,是否可以使用其他选择器？

可以,但是建议使用 ID选择器。

3、是否可以设置其他的DOM元素进行关联？

可以但是建议选择DIV, 不能使用HTML和Body标签。

data: 数据对象

1. Vue中用到的数据定义在data中。

2. data中可以写复杂类型。
3. 渲染复杂类型数据的时候,遵守js语法。

methods: 方法

类型: {[key: string] : Function}

具体作用: 定义属于Vue的一些方法, 可以在其他调用, 也可以在指令中使用。

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Mustache语法</title>
6  </head>
7  <body>
8      <!--2. 定义一个div元素-->
9      <div id="app">
10         <h3>{{message}}</h3>
11         <!--mustache语法中, 不仅仅可以直接写变量, 也可以写简单的表达式-->
12         <h3>{{firstName + ' ' + lastName}}</h3>
13         <h3>{{counter * 2}}</h3>
14     </div>
15
16     <script src="js/vue.js"></script>
17     <script>
18         // 创建对象
19         const app = new Vue({
20             // 挂载要管理的元素
21             el: '#app',
22
23             // 定义数据
24             data: {
25                 message: 'hello world!',
26                 firstName: 'kobe',
27                 lastName: 'bryant',
28                 counter: '22'
29             }
30         })
31     </script>
32 </body>
33 </html>
```

执行结果

The screenshot shows a browser window with the title "Mustache语法". The address bar displays "localhost:63342/Vue.js/01-Vuejs初始化/". Below the address bar is a navigation bar with icons for back, forward, search, and home. A toolbar below the navigation bar contains links to "大学公开课", "基本网站", "电子书下载", "linux学习", "pdf下载", and "mysql". The main content area of the browser displays three examples of Mustache syntax:

- hello world!**
- kobe bryant**
- 44**

3.2 v-once指令

基本特点

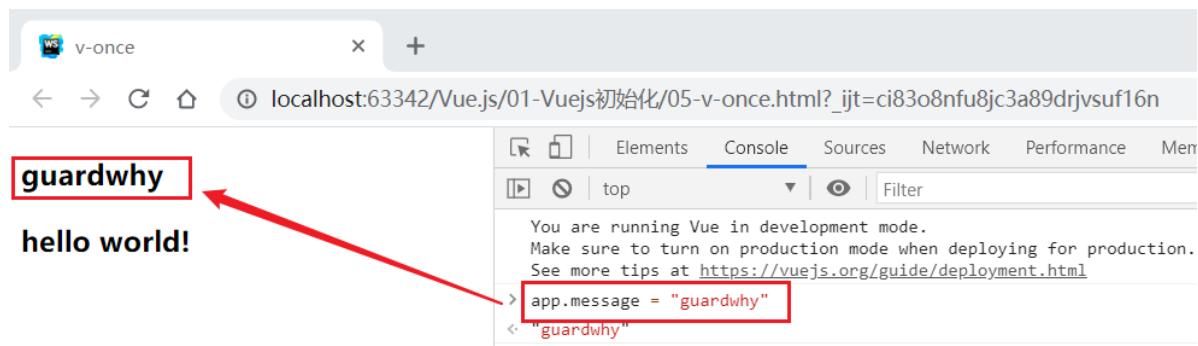
该指令后面不需要跟任何表达式(比如之前的v-for后面是由跟表达式的)。

该指令表示元素和组件只渲染一次，不会随着数据的改变而改变。

代码示例

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>v-once</title>
6  </head>
7  <body>
8      <!--2.定义一个div元素-->
9      <div id="app">
10         <h3>{{message}}</h3>
11
12         <!--表示元素和组件只渲染一次，不会随着数据的改变而改变。-->
13         <h3 v-once>{{message}}</h3>
14     </div>
15
16     <script src="js/vue.js"></script>
17     <script>
18         // 创建对象
19         const app = new Vue({
20             // 挂载要管理的元素
21             el: '#app',
22
23             // 定义数据
24             data: {
25                 message: 'hello world!'
26             }
27         })
28     </script>
29     </body>
30 </html>
```

执行结果



3.3 v-html指令

基本特点

该指令后面往往跟上一个string类型，会将string的html解析出来并且进行渲染。

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>v-html指令</title>
6 </head>
7 <body>
8 <!--2.定义一个div元素-->
9   <div id="app">
10    <h3>{{message}}</h3>
11    <h3>{{url}}</h3>
12
13    <!--该指令后面往往跟上一个string类型，会将string的html解析出来并且进行渲染-->
14    <h3 v-html="url"></h3>
15  </div>
16
17  <script src="js/vue.js"></script>
18  <script>
19    // 创建对象
20    const app = new Vue({
21      // 挂载要管理的元素
22      el: '#app',
23
24      // 定义数据
25      data: {
26        message: 'hello world!',
27        url: '<a href="https://www.bing.com">bing</a>'
28      }
29    })
30  </script>
31 </body>
32 </html>
```

执行结果



hello world!

```
<a href="https://www.bing.com">bing</a>
```

[bing](#)

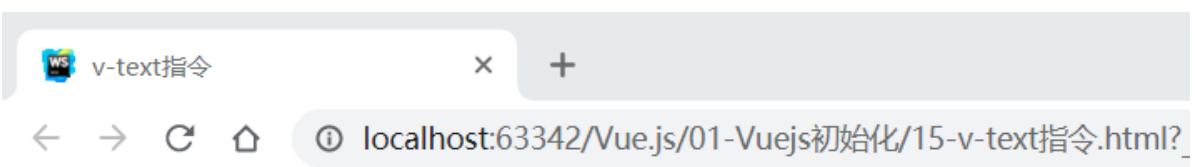
3.4 v-text指令

基本特点

都是用于将数据显示在界面中，v-text通常情况下，接受一个string类型。

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>v-text指令</title>
6  </head>
7  <body>
8      <div id="app">
9          <h3 v-text="message"></h3>
10         <h3>{{message}}</h3>
11     </div>
12
13     <script src="js/vue.js"></script>
14     <script>
15         const app = new Vue({
16             el: '#app',
17             data: {
18                 message: "你好啊, Vue.js!!"
19             }
20         })
21     </script>
22 </body>
23 </html>
```

执行结果



你好啊, Vue.js!!

你好啊, Vue.js!!

3.5 v-pre指令

基本特点

v-pre用于跳过这个元素和它子元素的编译过程。

第一个元素中的内容会被编译解析出来对应的内容，第二个元素中会直接显示{{message}}。

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>v-pre指令</title>
6 </head>
7 <body>
8 <div id="app">
9   <p>{{message}}</p>
10  <p v-pre>{{message}}</p>
11 </div>
12
13 <script src="js/vue.js"></script>
14 <script>
15   const app = new Vue({
16     el: '#app',
17     data: {
18       message: "你好啊, Vue.js!!"
19     }
20   })
21 </script>
22 </body>
23 </html>
```

执行结果



3.6 v-bind 指令

基本特点

作用：动态绑定属性

缩写：:

预期：any (with argument) | Object (without argument)

参数：attrOrProp (optional)

3.6.1 v-bind的基本使用

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>v-bind的基本使用</title>
6 </head>
7 <body>
8 <!--2.定义一个div元素-->
9   <div id="app">
10
11     <!--使用v-bind指令-->
12     
13     <a v-bind:href="aHref">百度一下</a>
14
15     <!--使用语法糖的用法-->
16     
17     <a :href="aHref">百度一下</a>
18   </div>
19
20   <script src="js/vue.js"></script>
21 <script>
22   // 创建对象
23   const app = new Vue({
24     // 挂载要管理的元素
25     el: '#app',
26
27     // 定义数据
28     data: {
29       imageURL: 'https://img11.360buyimg.com/mobilecms/' +
30         's350x250_jfs/t1/20559/1/1424/73138/5c125595E3cbfaa3c8/' +
31         '74fc2f84e53a9c23.jpg!q90!cc_350x250.webp',
32       aHref: 'http://www.baidu.com'
33     }
34   })
35 </script>
36 </body>
37 </html>
```

执行结果



3.6.2 v-bind绑定class

对象语法

用法一：直接通过{}绑定一个类

```
1 | <h2 :class="{ 'active': isActive}">Hello world</h2>
```

用法二：也可以通过判断，传入多个值

```
1 | <h2 :class="{ 'active': isActive, 'line': isLine}">Hello world</h2>
```

用法三：和普通的类同时存在，并不冲突

注：如果isActive和isLine都为true，那么会有title/active/line三个类

```
1 | <h2 class="title" :class="{ 'active': isActive, 'line': isLine}">Hello world</h2>
```

用法四：如果过于复杂，可以放在一个methods或者computed中
注：classes是一个计算属性

```
1 | <h2 class="title" :class="classes">Hello world</h2>
```

代码示例

当数据为某个状态时，字体显示红色，当数据另一个状态时，字体显示黑色。

```
1 | <!DOCTYPE html>
2 | <html lang="en">
3 | <head>
4 |   <meta charset="UTF-8">
5 |   <title>v-bind动态绑定class(对象语法)</title>
6 |   <style>
7 |     .active{
8 |       color: red;
9 |     }
10 |   </style>
11 | </head>
12 | <body>
13 |   <!--2.定义一个div元素-->
14 |   <div id="app">
15 |     <!--<h2 v-bind:class="{类名1: true, 类名2: boolean}">{{message}}</h2>-->
16 |     <h3 class="title" :class="{active: isActive, line: isLine}">
17 |       {{message}}</h3>
18 |       <!--定义一个getClasses()方法-->
19 |       <h3 class="title" :class="getClasses()>{{message}}</h3>
20 |
21 |       <!--绑定事件-->
22 |       <button @click="btnClick">按钮</button>
23 |     </div>
24 |
25 |     <script src="js/vue.js"></script>
26 |     <script>
27 |       // 创建对象
28 |       const app = new Vue({
```

```

28     // 挂载要管理的元素
29     el: '#app',
30
31     // 定义数据
32     data: {
33         message: 'hello vue.js',
34         isActive: true,
35         isLine: true
36     },
37     methods: {
38         btnClick: function () {
39             this.isActive = !this.isActive
40         },
41         /*执行getClasses()方法*/
42         getClasses: function () {
43             return {active: this.isActive, line: this.isLine}
44         }
45     }
46 }
47 </script>
48 </body>
49 </html>

```

执行结果

hello Vue.js

hello Vue.js

按钮

数组语法

结果

app.isActive = false
false
app.isActive = true
true

数组语法的含义是: class后面跟的是一个数组。

用法一：直接通过{}绑定一个类

```
1 | <h2 :class="['active']">Hello world</h2>
```

用法二：也可以传入多个值

```
1 | <h2 :class="['active', 'line']">Hello world</h2>
```

用法三：和普通的类同时存在，并不冲突，注：会有title/active/line三个类。

```
1 | <h2 class="title" :class="['active', 'line']">Hello world</h2>
```

用法四：如果过于复杂，可以放在一个methods或者computed中，注：classes是一个计算属性。

```
1 | <h2 class="title" :class="classes">Hello world</h2>
```

代码示例

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>v-bind动态绑定class(数组语法)</title>
6
7 </head>
8 <body>
9 <!--2.定义一个div元素-->
10 <div id="app">
11   <h3 class="title" :class="[active, line]">{{message}}</h3>
12   <h3 class="title" :class="getClasses()">{{message}}</h3>
13 </div>
14
15 <script src="js/vue.js"></script>
16 <script>
17   // 创建对象
18   const app = new Vue({
19     // 挂载要管理的元素
20     el: '#app',
21
22     // 定义数据
23     data: {
24       message: 'hello vue.js',
25       active: 'kobe',
26       line: 'jmaes'
27     },
28     methods: {
29       getClasses: function () {
30         return [this.active, this.line]
31       }
32     }
33   })
34 </script>
35 </body>
36 </html>
```

执行结果

The screenshot shows a browser window with the URL `localhost:63342/Vue.js/01-Vuejs初始化/09-v-bind动态绑定class.html?_ijt=p11m6qtr8ae4np5qph3j31bvig`. The title bar says "v-bind动态绑定class(数组语法)". The page content displays "hello Vue.js" twice. The developer tools element inspector is open, showing the DOM tree. A red box highlights the second "hello Vue.js" element, which has its class attribute set to "[active, line]" due to the array binding in the Vue.js code. The element inspector also shows the computed value of the class attribute as "kobe jmaes".

3.6.3 v-bind绑定style

对象语法

```
1 | :style="{color: currentColor, fontsize: fontsize + 'px'}"
```

style后面跟的是一个对象类型，对象的key是CSS属性名称，对象的value是具体赋的值，值可以来自于data中的属性。

代码示例

```
1 | <!DOCTYPE html>
2 | <html lang="en">
3 | <head>
4 |   <meta charset="UTF-8">
5 |   <title>v-bind动态绑定style(对象用法)</title>
6 |   <style>
7 |     .title {
8 |       font-size: 50px;
9 |       color: red;
10 |     }
11 |   </style>
12 | </head>
13 | <body>
14 | <!--2.定义一个div元素-->
15 |   <div id="app">
16 |     <!--<h3 :style="{key(属性名): value(属性值)}">{{message}}</h3>-->
17 |     <h5 :style="{fontsize: finalsize + 'px', backgroundColor:
finalColor}">{{message}}</h5>
18 |     <h5 :style="getStyles()>{{message}}</h5>
19 |   </div>
20 |
21 |   <script src="js/vue.js"></script>
22 |   <script>
23 |     // 创建对象
24 |     const app = new Vue({
25 |       // 挂载要管理的元素
26 |       el: '#app',
27 |       // 定义数据
28 |       data: {
29 |         message: 'hello vue.js!!!!',
30 |         finalsize: 30,
31 |         finalColor: 'red',
32 |       },
33 |
34 |       // 方法调用
35 |       methods: {
36 |         getStyles: function () {
37 |           return {fontSize: this.finalsize + 'px', background:
this.finalColor}
38 |         }
39 |       }
40 |     })
41 |   </script>
42 | </body>
43 | </html>
```

执行结果



hello Vue.js!!!

hello Vue.js!!!

数组语法

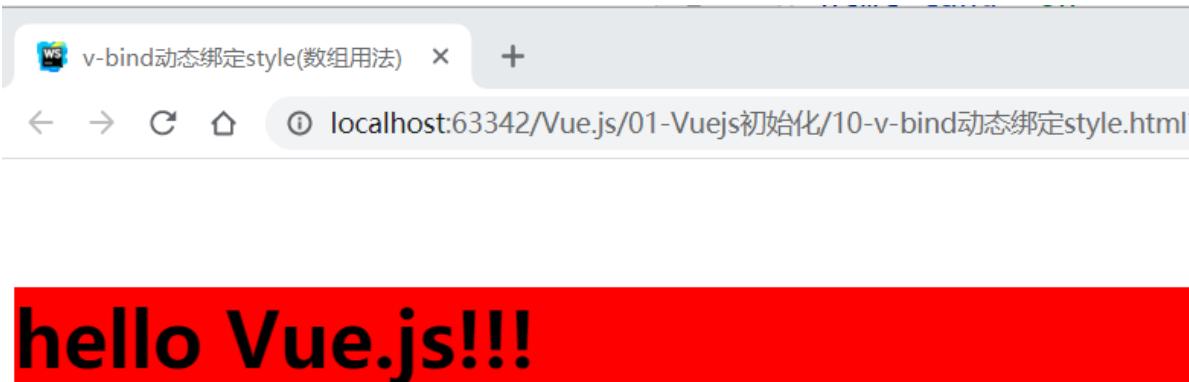
style后面跟的是一个数组类型，多个值以逗号分割即可。

```
1 | <div v-bind:style="[baseStyles, overridingStyles]"></div>
```

代码示例

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>v-bind动态绑定style(数组用法)</title>
6  </head>
7  <body>
8  <!--2.定义一个div元素-->
9      <div id="app">
10         <h5 :style="[baseStyle1,baseStyle2]">{{message}}</h5>
11     </div>
12
13     <script src="js/vue.js"></script>
14     <script>
15         // 创建对象
16         const app = new Vue({
17             // 挂载要管理的元素
18             el: '#app',
19             // 定义数据
20             data: {
21                 message: 'hello Vue.js!!!',
22                 baseStyle1:{backgroundColor: 'red'},
23                 baseStyle2:{fontSize:'40px'},
24             }
25         })
26     </script>
27  </body>
28  </html>
```

执行结果



4- 计算属性

4.1 computed作用

但是在某些情况，我们可能需要对数据进行一些转化后再显示，或者需要将多个数据结合起来进行显示。

减少运算次数，缓存运算结果，运用于重复相同的计算。

4.2 计算属性基本使用

计算属性是写在实例的computed选项中的。

代码示例

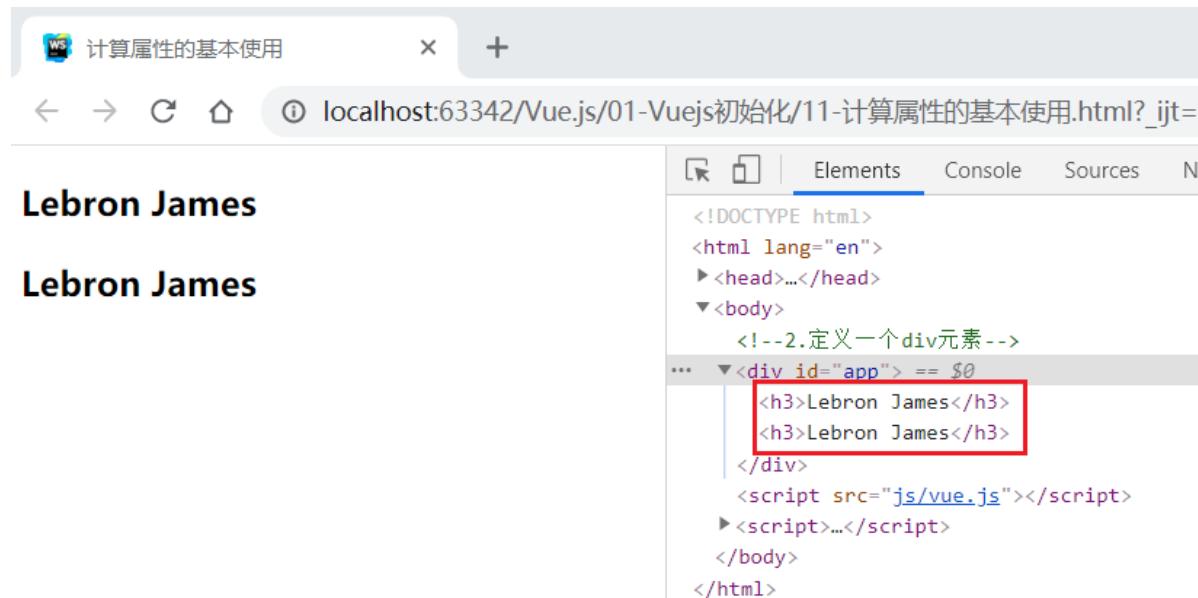
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>计算属性的基本使用</title>
6 </head>
7 <body>
8 <!--2.定义一个div元素-->
9   <div id="app">
10     <h3>{{fullName}}</h3>
11
12     <h3>{{getFullName()}}</h3>
13   </div>
14
15 <script src="js/vue.js"></script>
16 <script>
17   // 创建对象
18   const app = new Vue({
19     // 挂载要管理的元素
20     el: '#app',
21
22     // 定义数据
23     data: {
24       firstName: 'Lebron',
25       lastName: 'James'
26     },
27
28     // 计算属性
29   })
30 
```

```

29     computed: {
30         fullName: function () {
31             return this.firstName + ' ' + this.lastName
32         }
33     },
34
35     // 方法版
36     methods: {
37         getFullName(){
38             return this.firstName + ' ' + this.lastName
39         }
40     }
41 }
42 </script>
43 </body>
44 </html>

```

执行结果



4.3 计算属性复杂使用

计算属性中也可以进行一些更加复杂的操作

代码示例

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>计算属性的复杂使用</title>
6  </head>
7  <body>
8      <!--2.定义一个div元素-->
9      <div id="app">
10         <h3>总价格:{{totalPrice}}</h3>
11     </div>
12
13     <script src="js/vue.js"></script>
14     <script>

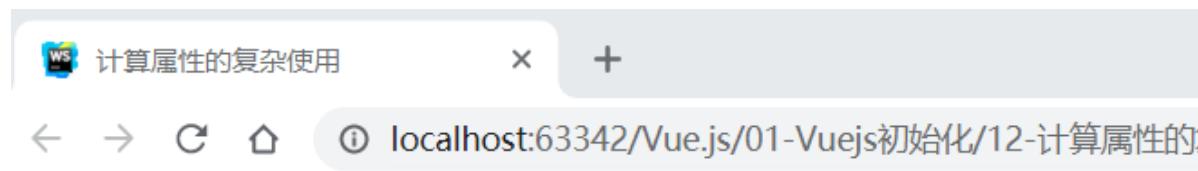
```

```

15     // 创建对象
16     const app = new Vue({
17         // 挂载要管理的元素
18         el: '#app',
19
20         // 定义数据
21         data: {
22             books: [
23                 {id: 100, name: 'Unix编程艺术', price: 119},
24                 {id: 111, name: '代码大全', price: 105},
25                 {id: 112, name: '深入理解计算机原理', price: 98},
26                 {id: 113, name: '现代操作系统', price: 87},
27             ]
28         },
29
30         computed: {
31             totalPrice: function () {
32                 // 定义result值
33                 let result = 0;
34                 /*for(let i=0; i < this.books.length; i++){
35                     result += this.books[i].price
36                 }*/
37
38                 // es6写法
39                 for (let i in this.books) {
40                     result += this.books[i].price;
41                 }
42
43                 // 返回结果值
44                 return result;
45             }
46         }
47     })
48     </script>
49 </body>
50 </html>

```

执行结果



总价格:409

4.4 计算属性setter和getter

注意: 每个计算属性都包含一个getter和一个setter, 计算属性一般是没有set方法的, 只读属性。

代码示例

```

1  <!DOCTYPE html>
2  <html lang="en">
3      <head>

```

```
4  <meta charset="UTF-8">
5  <title>计算属性的setter和getter</title>
6  </head>
7  <body>
8  <!--2.定义一个div元素-->
9  <div id="app">
10 <h3>{{fullName}}</h3>
11 </div>
12
13 <script src="js/vue.js"></script>
14 <script>
15 // 创建对象
16 const app = new Vue({
17 // 挂载要管理的元素
18 el: '#app',
19
20 // 定义数据
21 data: {
22   firstName: 'kobe',
23   lastName: 'Bryant'
24 },
25
26 // 计算属性
27 computed: {
28 // get和set方法
29   fullName: {
30     /* set: function (newValue) {
31       const names = newValue.split(' ');
32       this.firstName = names[0];
33       this.lastName = names[1];
34     }*/
35     get: function () {
36       return this.firstName + ' ' + this.lastName
37     }
38   },
39 }
40 })
41 </script>
42 </body>
43 </html>
```

执行结果

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <!--2.定义一个div元素-->
    <div id="app">
      <h3>kobe Bryant</h3>
    </div>
    <script src="js/vue.js"></script>
    <!--<script>...</script>-->
  </body>
</html>
```

4.5 计算属性和methods的比较

代码示例

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>计算属性和methods的比较</title>
6  </head>
7  <body>
8  <!--2.定义一个div元素-->
9    <div id="app">
10      <!--直接拼接-->
11      <h2>{{firstName}} {{lastName}}</h2>
12      <!--2.通过定义methods-->
13      <h2>{{getFullName()}}</h2>
14
15      <!--3.通过computed-->
16      <h2>{{fullName}}</h2>
17    </div>
18
19    <script src="js/vue.js"></script>
20    <script>
21      // 创建对象
22      const app = new Vue({
23        // 挂载要管理的元素
24        el: '#app',
25
26        // 定义数据
27        data: {
28          firstName: 'kobe',
29          lastName: 'Bryant'
30        },
31      });
32    </script>
```

```

32 // 函数
33 methods: {
34     getFullName: function () {
35         console.log('fullName');
36         // 返回值
37         return this.firstName + ' ' + this.lastName;
38     }
39 },
40
41 // 计算属性
42 computed: {
43     fullName: function () {
44         console.log('fullName');
45         return this.firstName + ' ' + this.lastName
46     }
47 }
48
49 })
50 </script>
51 </body>
52 </html>

```

执行结果

The screenshot shows a browser window with the title "计算属性和methods的比较". The address bar shows "localhost:63342/Vue.js/01-Vuejs初始化/14-计算属性和meth". The developer tools are open, specifically the "Elements" tab under "查看器". A search bar at the top of the tools says "搜索 HTML". Below it, the DOM tree is displayed. A blue bar highlights the root element: `<!DOCTYPE html>`. Underneath, the `body` element is expanded, showing its contents. Inside the `body` element, there is a `div` with the `id` attribute set to "app". This `div` contains three `

` elements, each with the text "kobe Bryant". These three `` elements are all enclosed within a red rectangular selection box, demonstrating that they are all instances of the same computed property.

computed小结

定义函数也可以实现与计算属性相同的效果,都可以简化运算。

不同的是计算属性是基于它们的响应式依赖进行缓存的。只在相关响应式依赖发生改变时它们才会重新求值。

5- 过滤器和侦听器

5.1 filter过滤器

5.1.1 什么是过滤器

过滤器是对即将显示的数据做进一步的筛选处理，然后进行显示，值得注意的是过滤器并没有改变原来的数据，只是在原数据的基础上产生新的数据。

数据加工车间,对值进行筛选加工。

5.1.2 过滤器使用位置

双括号插值内

- 1 {{ msg | filterA }} msg是需要处理的数据，filterA是过滤器，|这个竖线是管道，通过这个管道将数据传输给过滤器进行过滤。
- 2 加工操作

v-bind绑定的值的地方

- 1 <h1 v-bind:id=" msg | filterA"> {{ msg }} </h1>

5.1.3 过滤器

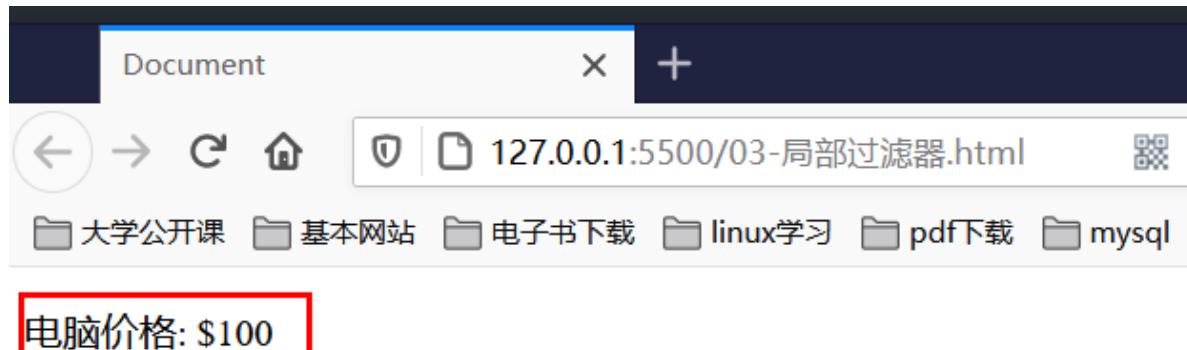
局部过滤器

基本需求: 通过过滤器给电脑价格前面 添加一个符号\$

```
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     <title>Document</title>
7   </head>
8   <body>
9     <div id="app">
10       <!-- 使用插值表达式,调用过滤器 -->
11       <p>电脑价格: {{price | addPrice}}</p>
12     </div>
13   </body>
14   <script src="js/vue.js"></script>
15   <script>
16     //局部过滤器 在vue实例的内部创建filter
17     const app = new Vue({
18       el: "#app", //挂载点
19       data: {
20         //model
21         price: 200,
22       },
23       //局部过滤器
24       filters: {
25         //定义处理函数 value = price
26         addPrice(value) {
27           return "$" + value;
28         },
29       },
30     })
31   </script>
```

```
30     });
31     </script>
32 </html>
```

执行结果



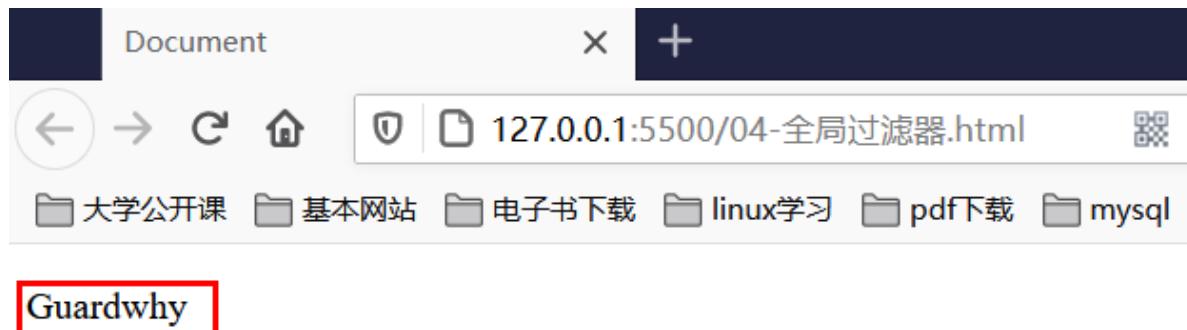
全局过滤器

基本需求: 将用户名开头字母大写。

```
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          <title>Document</title>
7          <style>
8              /*
9                  需求: 将用户名开头字母大写
10                 总结:
11                     1. 过滤器经常被用来处理文本格式化操作
12                     2. 过滤器使用的两个位置: {{}} 插值表达式中, v-bind表达式中
13                     3. 过滤器是通过管道传输数据的 |
14             */
15         </style>
16     </head>
17     <body>
18         <div id="app">
19             <p>{{user.name | changeName}}</p>
20         </div>
21     </body>
22     <script src="js/vue.js"></script>
23     <script>
24
25         //在创建vue实例之前, 创建全局过滤器
26         vue.filter("changeName", function (value) {
27             //将姓名的开头字母大写
28             return value.charAt(0).toUpperCase() + value.slice(1);
29         });
30
31         const app = new vue({
32             el: "#app",
33             data: {
34                 user: { name: "guardwhy" },
35             },
36         });
37     </script>
```

```
36     });
37     </script>
38 </html>
```

执行结果



5.1.4 总结

1. 过滤器常用来处理文本格式化的操作。过滤器可以用在两个地方：双花括号插值和 v-bind 表达式。
2. 过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符号指示

5.2 watch侦听器

5.2.1 什么是侦听器

Vue.js 提供了一个方法 watch，它用于观察Vue实例上的数据变动。

作用: 当你有一些数据需要随着其它数据变动而变动时，可以使用侦听属性。

5.2.2 代码示例

需求: 监听数字变化，实时显示！！！

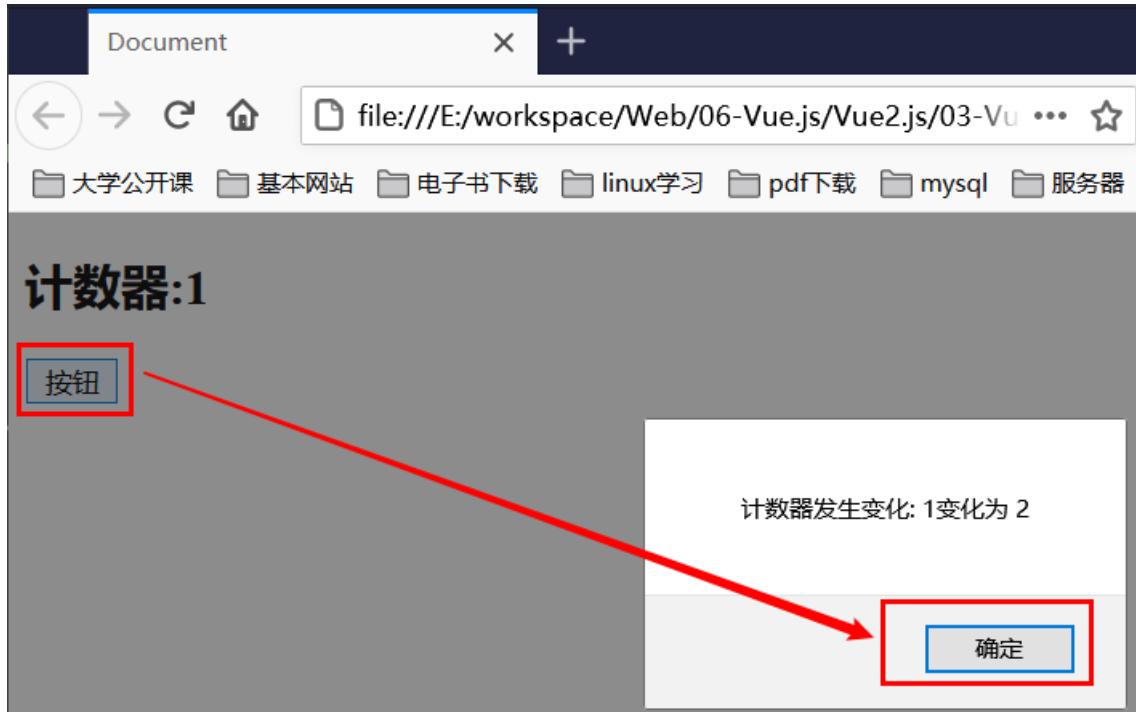
```
1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <meta charset="UTF-8" />
5          <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6          <title>Document</title>
7      </head>
8      <body>
9          <div id="app">
10             <h2>计数器:{{count}}</h2>
11             <input type="button" @click="count++" value="按钮" />
12         </div>
13     </body>
14     <script src="js/vue.js"></script>
15     <script>
16         const app = new Vue({
17             el: "#app",
18             data: {
19                 count: 1,
20             },
21             watch: {
22                 //监测属性的值的变化
23                 count: (num2, num1) =>
```

```

24     //参数1:原来的值 参数2:新的值
25     alert("计数器发生变化: " + num1 + "变化为 " + num2)
26   },
27 }
28 </script>
29 </html>

```

执行结果



6- 基础语法

6.1 v-on指令

基本特点

作用: 绑定事件监听器

缩写: @

预期: Function | Inline Statement | Object

参数: event

6.1.1 v-on的基本使用

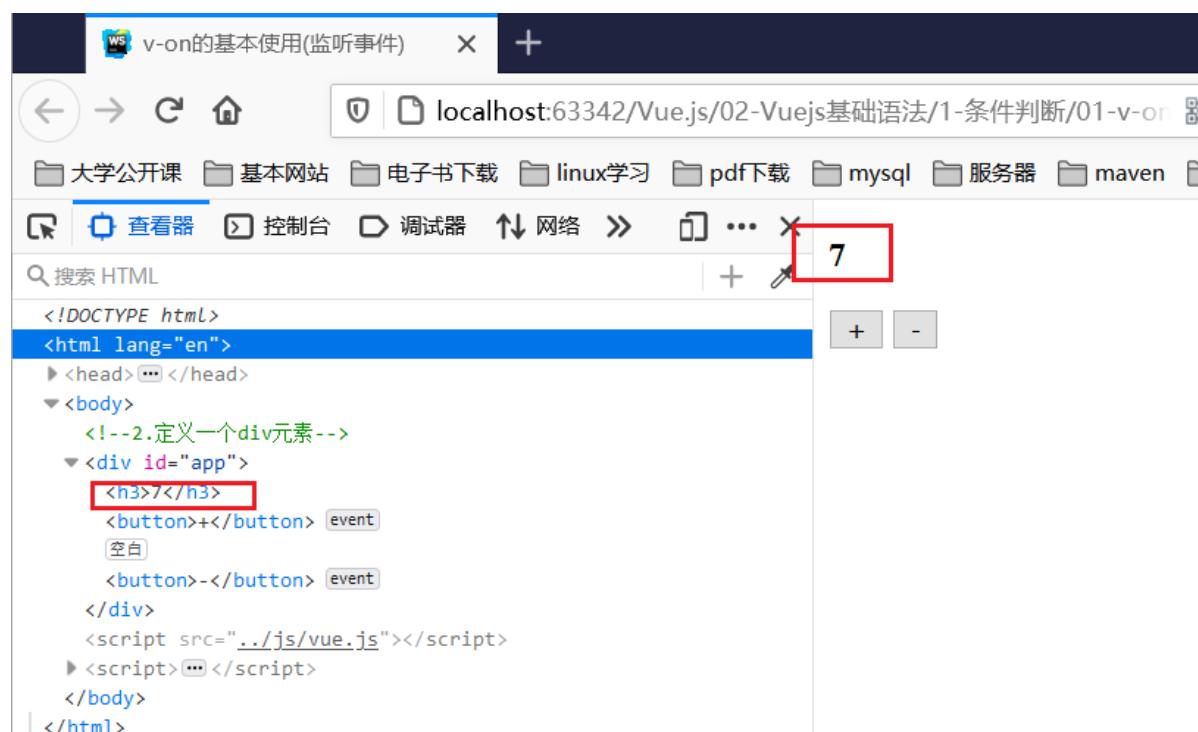
```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>v-on的基本使用(监听事件)</title>
6  </head>
7  <body>
8    <!--2. 定义一个div元素-->
9    <div id="app">
10      <h3>{{counter}}</h3>
11
12      <!--事件绑定语法糖-->
13      <button @click="increment">+</button>
14      <button @click="decrement">-</button>

```

```
15      <!--普通写法-->
16      <!--
17      <button v-on:click="increment">+</button>
18      <button v-on:click="decrement">-</button>
19      -->
20  </div>
21
22
23  <script src="../../js/vue.js"></script>
24  <script>
25      // 创建对象
26      const app = new Vue({
27          // 挂载要管理的元素
28          el: '#app',
29
30          // 定义数据
31          data: {
32              counter: 0
33          },
34          methods: {
35              increment(){
36                  this.counter++
37              },
38              decrement(){
39                  this.counter--
40              }
41          }
42      })
43  </script>
44  </body>
45  </html>
```

执行结果



6.1.2 v-on参数个数

注意事项

- 当通过methods中定义方法，以供@click调用时，需要注意参数问题。
- 如果该方法不需要额外参数，那么方法后的()可以不添加，如果方法本身中有一个参数，那么会默认将原生事件event参数传递进去。
- 如果需要同时传入某个参数，同时需要event时，可以通过\$event传入事件。

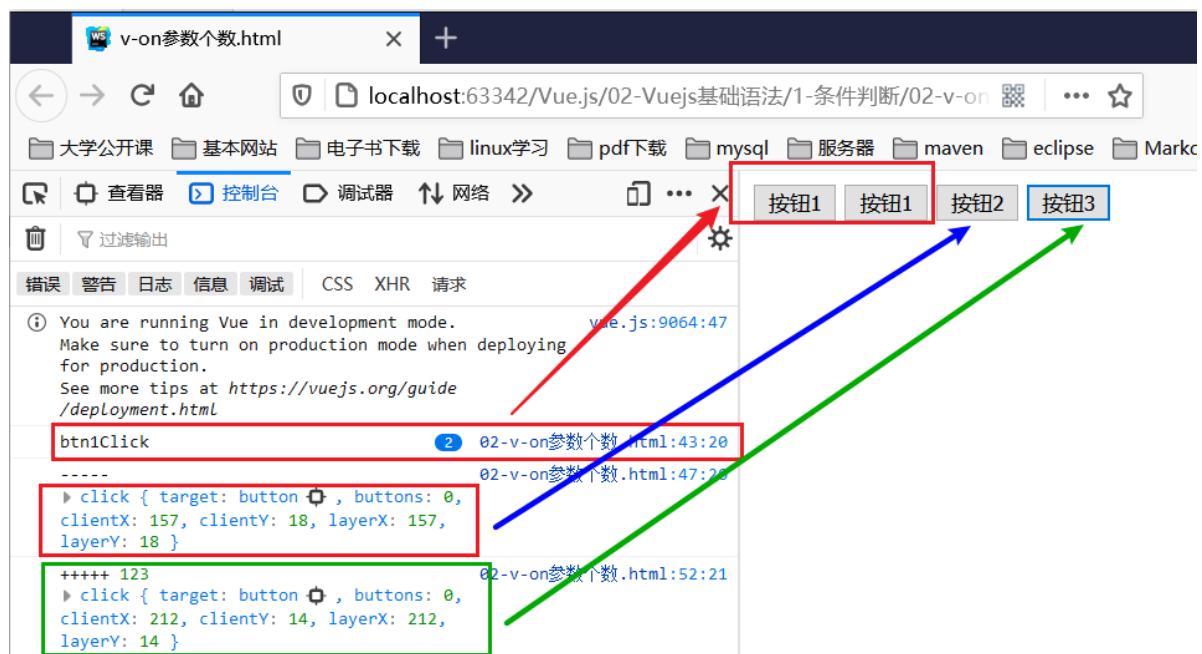
```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>v-on参数个数</title>
6  </head>
7  <body>
8      <!--2. 定义一个div元素-->
9      <div id="app">
10         <!--事件调用的方法没有参数-->
11         <button @click="btn1Click()">按钮1</button>
12         <button @click="btn1Click">按钮1</button>
13
14
15         <!--如果函数需要参数，但是没有传入，那么函数的形参为undefined-->
16         <!-- <button @click="btn2Click()">按钮2</button>-->
17
18         <!--在事件定义时，写方法时省略了小括号，但是方法本身是需要一个参数的。
19         这个时候，Vue会默认将浏览器生产的event事件对象作为参数传入到方法
20         -->
21         <button @click="btn2Click">按钮2</button>
22
23         <!--3. 方法定义时，我们需要event对象，同时又需要其他参数-->
24         <!-- 在调用方式，如何手动的获取到浏览器参数的event对象：$event-->
25         <button @click="btn3Click(abc, $event)">按钮3</button>
26     </div>
27
28     <script src="../../js/vue.js"></script>
29     <script>
30         // 创建对象
31         const app = new Vue({
32             // 挂载要管理的元素
33             el: '#app',
34
35             // 定义数据
36             data: {
37                 message: 'hello world!',
38                 abc: 123
39             },
40             methods: {
41                 btn1Click(){
42                     // 输出结果
43                     console.log("btn1Click");
44                 },
45                 btn2Click(event){
46                     // 输出结果
47                     console.log('----', event);
48                 },
49             }
50         }
51     </script>
```

```

49
50     btn3Click(abc, event){
51         // 输出结果
52         console.log('+++++', abc, event);
53     }
54 }
55 })
56 </script>
57 </body>
58 </html>

```

执行结果



6.1.3 v-on的修饰符

常用的修饰符

.stop - 调用 event.stopPropagation()。

```

1 <!--停止冒泡-->
2 <button @click.stop="doThis"></button>

```

.prevent - 调用 event.preventDefault()。

```

1 <!--阻止默认行为-->
2 <button @click.prevent="doThis"></button>
3 <!--阻止默认行为，没有表达式-->
4 <form @submit.prevent></form>
5 <!--串联修饰符-->
6 <button @click.stop.prevent="doThis"></button>

```

.{keyCode | keyAlias} - 只当事件是从特定键触发时才触发回调。

```
1 | <!--键修饰符，键别名-->
2 | <input @keyup.enter = "onEnter">
3 | <!--键修饰符，键代码-->
4 | <input @keyup.13="onEnter">
```

.once - 只触发一次回调。

```
1 | <!--点击回调只会触发一次-->
2 | <button @click.once="doThis"></button>
```

代码示例

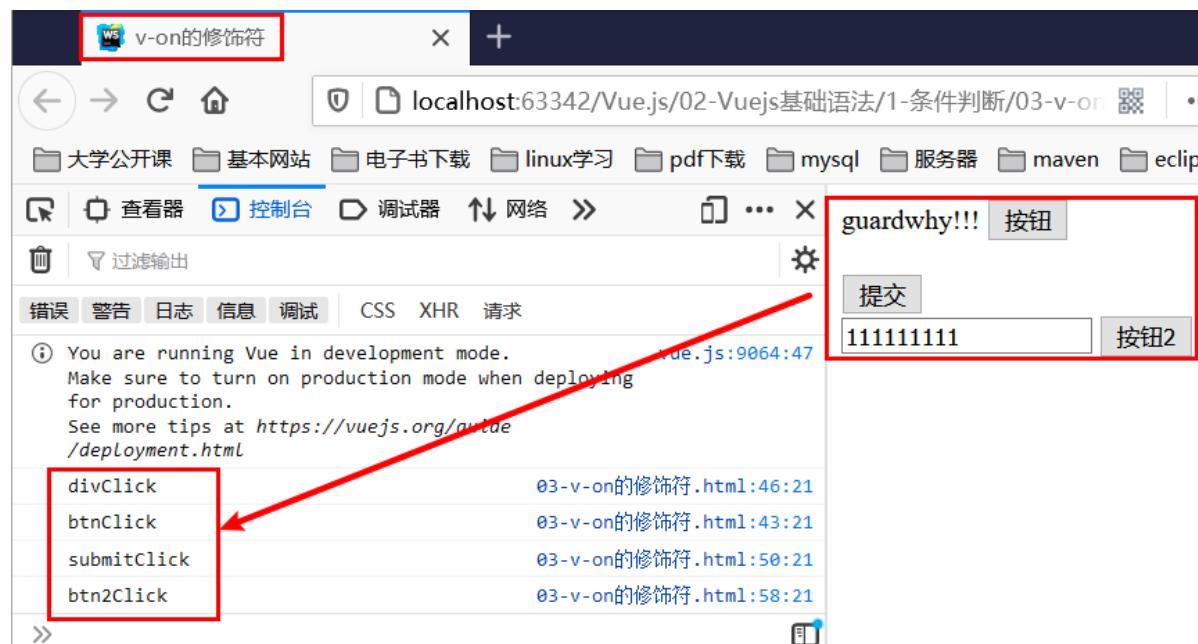
```
1 | <!DOCTYPE html>
2 | <html lang="en">
3 | <head>
4 |   <meta charset="UTF-8">
5 |   <title>v-on的修饰符</title>
6 | </head>
7 | <body>
8 |   <div id="app">
9 |     <!-- 1. 阻止冒泡事件.stop -->
10 |     <div @click="divClick">
11 |       guardwhy!!!
12 |       <button @click.stop="btnClick">按钮</button>
13 |     </div>
14 |
15 |     <!-- 2. 阻止默认事件 -->
16 |     <br/>
17 |     <form action="taobao">
18 |       <input type="submit" value="提交" @click.prevent="submitClick">
19 |     </form>
20 |
21 |     <!--3. 监听某个事件的键帽-->
22 |     <input type="text" @keyup.enter="keyUp">
23 |
24 |     <!--4. .once修饰符的使用-->
25 |     <button @click.once="btn2Click">按钮2</button>
26 |   </div>
27 |
28 |   <script src="../js/vue.js"></script>
29 |   <script>
30 |     // 创建对象
31 |     const app = new Vue({
32 |       // 挂载要管理的元素
33 |       el: '#app',
34 |
35 |       // 定义数据
36 |       data: {
37 |         message: 'hello world!'
38 |       },
39 |
40 |       // 方法
41 |       methods: {
42 |         btnClick() {
43 |           console.log("btnClick");
44 |         },
45 |       }
46 |     });
47 |   </script>
```

```

45     divclick() {
46         console.log('divclick');
47     },
48
49     submitclick(){
50         console.log('submitclick');
51     },
52
53     keyup() {
54         console.log('keyUp');
55     },
56
57     btn2click() {
58         console.log('btn2click')
59     }
60 }
61 })
62 </script>
63 </body>
64 </html>

```

执行结果



6.2 条件判断

6.2.1 v-if 指令

作用: 根据表达值的真假,切换元素的显示和隐藏(操纵dom元素)

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>v-if的使用</title>
6  </head>
7  <body>
8      <!--2.定义一个div元素-->
9      <div id="app">

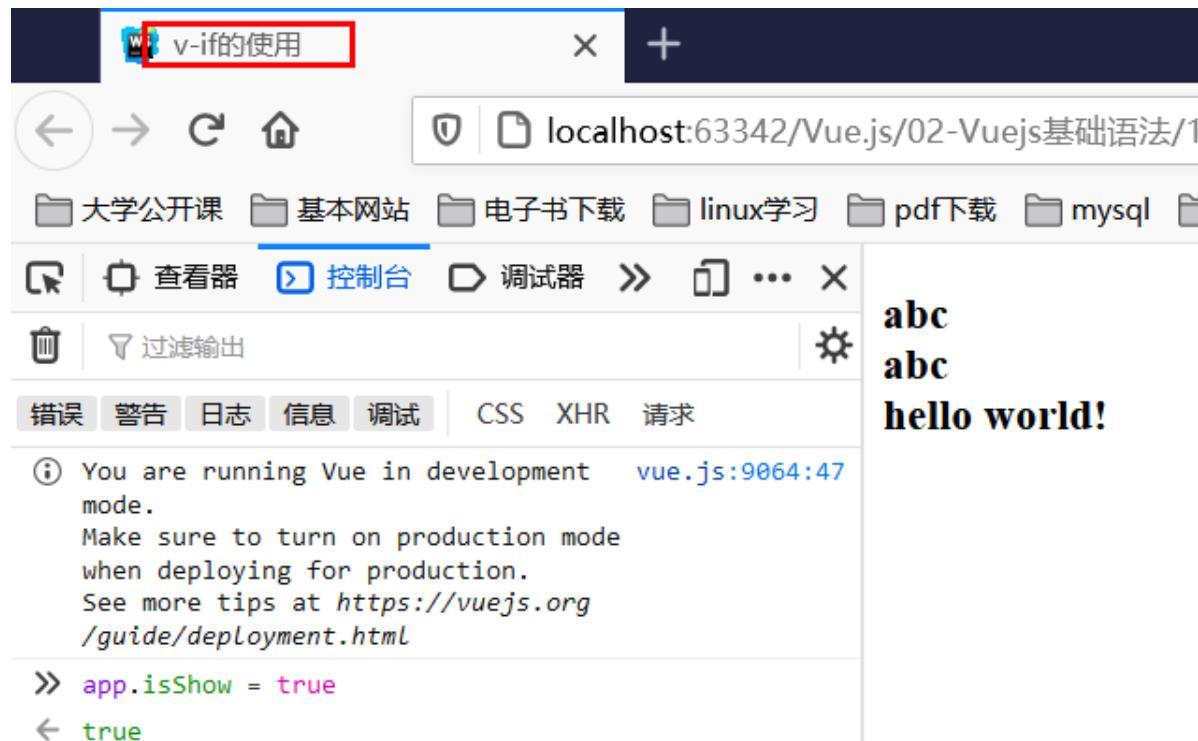
```

```

10      <!--条件判断-->
11      <h3 v-if="isShow">
12          <div>abc</div>
13          <div>abc</div>
14          {{message}}
15      </h3>
16  </div>
17
18  <script src="../../js/vue.js"></script>
19  <script>
20      // 创建对象
21      const app = new Vue({
22          // 挂载要管理的元素
23          el: '#app',
24
25          // 定义数据
26          data: {
27              message: 'hello world!',
28              isshow: true
29          }
30      })
31  </script>
32 </body>
33 </html>

```

执行结果



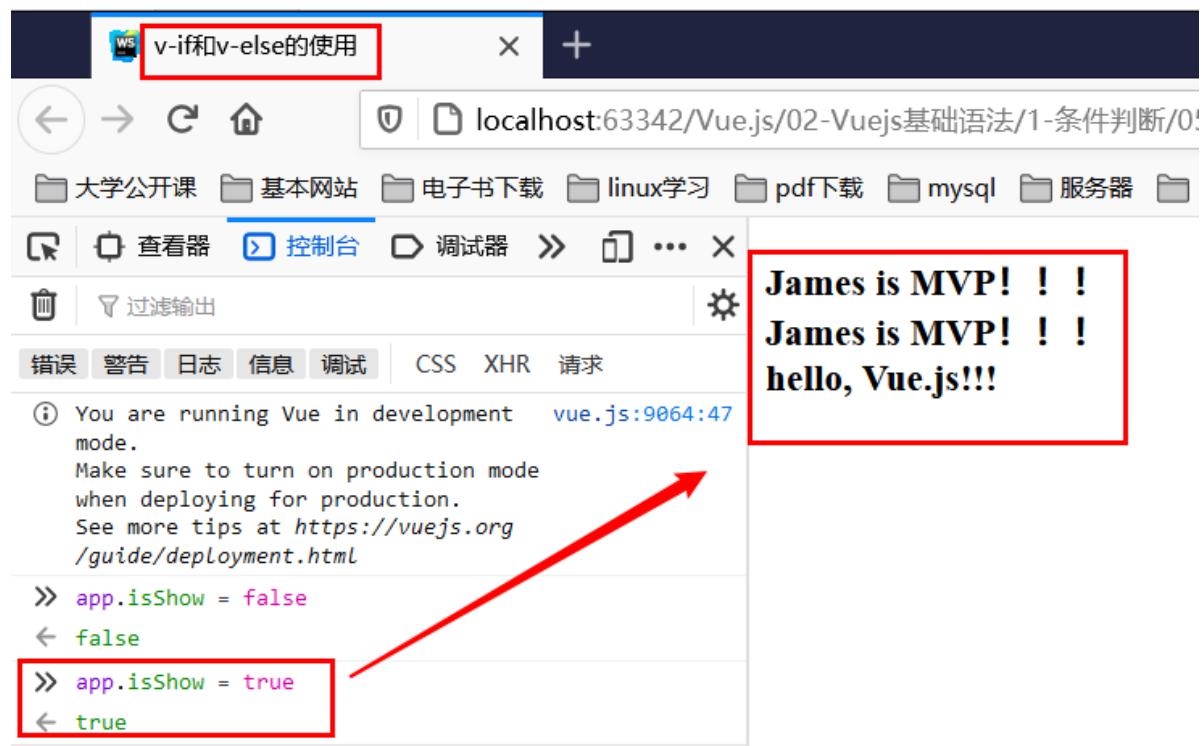
6.2.2 v-if和v-else指令

基本特点

- 可以根据表达式的值在DOM中渲染或销毁元素或组件。
- v-if后面的条件为false时，对应的元素以及其子元素不会渲染。

```
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>v-if和v-else的使用</title>
6   </head>
7   <body>
8     <!--2.定义一个div元素-->
9     <div id="app">
10       <h3 v-if="isShow">
11         <div>James is MVP! !!! </div>
12         <div>James is MVP! !!! </div>
13         {{message}}
14       </h3>
15
16       <h3 v-else>isShow为false时候,显示guardwhy!!!</h3>
17
18     </div>
19
20     <script src="../js/vue.js"></script>
21     <script>
22       // 创建对象
23       const app = new Vue({
24         // 挂载要管理的元素
25         el: '#app',
26         // 定义数据
27         data: {
28           message: 'hello, Vue.js!!!',
29           isShow: false
30         }
31       })
32     </script>
33   </body>
34 </html>
```

执行结果



复杂条件判断

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>复杂条件判断</title>
6  </head>
7  <body>
8      <!--2.定义一个div元素-->
9      <div id="app">
10         <h3>{{result}}</h3>
11     </div>
12
13     <script src="../js/vue.js"></script>
14     <script>
15         // 创建对象
16         const app = new Vue({
17             // 挂载要管理的元素
18             el: '#app',
19
20             // 定义数据
21             data: {
22                 score: 50
23             },
24
25             // 计算属性
26             computed: {
27                 result(){
28                     // 定义成绩变量
29                     let showMessage = '';
30                     // 条件判断
31                     if(this.score >= 90){
32                         showMessage = '优秀'
33                     }else if(this.score >= 80){
34                         showMessage = '良好'
35                     }else if(this.score >= 70){
36                         showMessage = '一般'
37                     }else if(this.score >= 60){
38                         showMessage = '及格'
39                     }else{
40                         showMessage = '不及格'
41                     }
42                     return showMessage
43                 }
44             }
45         })
46     </script>
47  </body>
48  </html>
```

执行结果

```
<!DOCTYPE html>
<html lang="en">
  <head> ...
  </head>
  <body>
    <!--2.定义一个div元素-->
    <div id="app">
      <h3>不及格</h3>
    </div>
    <script src="../js/vue.js"></script>
    <script>...
    </script>
  </body>
</html>
```

6.2.3 用户登录案例

存在问题

在有输入内容的情况下，切换了类型，会发现文字依然显示之前的内容。

解决方案

这是因为Vue在进行DOM渲染时，出于性能考虑，会尽可能的复用已经存在的元素，而不是重新创建新的元素。

Vue内部会发现原来的input元素不再使用，直接作为else中的input来使用了。如果不希望Vue出现类似重复利用的问题，**可以给对应的input添加key** 并且我们需要保证key的不同。

代码示例

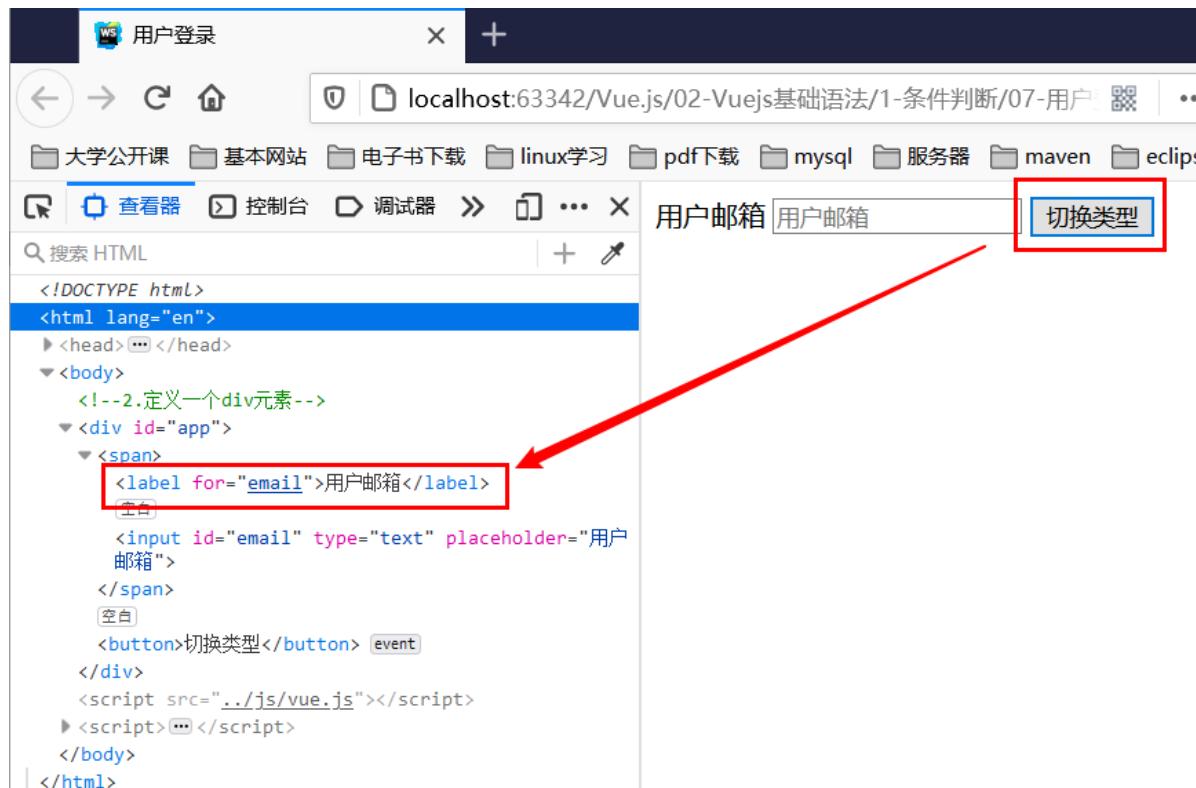
```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>用户登录</title>
6  </head>
7  <body>
8  <!--2.定义一个div元素-->
9    <div id="app">
10      <!--条件判断-->
11      <span v-if="isUser">
12        <label for="username">用户账户</label>
13        <input type="text" id="username" placeholder="用户账户"
key="username">
14        </span>
15      <span v-else>
16        <label for="email">用户邮箱</label>
17        <input type="text" id="email" placeholder="用户邮箱" key="email">
18      </span>
19
20      <!--切换按钮-->
```

```

21     <button @click="isUser=!isUser">切换类型</button>
22   </div>
23
24   <script src="../../js/vue.js"></script>
25   <script>
26     // 创建对象
27     const app = new Vue({
28       // 挂载要管理的元素
29       el: '#app',
30
31       // 定义数据
32       data: {
33         isUser: true
34       }
35     })
36   </script>
37 </body>
38 </html>

```

执行结果



6.2.4 v-show指令

基本特点

v-show的用法和v-if非常相似，也用于决定一个元素是否渲染。

v-if当条件为false时，压根不会有对应的元素在DOM中，v-show当条件为false时，仅仅是将元素的display属性设置为none而已。

```

1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8">

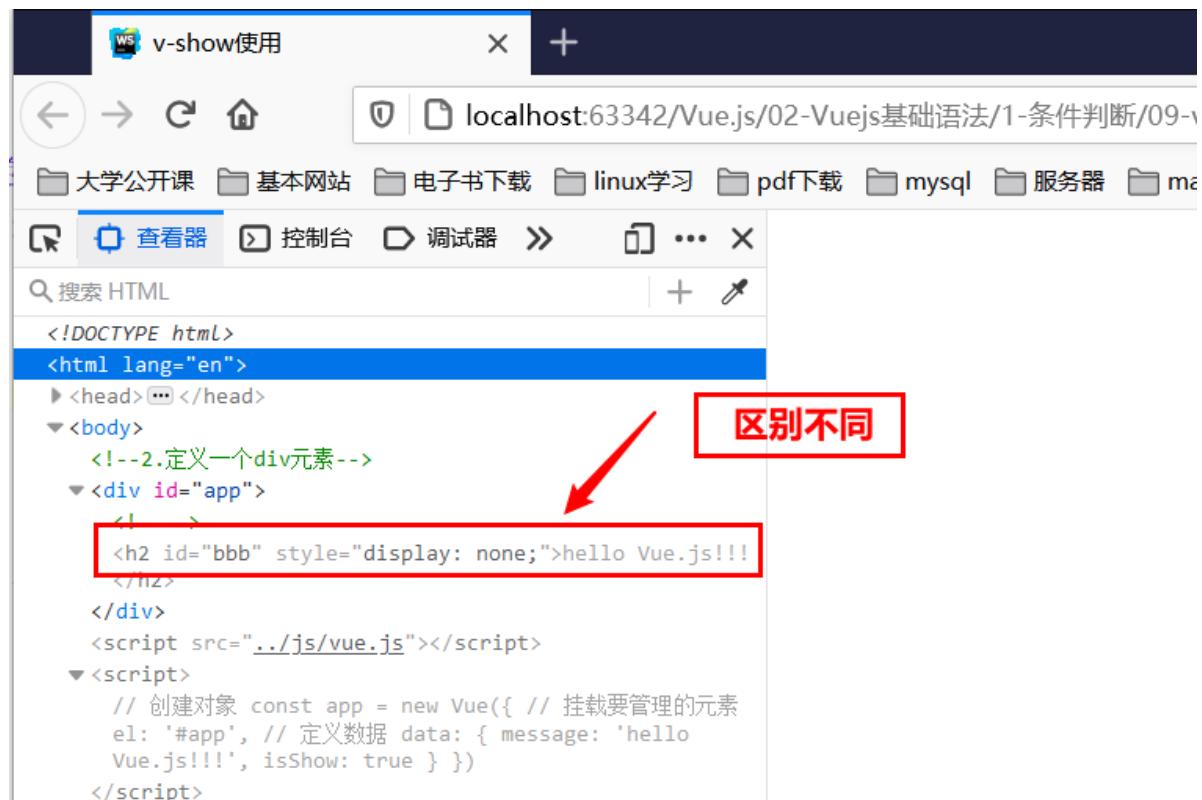
```

```

5   <title>v-show使用</title>
6   </head>
7   <body>
8   <!--2.定义一个div元素-->
9     <div id="app">
10    <!--v-if: 当条件为false时，包含v-if指令的元素，根本就不会存在dom中-->
11    <h2 v-if="isShow" id="aaa">{{message}}</h2>
12
13    <!--v-show: 当条件为false时，v-show只是给我们的元素添加一个行内样式：display:none-->
14    <h2 v-show="isShow" id="bbb">{{message}}</h2>
15  </div>
16
17  <script src="../js/vue.js"></script>
18  <script>
19    // 创建对象
20    const app = new Vue({
21      // 挂载要管理的元素
22      el: '#app',
23
24      // 定义数据
25      data: {
26        message: 'hello Vue.js!!!',
27        isShow: true
28      }
29    })
30  </script>
31 </body>
32 </html>

```

执行结果



6.3 循环遍历

6.3.1 v-for 遍历数组

基本特点

v-for的语法类似于JavaScript中的for循环，格式如下：item in items的形式。

遍历的过程中不需要使用索引值

1 | `v-for="movie in movies"`

2 | 依次从movies中取出movie，并且在元素的内容中，可以使用Mustache语法，来使用movie

遍历的过程中，需要拿到元素在数组中的索引值

1 | 语法格式：`v-for=(item, index) in items`

2 | 其中的index就代表了取出的item在原数组的索引值

代码示例

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>v-for遍历数组</title>
6 </head>
7 <body>
8 <!--2. 定义一个div元素-->
9   <div id="app">
10    <!--在遍历的过程中，没有使用索引值(下标值)-->
11    <ul>
12      <li v-for="item in names" :key="item">{{item}}</li>
13    </ul>
14
15    <!--在遍历的过程中，获取索引值-->
16    <ul>
17      <li v-for="(item, index) in names">{{index+1}}.{{item}}</li>
18    </ul>
19  </div>
20
21  <script src="../js/vue.js"></script>
22  <script>
23    // 创建对象
24    const app = new Vue({
25      // 挂载要管理的元素
26      el: '#app',
27
28      // 定义数据
29      data: {
30        names: ['guardwhy', 'kobe', 'james', 'curry']
31      }
32    })
33  </script>
34 </body>
35 </html>
```

执行结果

The screenshot shows a browser window with the title "v-for遍历数组". The address bar displays "localhost:63342/Vue.js/02-Vuejs基础语法/2-循环遍历/01". The developer tools sidebar includes tabs for "查看器" (Inspector), "控制台" (Console), "调试器" (Debugger), and "..." (More). A search bar for "搜索 HTML" is present. The left panel shows the DOM tree:

```
<!DOCTYPE html>
<html lang="en">
  <head> ...
  </head>
  <body>
    <!--2.定义一个div元素-->
    <div id="app"> ...
    <script src="../js/vue.js"></script>
    <script> ...
  </body>
</html>
```

The right panel displays a list of names:

- guardWhy
- kobe
- james
- curry

Below this, another list is shown:

- 1.guardWhy
- 2.kobe
- 3.james
- 4.curry

6.3.2 v-for遍历对象

代码示例

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>v-for遍历对象</title>
6  </head>
7  <body>
8    <!--2.定义一个div元素-->
9    <div id="app">
10      <!--在遍历对象的过程中，如果只是获取一个值，那么获取到的是value-->
11      <ul>
12        <li v-for="item in info">{{item}}</li>
13      </ul>
14
15      <!--获取key和value 格式：(value, key) -->
16      <ul>
17        <li v-for="(value, key) in info">{{value}}-{{key}}</li>
18      </ul>
19
20      <!--获取key和value和index 格式：(value, key, index) -->
21      <ul>
22        <li v-for="(value, key, index) in info">{{value}}-{{key}}-{{index}}</li>
23      </ul>
24    </div>
25
26    <script src="../js/vue.js"></script>
27    <script>
28      // 创建对象
29      const app = new Vue({
30        // 挂载要管理的元素
31        el: '#app',
```

```

32         // 定义数据
33         data: {
34             // 创建对象
35             info: {
36                 name: 'guardwhy',
37                 age: 27,
38                 height: 1.71
39             }
40         }
41     }
42 }
43 </script>
44 </body>
45 </html>

```

执行结果

The screenshot shows a browser window with the title "v-for遍历对象". The address bar indicates the URL is "localhost:63342/Vue.js/02-Vuejs基础语法/2-循环遍历/02-v-". The developer tools sidebar has "查看器" (Inspector) selected. The left panel displays the HTML code:

```

<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <!--2.定义一个div元素-->
    <div id="app">
      <ul>...</ul>
      <ul>...</ul>
      <ul>
        <li>...</li>
        <li>...</li>
        <li>...</li>
      </ul>
    </div>
    <script src="../js/vue.js"></script>
  </body>
</html>

```

The right panel shows the Vue.js component state for the "app" div, which contains three items:

- guardwhy
- 27
- 1.71

Each item is further expanded to show its properties:

- guardwhy-name
- 27-age
- 1.71-height

The third item's properties are also expanded:

- guardwhy-name-0
- 27-age-1
- 1.71-height-2

6.3.3 组件的key属性

官方推荐我们在使用v-for时，给对应的元素或组件添加上一个:key属性，key的作用主要是为了高效的更新虚拟DOM。

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>v-for遍历对象</title>
6  </head>
7  <body>
8  <!--2.定义一个div元素-->
9      <div id="app">
10         <!--ey的作用主要是为了高效的更新虚拟DOM-->

```

```
11     <ul>
12         <li v-for="item in letters" :key="item">{{item}}</li>
13     </ul>
14 </div>
15
16 <script src="../js/vue.js"></script>
17 <script>
18     // 创建对象
19     const app = new Vue({
20         // 挂载要管理的元素
21         el: '#app',
22
23         // 定义数据
24         data: {
25             letters: ['A', 'B', 'C', 'D', 'E']
26         }
27     })
28 </script>
29 </body>
30 </html>
```

执行结果

The screenshot shows a browser window with the title "v-for遍历对象". The address bar indicates the URL is "localhost:63342/Vue.js/02-Vuejs基础语法/2-循环遍历". The page content is a simple HTML structure generated by Vue.js:

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <!--2.定义一个div元素-->
    <div id="app">
      <ul>
        <li>::marker<br/>A</li>
        <li>...</li>
        <li>...</li>
        <li>...</li>
        <li>...</li>
      </ul>
    </div>
    <script src="../js/vue.js"></script>
    <script>...</script>
  </body>
</html>
```

A red box highlights the first list item in the rendered HTML, specifically the text "A". A red arrow points from this highlighted text to a red box containing the list items "A", "B", "C", "D", and "E" from the right sidebar of the developer tools. This visual comparison demonstrates that the "A" in the DOM is correctly mapped to the "A" in the data object.

6.3.4 数组方法响应式

Vue是响应式的，所以当数据发生变化时，Vue会自动检测数据变化，视图会发生对应的更新。

代码示例

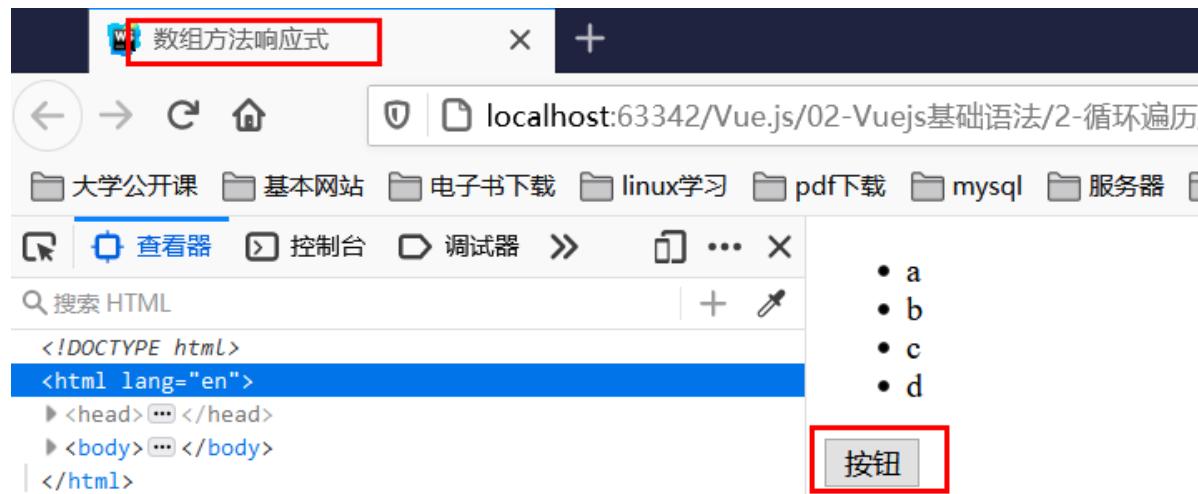
```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>数组方法响应式</title>
6  </head>
7  <body>
8      <!--2.定义一个div元素-->
9      <div id="app">
10         <!--遍历-->
11         <ul>
12             <li v-for="item in letters">{{item}}</li>
13         </ul>
14
15         <button @click="btnClick">按钮</button>
16     </div>
17
18     <script src="../js/vue.js"></script>
19     <script>
20         // 创建对象
21         const app = new Vue({
22             // 挂载要管理的元素
23             el: '#app',
24
25             // 定义数据
26             data: {
27                 letters: ['a', 'b', 'c', 'd']
28             },
29
30             // 方法
31             methods: {
32                 btnClick(){
33                     // 1.push方法
34                     // this.letters.push('aaa')
35                     // this.letters.push("kobe","james","rondo");
36
37                     // 2.pop(): 删除数组中的最后一个元素
38                     // this.letters.pop();
39
40                     // 3.shift(): 删除数组中的第一个元素
41                     // this.letters.shift();
42
43                     // 4.unshift(): 在数组最前面添加元素
44                     // this.letters.unshift('aaa', 'bbb', 'ccc')
45
46                     /*
47                     5.splice作用: 删除元素/插入元素/替换元素
48                         删除元素: 第二个参数传入你要删除几个元素(如果没有传,就删除后面所有的元
素)
49                         替换元素: 第二个参数, 表示我们要替换几个元素, 后面是用于替换前面的元素
50                         插入元素: 第二个参数, 传入0, 并且后面跟上要插入的元素
51                     */
52     
```

```

52
53     // this.letters.splice(1, 3, 'm', 'n', 'l', 'x')
54     this.letters.splice(1, 0, 'x', 'y', 'z')
55
56     // 6.sort()
57     // this.letters.sort()
58
59     // 7.reverse()
60     // this.letters.reverse()
61
62     // 注意：通过索引值修改数组中的元素
63     // this.letters[0] = 'bbbbbbb';
64     // this.letters.splice(0, 1, 'bbbbbbb')
65
66     // set(要修改的对象, 索引值, 修改后的值)
67     // vue.set(this.letters, 0, 'bbbbbbb')
68
69     // 9.通过索引值修改数组中的元素
70     // this.letters[0] = "guardwhy";
71 }
72 }
73 })
74 </script>
75 </body>
76 </html>

```

执行结果



6.3.5 点击换背景

代码示例

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>点击换背景</title>
6      <style>
7          .active{
8              color: red;
9          }
10     </style>

```

```
11 </head>
12 <body>
13 <!--2.定义一个div元素-->
14   <div id="app">
15     <!--遍历数组-->
16     <ul>
17       <li v-for="(item, index) in starts"
18         :class="{active: currentIndex == index}"
19         @click="liclick(index)">
20           {{index}}.{{item}}
21         </li>
22       </ul>
23     </div>
24
25   <script src="../../js/vue.js"></script>
26 <script>
27   // 创建对象
28   const app = new Vue({
29     // 挂载要管理的元素
30     el: '#app',
31
32     // 定义数据
33     data: {
34       starts: ['kobe', 'Jmaes', 'curry', 'Rando'],
35       // 设置点击索引值为0
36       currentIndex: 0
37     },
38
39     // 方法
40     methods: {
41       liclick(index){
42         this.currentIndex = index
43       }
44     }
45   })
46 </script>
47 </body>
48 </html>
```

执行结果

The screenshot shows a browser window with developer tools open. The address bar says 'localhost:63342/Vue.js/02-Vuejs基础语法/2-循环遍历/05-'. The developer tools sidebar has tabs for '查看器' (Inspector), '控制台' (Console), and '调试器' (Debugger). The '查看器' tab is selected. It shows an HTML tree with a red box around the 'active' class on the third list item. A red arrow points from this item to a list of names on the right: '0.kobe', '1.Jmaes', '2.curry' (with a red box around it), and '3.Rando'. The list items are represented by dots.

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <!--2.定义一个div元素-->
    <div id="app">
      <ul>
        <li class="">...</li> event
        <li class="">...</li> event
        <li class="active">...</li> event
        <li class="">...</li> event
      </ul>
    </div>
    <script src="../js/vue.js"></script>
    <script>...</script>
  </body>
</html>
```

6.4 高阶函数

基本特点

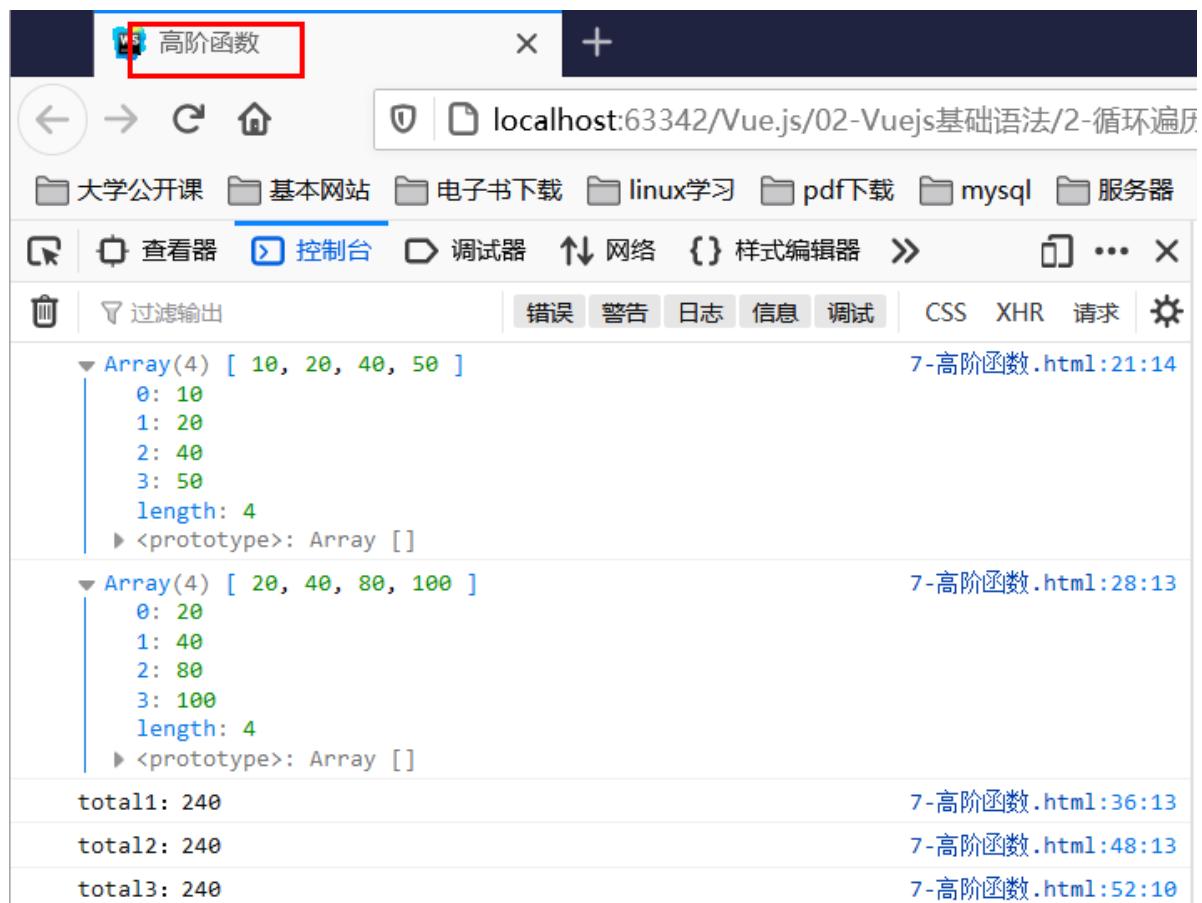
高阶函数是至少满足下面一个条件的函数：

- 1、接收一个或多个函数作为参数。
- 2、返回一个函数。

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>高阶函数</title>
6  </head>
7  <body>
8    <script type="text/javascript">
9      // 创建数组nums
10     const nums = [10, 20, 111, 222, 444, 40, 50];
11
12    /*
13     1.filter函数的使用
14     filter中的回调函数有一个要求：必须返回一个boolean值。
15     true: 当返回true时，函数内部会自动将这次回调的n加入到新的数组中
16     false: 当返回false时，函数内部会过滤掉这次的n。
17   */
18   let newNums = nums.filter(function (n) {
19     return n < 100
20   });
21   console.log(newNums);
22
23   // 2. map函数的使用
24   let new2Nums = newNums.map(function (n) {
```

```
25     return n * 2;
26 );
27 // 输出结果
28 console.log(new2Nums);
29
30 // 3. reduce函数的使用:reduce作用对数组中所有的内容进行汇总。
31 let total1 = new2Nums.reduce(function (preValue, n) {
32     return preValue + n;
33 }, 0)
34
35 // 输出结果值
36 console.log("total1: " + total1);
37
38 // 函数式编程
39 let total2 = nums.filter(function (n) {
40     return n < 100;
41 }).map(function (n) {
42     return n * 2;
43 }).reduce(function (preValue, n) {
44     return preValue + n;
45 }, 0);
46
47 // 输出total2
48 console.log("total2: " + total2);
49
50 // 5.ES6方式实现
51 let total3 = nums.filter(n=>n <100).map(n => n * 2).reduce((pre, n) =>
52     pre + n);
53     console.log("total3: " + total3);
54 </script>
55 </body>
56 </html>
```

代码示例



6- 购物车案例

6.1 项目目录



6.2 index.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>购物车案例</title>
6    <link rel="stylesheet" href="style.css">
7  </head>
8  <body>
9    <div id="app">
10      <div v-if="books.length">
11        <table>
12          <thead>
13            <tr>
14              <th></th>
15              <th>书籍名称</th>
16              <th>出版日期</th>
17              <th>价格</th>
```

```

18         <th>购买数量</th>
19         <th>操作</th>
20     </tr>
21 </thead>
22
23     <tbody>
24     <tr v-for="(item, index) in books">
25         <td>{{item.id}}</td>
26         <td>{{item.name}}</td>
27         <td>{{item.date}}</td>
28         <td>{{item.price | showPrice}}</td>
29         <td>
30             <button @click="decrement(index)" :disabled="item.count <= 0">-
</button>
31             {{item.count}}
32             <button @click="increment(index)">+</button>
33         </td>
34         <td>
35             <button @click="removeHandle(index)">移除</button>
36         </td>
37     </tr>
38 </tbody>
39 </table>
40 <h3>总价格:{{totalPrice | showPrice}}</h3>
41 </div>
42 <h3 v-else>购物车为空</h3>
43 </div>
44 </body>
45 <!--引入vue-->
46 <script src="../js/vue.js"></script>
47 <!--引入main.js文件-->
48 <script src="main.js"></script>
49 </html>

```

6.3 style.css

```

1 table {
2     border: 1px solid #e9e9e9;
3     border-collapse: collapse;
4     border-spacing: 0;
5 }
6
7 th, td {
8     padding: 8px 16px;
9     border: 1px solid #e9e9e9;
10    text-align: left;
11 }
12
13 th {
14     background-color: #f7f7f7;
15     color: #5c6b77;
16     font-weight: 600;
17 }

```

6.4 main.js

```
1 const app = new Vue({
2   el: '#app',
3   data: {
4     books: [
5       {
6         id: 1,
7         name: '《算法导论》',
8         date: '2006-9',
9         price: 85.00,
10        count: 1
11      },
12      {
13        id: 2,
14        name: '《UNIX编程艺术》',
15        date: '2006-2',
16        price: 59.00,
17        count: 1
18      },
19      {
20        id: 3,
21        name: '《编程珠玑》',
22        date: '2008-10',
23        price: 39.00,
24        count: 1
25      },
26      {
27        id: 4,
28        name: '《代码大全》',
29        date: '2006-3',
30        price: 128.00,
31        count: 1
32      },
33    ]
34  },
35
36  // 方法
37  methods: {
38    // 加操作
39    increment(index) {
40      this.books[index].count++
41    },
42
43    // 减操作
44    decrement(index) {
45      this.books[index].count--
46    },
47
48    // 移除操作
49    removeHandle(index){
50      this.books.splice(index,1)
51    }
52  },
53
54  // 计算属性
55  computed: {
```

```
56 // 定义总价格函数
57 totalPrice(){
58     // 设置总价格为空
59     let totalPrice = 0;
60     // 方式 1遍历循环
61     /*
62     for(let i=0; i<this.books.length; i++){
63         totalPrice += this.books[i].price * this.books[i].count;
64     }
65     */
66
67     // 方式2 遍历循环
68     /*
69     for(let i in this.books){
70         totalPrice += this.books[i].price * this.books[i].count;
71     }
72     */
73     // 方式3 遍历循环
74     /*
75     for (let item of this.books){
76         totalPrice += item.price * item.count;
77     }*/
78
79     // 返回总价格
80     // return totalPrice;
81
82     // 方式四 通过高阶函数实现
83     return this.books.reduce(function (preValue,book){
84         return preValue + book.price * book.count
85     },0)
86
87
88 }
89 },
90
91 // 过滤器
92 filters: {
93     showPrice(price){
94         return '$' + price.toFixed(2);
95     }
96 }
97 })
```

执行结果

	书籍名称	出版日期	价格	购买数量	操作
1	《算法导论》	2006-9	¥85.00	<button>-</button> 5 <button>+</button>	<button>移除</button>
2	《UNIX编程艺术》	2006-2	¥59.00	<button>-</button> 1 <button>+</button>	<button>移除</button>
3	《编程珠玑》	2008-10	¥39.00	<button>-</button> 0 <button>+</button>	<button>移除</button>
4	《代码大全》	2006-3	¥128.00	<button>-</button> 2 <button>+</button>	<button>移除</button>

总价格:¥740.00

7- v-model指令

7.1 v-model基本使用

Vue中使用v-model指令来实现表单元素和数据的双向绑定。

案例的解析

- 当输入框输入内容时，因为input中的v-model绑定了message。所以会实时将输入的内容传递给message，message发生改变。
- 当message发生改变时，因为上面我们使用Mustache语法，将message的值插入到DOM中，所以DOM会发生响应的改变。
- 所以，通过v-model实现了双向的绑定。

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>v-model的基本使用</title>
6  </head>
7  <body>
8  <!--2.定义一个div元素-->
9      <div id="app">
10         <input type="text" v-model="message">
11         <h2>{{message}}</h2>
12     </div>
13
14     <script src="../js/vue.js"></script>
15     <script>
16         // 创建对象
17         const app = new Vue({
18             // 挂载要管理的元素
19             el: '#app',
20
21             // 定义数据
22             data: {
23                 message: 'hello vue.js!!!'
24             }
25         })
26     </script>
27 </body>
28 </html>

```

执行结果

hello Vue.js!!!

hello Vue.js!!!

7.2 v-mode的原理

```
<div id="app">
  <input type="text" v-model="message">
  <h2>{{message}}</h2>
</div>

<script src=".//js/vue.js"></script>
<script>
  // 创建对象
  const app = new Vue({
    // 挂载要管理的元素
    el: '#app',
    // 定义数据
    data: {
      message: 'hello Vue.js!!!'
    }
  })
</script>
```

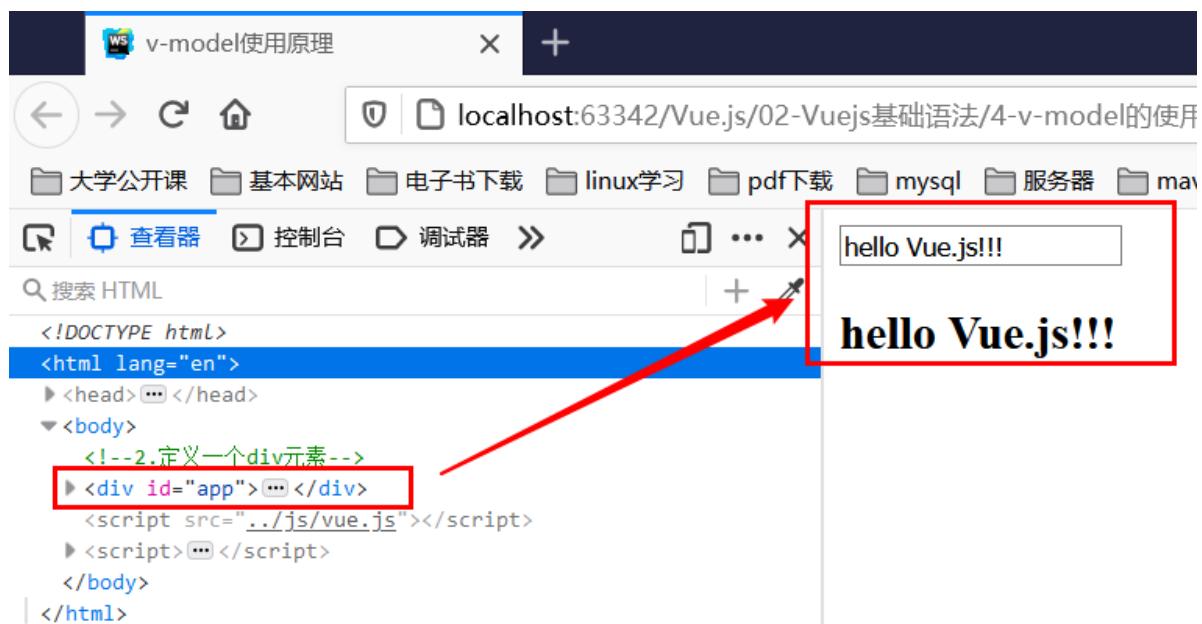
v-model其实是一个语法糖，它的背后本质上是包含两个操作

- v-bind绑定一个value属性。
- v-on指令给当前元素绑定input事件。

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>v-model使用原理</title>
```

```
6  </head>
7  <body>
8  <!--2.定义一个div元素-->
9   <div id="app">
10    <!--输入和监听-->
11    <input type="text" :value="message" @input="message = $event.target.value">
12
13    <h2>{{message}}</h2>
14  </div>
15
16  <script src="../js/vue.js"></script>
17  <script>
18    // 创建对象
19    const app = new Vue({
20      // 挂载要管理的元素
21      el: '#app',
22
23      // 定义数据
24      data: {
25        message: 'hello world!'
26      },
27      // 方法
28      methods: {
29        valueChange(event){
30          this.message = event.target.value;
31        }
32      }
33    })
34  </script>
35 </body>
36 </html>
```

执行结果



7.3 v-model结合radio类型

当存在多个单选框时。

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>v-model结合radio</title>
6 </head>
7 <body>
8 <!--2.定义一个div元素-->
9   <div id="app">
10    <label for="male">
11      <input type="radio" id="male" value="男" v-model="sex">男
12    </label>
13    <label for="female">
14      <input type="radio" id="female" value="女" v-model="sex">女
15    </label>
16
17    <h3>您选择的性别是:{{sex}}</h3>
18  </div>
19
20  <script src="../../js/vue.js"></script>
21  <script>
22    // 创建对象
23    const app = new Vue({
24      // 挂载要管理的元素
25      el: '#app',
26
27      // 定义数据
28      data: {
29        sex: '男'
30      }
31    })
32  </script>
33 </body>
34 </html>
```

执行结果

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>v-model结合radio</title>
</head>
<body>
    <!--2.定义一个div元素-->
    <div id="app">
        <label for="male">...</label>
        <label for="female">...</label>
        <h3>您选择的性别是:女</h3>
    </div>
    <script src="../js/vue.js"></script>
    <script>...</script>
</body>
</html>

```

7.4 v-mode结合单选框

单个勾选框

v-model即为布尔值，此时input的value并不影响v-model的值。

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>v-mode结合单选框</title>
6  </head>
7  <body>
8      <!--2.定义一个div元素-->
9      <div id="app">
10         <label for="agree">
11             <input type="checkbox" id="agree" v-model="isAgree">同意协议
12         </label>
13         <h2>你的选择是:{{isAgree}}</h2>
14
15         <!--取反操作-->
16         <button :disabled="!isAgree">下一页</button>
17     </div>
18
19     <script src="../js/vue.js"></script>
20     <script>
21         // 创建对象
22         const app = new Vue({
23             // 挂载要管理的元素
24             el: '#app',
25
26             // 定义数据
27             data: {
28                 message: 'hello world!',
29                 isAgree: false
30             }
31         })
32     </script>

```

```
33 </body>
34 </html>
```

执行结果

```
<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <!--2.定义一个div元素-->
    <div id="app">
      <label for="agree"></label>
      <h2>你的选择是:<span>true</span></h2>
      <button>下一页</button>
    </div>
    <script src="../../js/vue.js"></script>
    <script></script>
  </body>
</html>
```

7.5 v-mode结合复选框

多个复选框

当是多个复选框时，因为可以选中多个，所以对应的data中属性是一个数组。

当选中某一个时，就会将input的value添加到数组中。

代码示例

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>v-mode结合复选框</title>
6  </head>
7  <body>
8  <!--2.定义一个div元素-->
9    <div id="app">
10      <input type="checkbox" value="篮球" v-model="hobbies">篮球
11      <input type="checkbox" value="足球" v-model="hobbies">足球
12      <input type="checkbox" value="乒乓球" v-model="hobbies">乒乓球
13      <input type="checkbox" value="羽毛球" v-model="hobbies">羽毛球
14      <h2>您的爱好是: {{hobbies}}</h2>
15
16    </div>
17
18    <script src="../../js/vue.js"></script>
19    <script>
20      // 创建对象
21      const app = new Vue({
22        // 挂载要管理的元素
23      })
24    </script>
25  </body>
26</html>
```

```

23     el: '#app',
24     // 定义数据
25     data: {
26       message: 'hello world!',
27       isAgree: false,
28       hobbies: []
29     }
30   })
31 </script>
32 </body>
33 </html>

```

执行结果

localhost:63342/Vue.js/02-Vuejs基础语法/4-v-model的使用/0

您的爱好是: ["篮球", "足球", "乒乓球"]

7.6 v-model结合select类型

基本特点

单选: 只能选中一个值。

- v-model绑定的是一个值。
- 当我们选中option中的一个时，会将它对应的value赋值到mySelect中

多选: 可以选中多个值。

- v-model绑定的是一个数组。
- 当选中多个值时，就会将选中的option对应的value添加到数组mySelects中。

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>v-model结合select类型</title>
6  </head>
7  <body>
8  <!--2.定义一个div元素-->
9  <div id="app">
10   <!--选择一个-->
11   <select name="abc" v-model="fruit">
12     <option value="苹果">苹果</option>
13     <option value="香蕉">香蕉</option>
14     <option value="榴莲">榴莲</option>

```

```

15         <option value="葡萄">葡萄</option>
16     </select>
17
18     <h3>您选择的水果是:{{fruit}}</h3>
19
20     <!--多选择-->
21     <select name="abc" v-model="fruits" multiple>
22         <option value="苹果">苹果</option>
23         <option value="香蕉">香蕉</option>
24         <option value="榴莲">榴莲</option>
25         <option value="葡萄">葡萄</option>
26     </select>
27     <h3>你选择的水果是:{{fruits}}</h3>
28 </div>
29
30 <script src="../js/vue.js"></script>
31 <script>
32     // 创建对象
33     const app = new Vue({
34         // 挂载要管理的元素
35         el: '#app',
36
37         // 定义数据
38         data: {
39             fruit: '香蕉',
40             fruits: []
41         }
42     })
43 </script>
44 </body>
45 </html>

```

执行结果

v-model结合select类型

localhost:63342/Vue.js/02-Vuejs基础语法/4-v-model的使用/0

您选择的水果是:榴莲

你选择的水果是:["苹果", "香蕉", "榴莲"]

```

<!DOCTYPE html>
<html lang="en">
<head>...</head>
<body>
    <!--2.定义一个div元素-->
    <div id="app">
        <select name="abc">...</select> event
        <h3>您选择的水果是:榴莲</h3>
        <select multiple="multiple" name="abc">...</select> event
        [滚动]
        <h3>你选择的水果是:[ "苹果", "香蕉", "榴莲" ]</h3>
    </div>
    <script src="../js/vue.js"></script>
    <script>...</script>
</body>
</html>

```

7.7 值绑定

值绑定就是动态的给value赋值而已。

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>值绑定</title>
6 </head>
7 <body>
8   <div id="app">
9     <label v-for="item in originHobbies" :for="item">
10       <input type="checkbox" :value="item" v-model="hobbies">{{item}}
11     </label>
12     <h3>您的爱好是: {{hobbies}}</h3>
13   </div>
14
15   <script src="../js/vue.js"></script>
16   <script>
17     // 创建对象
18     const app = new Vue({
19       // 挂载要管理的元素
20       el: '#app',
21
22       // 定义数据
23       data: {
24         hobbies: [],
25         originHobbies: ['篮球', '足球', '乒乓球', '羽毛球', '台球', '高尔夫
26         球']
27       }
28     })
29   </script>
30 </body>
31 </html>
```

执行结果

The screenshot shows a browser window with the title "值绑定". The address bar shows "localhost:63342/Vue.js/02-Vuejs基础语法/4-v-model的使用/01". The developer tools are open, specifically the "查看器" (Inspector) tab. The DOM tree on the left shows a `<div id="app">` element containing several `<label>` elements with `v-for` directives. The browser output on the right shows the rendered HTML with three checkboxes checked: "篮球", "足球", and "乒乓球". A red arrow points from the "乒乓球" checkbox in the DOM tree to its checked state in the browser output.

```
<!DOCTYPE html>
<html lang="en">
<head>
</head>
<body>
  <div id="app">
    <label for="篮球"></label>
    <label for="足球"></label>
    <label for="乒乓球"></label>
    <label for="羽毛球"></label>
    <label for="台球"></label>
    <label for="高尔夫球"></label>
    <h3>您的爱好是: [ "篮球", "足球", "乒乓球" ]</h3>
  </div>
<script src="../js/vue.js"></script>
</body>
</html>
```

7.8 v-model修饰符

lazy修饰符

- 默认情况下，v-model默认是在input事件中同步输入框的数据的。
- 也就是说，一旦有数据发生改变对应的data中的数据就会自动发生改变。
- lazy修饰符可以让数据在失去焦点或者回车时才会更新。

number修饰符

- 默认情况下，在输入框中无论我们输入的是字母还是数字，都会被当做字符串类型进行处理。
- 但是如果我们希望处理的是数字类型，那么最好直接将内容当做数字处理。
- number修饰符可以让在输入框中输入的内容自动转成数字类型。

trim修饰符

如果输入的内容首尾有很多空格，通常我们希望将其去除，trim修饰符可以过滤内容左右两边的空格。

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>v-model修饰符作用</title>
6  </head>
7  <body>
8      <div id="app">
9          <!--1.修饰符:lazy可以让数据在失去焦点或者回车时才会更新-->
10         <input type="text" v-model.lazy="message">
11         <h2>{{message}}</h2>
12
13         <!--2.修饰符:number可以让在输入框中输入的内容自动转成数字类型-->
14         <input type="number" v-model.number="age">
15         <h2>{{age}}-{{typeof age}}</h2>
16
17         <!--3.trim修饰符可以过滤内容左右两边的空格-->
18         <input type="text" v-model.trim="name">
19         <h2>你输入的名字:{{name}}</h2>
20     </div>
21
22     <script src="../js/vue.js"></script>
23     <script>
24         // 创建对象
25         const app = new Vue({
26             // 挂载要管理的元素
27             el: '#app',
28             // 定义数据
29             data: {
30                 message: 'hello vue.js!!!',
31                 age: 0,
32                 name: ''
33             }
34         })
35     </script>
36     </body>
37     </html>
```

执行结果

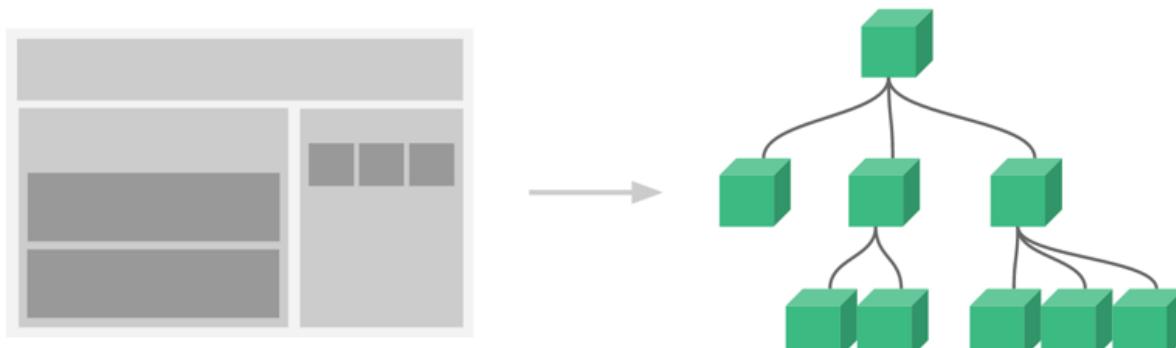
```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <div id="app">
      <input type="text"> event
      <h2>hello</h2>
      <input type="number"> event
      <h2>123-number</h2>
      <input type="text"> event
      <h2>你输入的名字:guardwhy</h2>
    </div>
    <script src="../../js/vue.js"></script>
  </body>
</html>
```

8- 组件化开发

8.1 基本概念

组件化是Vue.js中的重要思想。

它提供了一种抽象，让我们可以开发出一个个独立可复用的小组件来构造我们的应用。
任何的应用都会被抽象成一颗组件树。

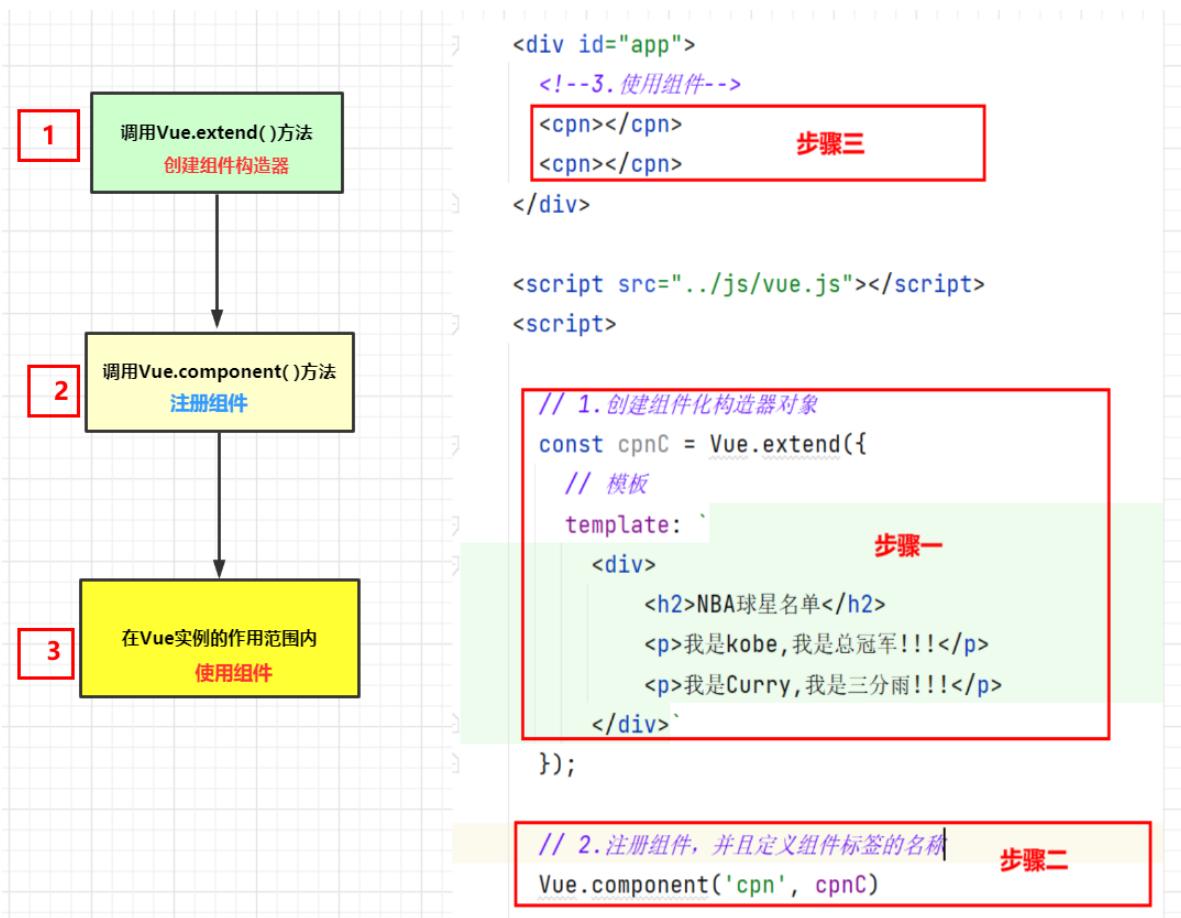


组件化思想的应用

- 有了组件化的思想，我们在之后的开发中就要充分的利用它。
- 尽可能的将页面拆分成一个个小的、可复用的组件。
- 这样让我们的代码更加方便组织和管理，并且扩展性也更强。

8.2 组件化的基本使用

组件的使用分成三个步骤



- 创建组件构造器。
- 注册组件
- 使用组件。

代码示例

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>组件化的基本使用</title>
6  </head>
7  <body>
8  <!--2. 定义一个div元素-->
9    <div id="app">
10      <!--3. 使用组件-->
11      <cpn></cpn>
12      <cpn></cpn>
13    </div>

15  <script src="../js/vue.js"></script>
16  <script>

18  // 1. 创建组件化构造器对象
19  const cpnC = Vue.extend({
20    // 模板
21    template: `
22      <div>
23        <h2>NBA球星名单</h2>
24        <p>我是kobe, 我是总冠军!!!</p>
25        <p>我是Curry, 我是三分雨!!!</p>

```

```

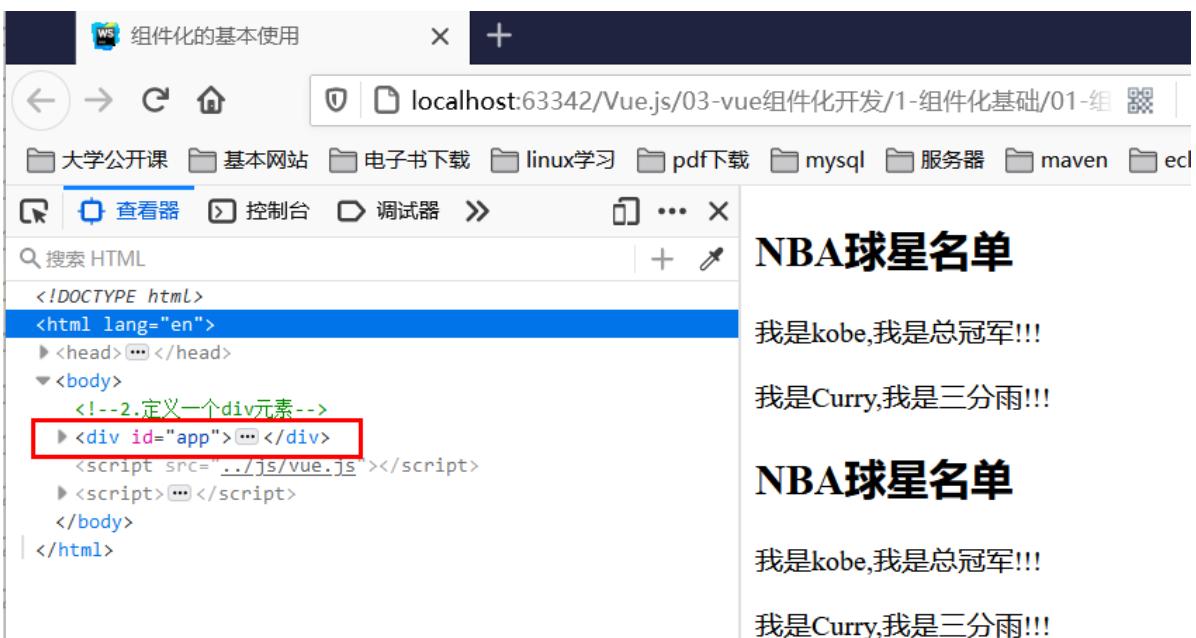
26         `
```

```

27     });
28
29     // 2.注册组件
30     Vue.component('cpn', cpnC)
31
32
33     // 创建对象
34     const app = new Vue({
35         // 挂载要管理的元素
36         el: '#app'
37     })
38     </script>
39 </body>
40 </html>

```

执行结果



8.3 注册组件步骤解析

Vue.extend()

- 调用Vue.extend()创建的是一个组件构造器。
- 通常在创建组件构造器时，传入template代表我们自定义组件的模板。
- 该模板就是在使用到组件的地方，要显示的HTML代码。

Vue.component()

- 调用Vue.component()是将刚才的组件构造器注册为一个组件，并且给它起一个组件的标签名称。
- 所以需要传递两个参数：1、注册组件的标签名 2、组件构造器

在Vue实例的作用范围内使用组件

组件必须挂载在某个Vue实例下，否则它不会生效。



8.4 全局组件和局部组件

调用Vue.component()注册组件时，组件的注册是全局的，这意味着该组件可以在任意Vue示例下使用。

如果注册的组件是挂载在某个实例中，那么就是一个局部组件。

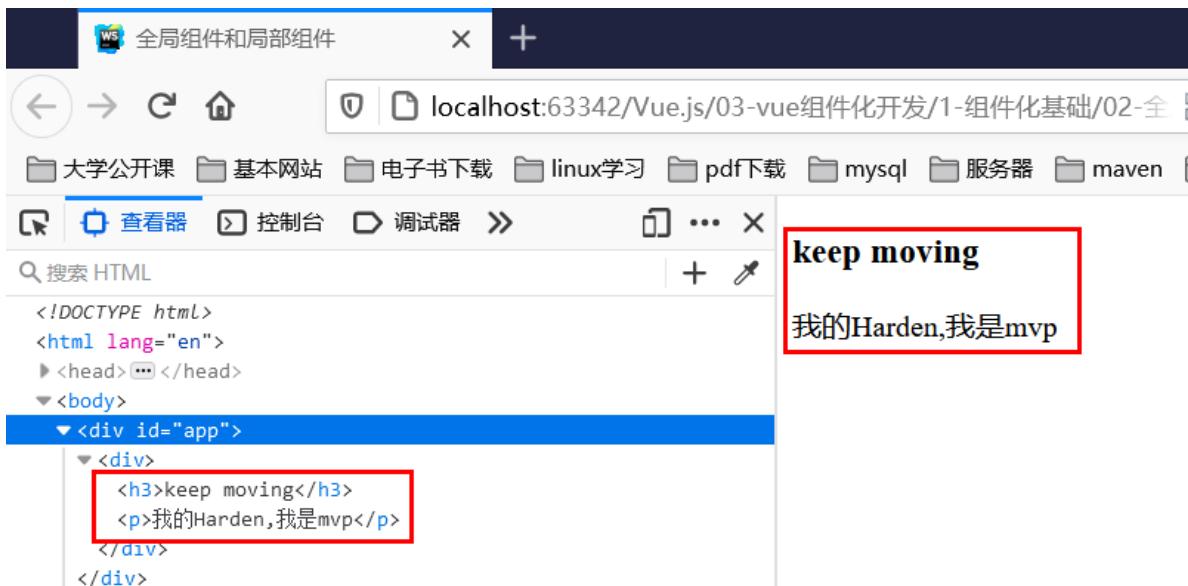
```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>全局组件和局部组件</title>
6  </head>
7  <body>
8      <div id="app">
9          <!--3. 使用组件-->
10         <cpn></cpn>
11     </div>
12     <script src="../js/vue.js"></script>
13     <script>
14         // 1. 创建组件构造器
15         const cpnC = Vue.extend({
16             template:
17                 <div>
18                     <h3>keep moving</h3>
19                     <p>我的Harden,我是mvp</p>
20                 </div>
21         });
22
23
24         // 注册组件(全局组件, 意味着可以在多个vue的实例下面使用)
25         // vue.component('cpn', cpnC);
26
27
28         // 创建对象
29         const app = new Vue({
30             // 挂载要管理的元素
31             el: '#app',
32             data: {
33                 message: 'hello vue.js!!!'
34             },
35             // 注册局部组件
36             components: {
37                 // 使用组件时候的标签名(cpn),组件构造器(cpnC)
38                 cpn: cpnC
39             }
40         })
41
42         // 在这里使用组件
43         <div>
44             <h2>NBA球星名单</h2>
45             <p>我是kobe,我是总冠军!!!</p>
46             <p>我是Curry,我是三分雨!!!</p>
47         </div>
48     </div>
49     <script src="..../js/vue.js"></script>
50 </script>
51 </body>
52 </html>

```

```
38     }
39   })
40 </script>
41 </body>
42 </html>
```

执行结果



8.5 父组件和子组件

组件和组件之间存在层级关系，而其中一种非常重要的关系就是父子组件的关系。

代码示例

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>父组件和子组件</title>
6  </head>
7  <body>
8  <!--2.定义一个div元素-->
9    <div id="app">
10      <cpn2></cpn2>
11    </div>
12
13    <script src="../../js/vue.js"></script>
14    <script>
15
16      // 1.创建第一个组件构造器(子组件)
17      const cpnC1 = Vue.extend({
18        template: `
19          <div>
20            <h2>我是NBA球星</h2>
21            <p>我是curry,我是三分王!!!</p>
22            </div>
23          `
24      });
25
```

```

26 // 2. 创建第二个组件构造器(父组件)
27 const cpnC2 = Vue.extend({
28   template: `
29     <div>
30       <h2>我是NBA球星</h2>
31       <p>我是kobe,我是MVP!!!</p>
32       <cpn1></cpn1>
33     </div>
34   `,
35   components: {
36     cpn1: cpnC1
37   }
38 });
39
40 // root组件
41 const app = new Vue({
42   // 挂载要管理的元素
43   el: '#app',
44   // 定义数据
45   data: {
46     message: 'hello world!'
47   },
48   components: {
49     cpn2: cpnC2
50   }
51 })
52 </script>
53 </body>
54 </html>

```

执行结果

The screenshot shows a browser window titled "父组件和子组件" (Parent Component and Child Component) displaying the rendered HTML output. The rendered content includes two main sections: one with "我是NBA球星" and "我是kobe,我是MVP!!!", and another with "我是NBA球星" and "我是curry,我是三分王!!!". The browser's developer tools are open, showing the DOM structure. Two specific parts of the DOM are highlighted with red boxes:

- A red box highlights the inner content of the first child component's template: <h2>我是NBA球星</h2> and <p>我是kobe,我是MVP!!!</p>.
- A red box highlights the inner content of the second child component's template: <h2>我是NBA球星</h2> and <p>我是curry,我是三分王!!!</p>.

The browser's address bar shows the URL: localhost:63342/Vue.js/03-vue组件化开发/1-组件化基础/03-父组件和子组件

8.6 注册组件语法糖

Vue为了简化这个过程，提供了注册的语法糖，主要是省去了调用Vue.extend()的步骤，而是可以直接使用一个对象来代替。

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>组件语法糖注册</title>
6  </head>
7  <body>
8      <!--2.定义一个div元素-->
9      <div id="app">
10         <cpn1></cpn1>
11         <cpn2></cpn2>
12     </div>
13
14     <script src="../js/vue.js"></script>
15     <script>
16
17         // 注册全局主键语法糖
18         // 1.创建组件构造器
19         // const cpn1 = Vue.extend()
20
21         // 2.注册组件(全局组件注册语法糖)
22         Vue.component('cpn1', {
23             template: `
24                 <div>
25                     <h2>NBA球星世界</h2>
26                     <p>我是kobe, 我是mvp!!!!</p>
27                 </div>
28             `
29         );
30
31         // 创建对象
32         const app = new Vue({
33             // 挂载要管理的元素
34             el: '#app',
35
36             // 定义数据
37             data: {
38                 message: 'hello world!'
39             },
40             // 注册局部组件(局部组件注册语法糖)
41             components: {
42                 'cpn2': {
43                     template: `
44                         <div>
45                             <h2>NBA球星世界</h2>
46                             <p>我是Curry, 我是三分王!!!!</p>
47                         </div>
48                     `
49                 }
50             }
51         })
52     </script>
```

```
53 </body>
54 </html>
```

执行结果

The screenshot shows a browser window with the title "组件语法糖注册". The address bar displays "localhost:63342/Vue.js/03-vue组件化开发/1-组件化基础/04-". The toolbar includes icons for back, forward, search, and developer tools. The developer tools panel is open, showing the "查看器" (Inspector) tab selected. The left sidebar lists file navigation items like "大学公开课", "基本网站", etc. The main content area displays the rendered HTML and its corresponding DOM structure.

HTML Content:

```
<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <!--2.定义一个div元素-->
    <div id="app">
      <div>
        <h2>NBA球星世界</h2>
        <p>我是kobe, 我是mvp!!!!</p>
      </div>
      <div>
        <h2>NBA球星世界</h2>
        <p>我是Curry, 我是三分王!!!!</p>
      </div>
    </div>
  </body>
</html>
```

DOM Structure (highlighted by red boxes):

- First Div:** Contains:
- **H2:** NBA球星世界
- **P:** 我是kobe, 我是mvp!!!!
- Second Div:** Contains:
- **H2:** NBA球星世界
- **P:** 我是Curry, 我是三分王!!!!

Right Panel (Rendered Content):

- Top Section:** NBA球星世界
我是kobe, 我是mvp!!!!
- Bottom Section:** NBA球星世界
我是Curry, 我是三分王!!!!

Bottom Bar: 双方对比

```

// 2. 注册组件(全局组件注册语法糖)
Vue.component('cpn1', {
  template: `
    <div>
      <h2>NBA球星世界</h2>
      <p>我是kobe, 我是mvp!!!!</p>
    </div>
  `;
});

// 创建对象
const app = new Vue({
  // 挂载要管理的元素
  el: '#app',
  // 注册局部组件(局部组件注册语法糖)
  components: {
    'cpn2': {
      template: `
        <div>
          <h2>NBA球星世界</h2>
          <p>我是Curry, 我是三分王!!!!</p>
        </div>
      `;
    }
  }
});

// 1. 创建组件构造器
const cpnC = Vue.extend({
  template: `
    <div>
      <h3>keep moving</h3>
      <p>我的Harden,我是mvp</p>
    </div>
  `);

  // 注册组件(全局组件, 意味着可以在多个Vue的实例下面使用)
  Vue.component('cpn', cpnC);
}

// 1. 创建组件构造器
const cpnC = Vue.extend({
  template: `
    <div>
      <h3>keep moving</h3>
      <p>我的Harden,我是mvp</p>
    </div>
  `);

  // 创建对象
  const app = new Vue({
    // 挂载要管理的元素
    el: '#app',
    data: {
      message: 'hello Vue.js!!!'
    },
    // 注册局部组件
    components: {
      // 使用组件时候的标签名(cpn), 组件构造器(cpnC)
      cpn: cpnC
    }
  });
}

```

8.7 组件模板分离

存在问题

- 通过语法糖简化了Vue组件的注册过程，另外还有一个地方的写法比较麻烦，就是template模块写法。
- 如果能将其中的HTML分离出来写，然后挂载到对应的组件上，必然结构会变得非常清晰。

Vue提供了两种方案来定义HTML模块内容

- 1 使用<script>标签
- 2 使用<template>标签

代码示例

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>组件模板分离</title>
6  </head>
7  <body>
8    <div id="app">
9      <!--使用组件-->
10     <cpn1></cpn1>
11     <cpn1></cpn1>
12   </div>

```

```
13    <!--方式1. script标签, 注意: 类型必须是text/x-template-->
14    <script type="text/x-template" id="cpn">
15        <div>
16            <h3>PHP</h3>
17            <p>是世界上最好的语言！！！</p>
18        </div>
19    </script>
20
21    <!--方式二: template标签-->
22    <!--<template id="cpn">
23        <div>
24            <h3>PHP</h3>
25            <p>是世界上最好的语言！！！</p>
26        </div>
27    </template>-->
28
29    <script src="../js/vue.js"></script>
30    <script>
31
32        // 1.注册全局组件
33        Vue.component('cpn1', {
34            template: '#cpn'
35        })
36        // 创建对象
37        const app = new Vue({
38            // 挂载要管理的元素
39            el: '#app',
40        })
41    </script>
42 </body>
43 </html>
```

执行结果

```
<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <div id="app">
      <div>
        <h3>PHP</h3>
        <p>是世界上最好的语言！！！</p>
      </div>
      <div>
        <h3>PHP</h3>
        <p>是世界上最好的语言！！！</p>
      </div>
    </div>
    <!--方式1. script标签, 注意: 类型必须是text/x-template-->
    <script id="cpn" type="text/x-template"></script>
    <!--方式二: template标签-->
    <!--
    <template id="cpn"> <div> <h3>PHP</h3> <p>是世界上最好的语
    言！！！</p> </div> </template>
    -->
    <script src="../js/vue.js"></script>
  </body>
</html>
```

8.8 组件中的数据存放

8.8.1 组件中的数据存放问题

组件可以访问Vue实例数据吗？

不能！！！

这个模块有属于自己的HTML模板，也应该有属性自己的数据data。

发现不能访问，而且即使可以访问，如果将所有的数据都放在Vue实例中，Vue实例就会变的非常臃肿。

The screenshot shows a browser developer tools console with the following details:

- Console Output:**
 - [Vue warn]: Property or method "message" is not defined on the instance but referenced during render. Make sure that this property is reactive, either in the data option, or for class-based components, by initializing the property. See: <https://vuejs.org/v2/guide/reactivity.html#Declaring-Reactive-Properties>.
 - You are running Vue in development mode. Make sure to turn on production mode when deploying for production. See more tips at <https://vuejs.org/guide/deployment.html>
- Code Preview:**

```

<div id="app">
  <cpn1></cpn1>
</div>

<!--template标签-->
<template id="cpn">
  <div>
    <h3>{{message}}</h3>
  </div>
</template>

<script src="../../js/vue.js"></script>
<script>
  // 创建对象
  const app = new Vue({
    // 挂载要渲染的元素
    el: '#app'
    // 定义数据
    data: {
      message: 'hello world!'
    },
    components: {
      'cpn1': {
        template: '#cpn'
      }
    }
  })
</script>

```

结论：Vue组件应该有自己保存数据的地方。

组件数据的存放

- 组件对象也有一个data属性(也可以有methods等属性)，只是这个data属性必须是一个函数。
- 而且这个函数返回一个对象，对象内部保存着数据。

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>组件中的数据存放</title>
6  </head>
7  <body>
8  <!--2.定义一个div元素-->
9    <div id="app">
10      <cpn1></cpn1>
11    </div>
12
13  <!--template标签-->
14  <template id="cpn">
15    <div>
16      <h2>{{title}}</h2>
17      <p>hello vue.js!!!</p>
18    </div>
19  </template>
20  <script src="../../js/vue.js"></script>
21  <script>
22
23  // 注册全局组件
24  Vue.component('cpn1', {
25    // 定义模板
26    template: '#cpn',

```

```

27 // 组件对象有一个data属性，它是个函数
28 data(){
29     // 返回一个实例的值
30     return {
31         title: 'guardwhy'
32     }
33 }
34 })
35 // 创建对象
36 const app = new Vue({
37     // 挂载要管理的元素
38     el: '#app',
39 })
40 </script>
41 </body>
42 </html>

```

执行结果



8.8.2 组件中的data为啥是函数

具体原因

首先，如果不是一个函数，Vue直接就会报错。

其次，原因是**在于Vue让每个组件对象都返回一个新的对象，因为如果是同一个对象的，组件在多次使用后会相互影响。**

代码示例

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>组件中的data为啥是函数</title>
6 </head>

```

```
7 <body>
8   <div id="app">
9     <cpn1></cpn1>
10    <cpn1></cpn1>
11  </div>
12  <!--创建模板-->
13  <template id="cpn">
14    <div>
15      <h2>当前计数:{{counter}}</h2>
16      <button @click="increment">+</button>
17      <button @click="decrement">-</button>
18    </div>
19  </template>
20  <script src="../js/vue.js"></script>
21  <script>
22    // 1.注册组件
23  Vue.component('cpn1', {
24    // 2.定义模板
25    template: '#cpn',
26    data(){
27      return {
28        // 定义计数器的值
29        counter: 0
30      }
31    },
32    // 定义方法
33    methods: {
34      increment(){
35        this.counter++
36      },
37      decrement(){
38        this.counter--
39      }
40    }
41  })
42  // 创建对象
43  const app = new Vue({
44    // 挂载要管理的元素
45    el: '#app'
46  })
47  </script>
48  </body>
49  </html>
```

执行结果

```
<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <div id="app">
      <div>
        <h2>当前计数:10</h2>
        <button>+</button> [event]
        [空白]
        <button>-</button> [event]
      </div>
      <div>
        <h2>当前计数:5</h2>
        <button>+</button> [event]
        [空白]
        <button>-</button> [event]
      </div>
      <!--创建模板-->
    <template id="cpn"></template>
    <script src="../js/vue.js"></script>
    <script></script>
  </body>
</html>
```

9- 组件的通信

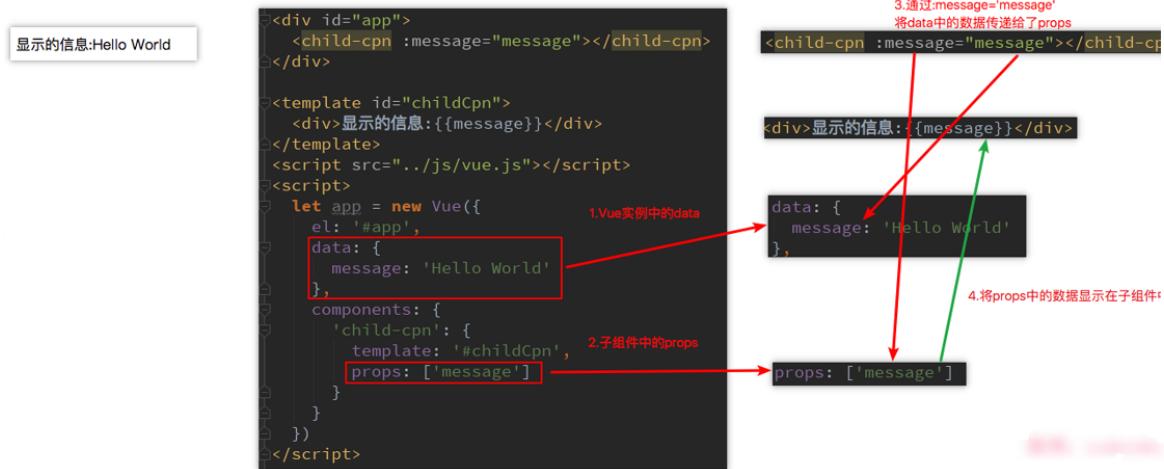
真实的开发中，Vue实例和子组件的通信和父组件和子组件的通信过程是一样的。

9.1 父组件向子组件传递

子组件是不能引用父组件或者Vue实例的数据的。这个时候，并不会让子组件再次发送一个网络请求，而是直接让大组件(父组件)将数据传递给小组件(子组件)。

props基本用法

- 在组件中，使用选项props来声明需要从父级接收到的数据。
- props的值有两种方式
 - 方式一：字符串数组，数组中的字符串就是传递时的名称。
 - 方式二：对象，对象可以设置传递时的类型，也可以设置默认值等。
- props传递案例



代码示例

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>父组件向子组件传递参数</title>
6  </head>
7  <body>
8    <div id="app">
9      <!--3.使用组件-->
10     <!--绑定属性-->
11     <cmessage :cmessage="message" :cstarts="starts"></cmessage>
12   </div>
13
14  <!--创建模板-->
15  <template id="cpn">
16    <div>
17      <ul>
18        <li v-for="item in cstarts">{{item}}</li>
19      </ul>
20      <h3>{{cmessage}}</h3>
21    </div>
22  </template>
23
24  <script src="../js/vue.js"></script>
25  <script>
26    const cpn1 = {
27      // 定义模板
28      template: '#cpn',
29      // 父组件向子组件传递内容(父传子: props)
30      props: {
31        // 类型限制, 提供默认值, 和及其必传值
32        cmessage: {
33          type: String,
34          default: 'Curry',
35          required: true
36        },
37
38        // 类型是数组或者对象类型时候, 默认值必须是一个函数
39        cstarts: {
40          type: Array,
41          default() {

```

```

42         return []
43     }
44   }
45 }
46 }
47 // 创建对象
48 const app = new Vue({
49   // 挂载要管理的元素
50   el: '#app',
51
52   // 定义数据
53   data: {
54     message: 'hello world!',
55     starts: ['kobe', 'Jmaes', 'Curry', 'Duncan']
56   },
57
58   // 注册组件
59   components: {
60     cpn1
61   }
62 })
63 </script>
64 </body>
65 </html>

```

执行结果

The screenshot shows a browser window with the title "父组件向子组件传递参数". The address bar indicates the URL is `localhost:63342/Vue.js/03-vue组件化开发/1-组件化基础/01-父子组件通信`. The developer tools DOM tab is active, displaying the following HTML structure:

```

<!DOCTYPE html>
<html lang="en">
  <head> ...
  </head>
  <body>
    <div id="app">
      <div>
        <ul> ...
        <h3>hello world!</h3>
      </div>
    </div>
    <!--创建模板-->
    <template id="cpn">
      #document-fragment
    </template>
    <script src="../js/vue.js"></script>
    <script>...</script>
  </body>
</html>

```

A red box highlights the `` element and its child `<h3>hello world!</h3>`. A red arrow points from this highlighted area to a red-bordered box containing the rendered output:

- kobe
- Jmaes
- Curry
- Duncan

hello world!

9.2 props驼峰标识问题

注意: props传递中的不支持驼峰标识! ! !

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>props中的驼峰标识</title>
6  </head>
7  <body>
8      <div id="app">
9          <!--这里不支持驼峰标识-->
10         <cpn1 :c-info="info" :child-my-message="message"></cpn1>
11     </div>
12
13     <!--定义模板-->
14     <template id="cpn">
15         <!--定义子组件模板的时候, 要用外层的div将其包裹起来-->
16         <div>
17             <h2>{{cInfo}}</h2>
18             <h2>{{childMyMessage}}</h2>
19         </div>
20     </template>
21     <script src="../js/vue.js"></script>
22     <script>
23         const cpn1 = {
24             template: '#cpn',
25             props: {
26                 cInfo: {
27                     type: Object,
28                     default() {
29                         return {}
30                     }
31                 },
32                 childMyMessage: {
33                     type: String,
34                     default: ''
35                 }
36             }
37         }
38
39         // 创建对象
40         const app = new Vue({
41             // 挂载要管理的元素
42             el: '#app',
43
44             // 定义数据
45             data: {
46                 info: {
47                     name: 'guardwhy',
48                     age: 26,
49                     height: 1.71
50                 },
51                 message: 'curry'
52             },
53             // 注册组件
```

```

54     components: {
55       cpn1
56     }
57   })
58 </script>
59 </body>
60 </html>

```

执行结果

The screenshot shows the browser's developer tools with the Network tab selected. It displays the component structure and the props passed to it. The props object is shown as:

```
{ "name": "guardWhy", "age": 26, "height": 1.71 }
```

Below the props, the word "curry" is visible, likely indicating the component name or a specific method.

9.3 子级向父级传递(自定义事件)

什么时候需要自定义事件呢?

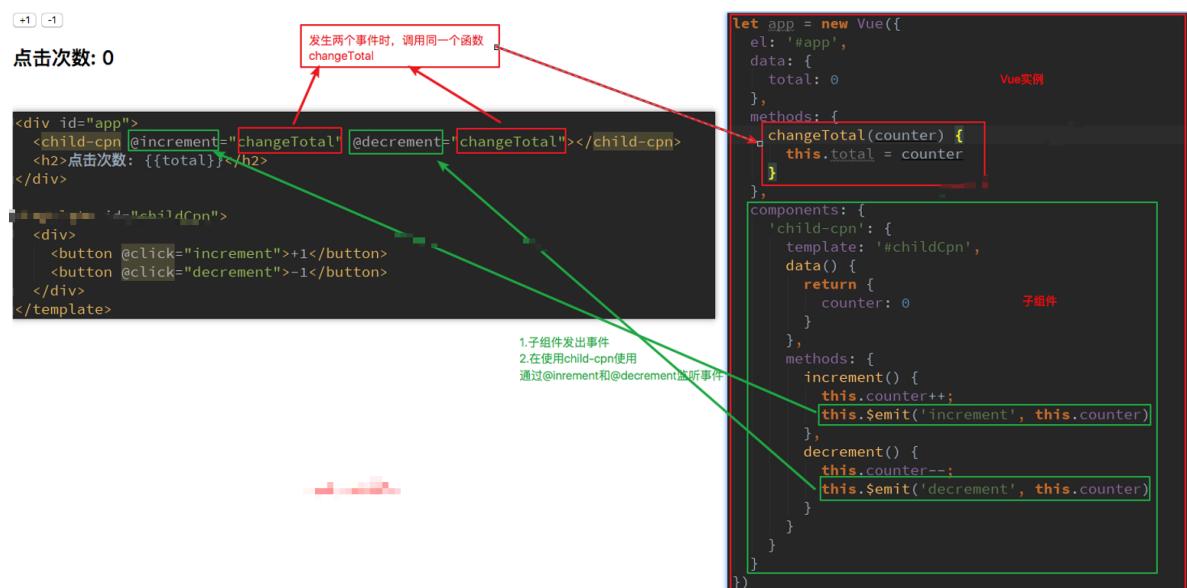
当子组件需要向父组件传递数据时，就要用到自定义事件了。

之前学习的v-on不仅仅可以用于监听DOM事件，也可以用于组件间的自定义事件。

自定义事件的流程

在子组件中，通过\$emit()来触发事件。

在父组件中，通过v-on来监听子组件事件。



代码示例

```

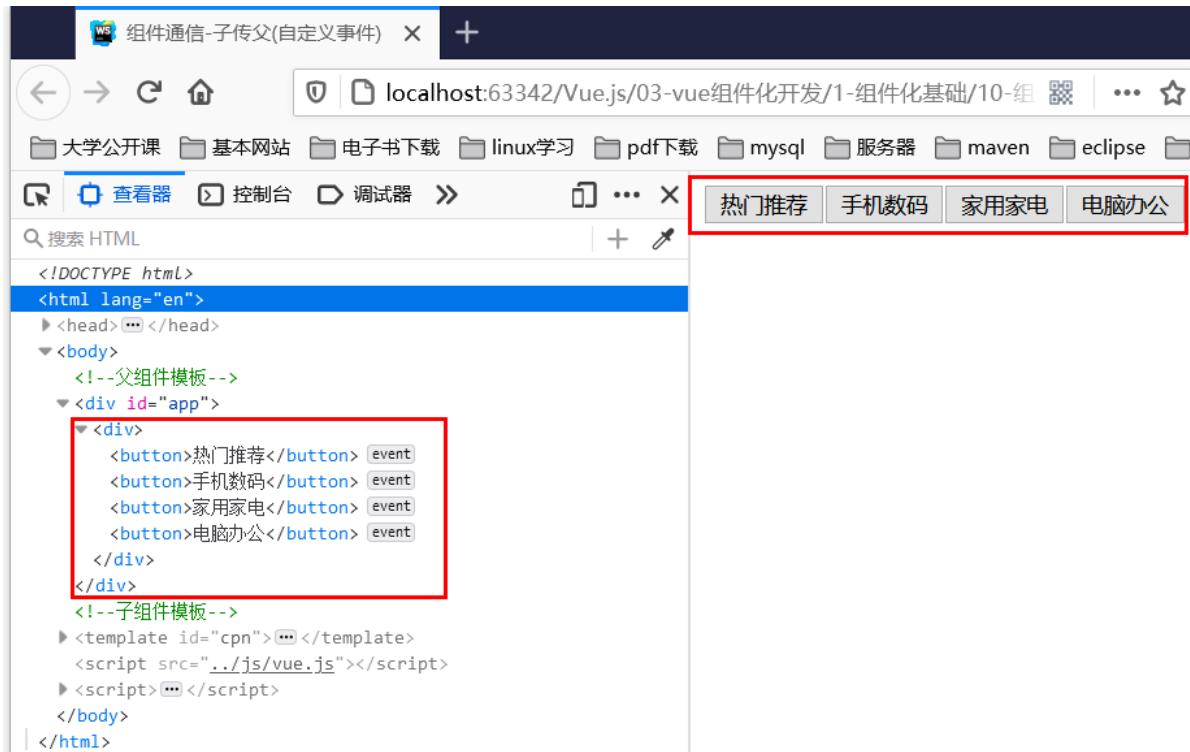
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">

```

```
5   <title>子传父自定义事件</title>
6 </head>
7 <body>
8 <!--父组件模板-->
9   <div id="app">
10    <!--父组件监听(v-on)事件-->
11    <cpn1 @item-click="cpnClick"></cpn1>
12  </div>
13
14
15 <!--子组件模板-->
16 <template id="cpn">
17   <div>
18     <button v-for="item in categories"
19       @click="btnClick(item)">
20       {{item.name}}
21     </button>
22   </div>
23 </template>
24
25 <script src="../js/vue.js"></script>
26 <script>
27
28 // 子组件
29 const cpn1 = {
30   // 定义模板
31   template: '#cpn',
32   data(){
33     return{
34       categories: [
35         {id: 'aaa', name: '热门推荐'},
36         {id: 'bbb', name: '手机数码'},
37         {id: 'ccc', name: '家用家电'},
38         {id: 'ddd', name: '电脑办公'},
39       ]
40     }
41   },
42
43   methods: {
44     btnClick(item){
45       // console.log(item);
46       // 发射事件:自定义事件
47       this.$emit('item-click', item)
48     }
49   }
50 }
51
52 // 父组件
53 const app = new Vue({
54   // 挂载要管理的元素
55   el: '#app',
56   components : {
57     cpn1
58   },
59
60   methods: {
61     cpnClick(item){
62       // 输出item
63     }
64   }
65 })
```

```
63         console.log('cpnClick', item);
64     }
65   }
66 })
67 </script>
68 </body>
69 </html>
```

执行结果



The screenshot shows a browser window with the URL `localhost:63342/Vue.js/03-vue组件化开发/1-组件化基础/10-组`. The developer tools sidebar is open, showing the DOM structure. A red box highlights a section of the DOM where multiple buttons are rendered. Another red box highlights the '热门推荐' tab in the browser's tab bar, indicating it is the active tab.

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <!--父组件模板-->
    <div id="app">
      <div>
        <button>热门推荐</button> event
        <button>手机数码</button> event
        <button>家用家电</button> event
        <button>电脑办公</button> event
      </div>
    </div>
    <!--子组件模板-->
    <template id="cpn">...</template>
    <script src="../js/vue.js"></script>
    <script>...</script>
  </body>
</html>
```

9.4 组件通信集合双向绑定

代码示例

方式一

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>组件通信(双向绑定)</title>
6  </head>
7  <body>
8  <!--父组件模板-->
9    <div id="app">
10      <cpn1 :number1="num1" :number2="num2"
11        @num1change="num1change"
12        @num2change="num2change"></cpn1>
13
14    </div>
15  <!--子组件模板-->
16  <template id="cpn">
17    <div>
18      <h2>props: {{number1}}</h2>
19      <h2>data: {{dnumber1}}</h2>
```

```
20      <!--    <input type="text" v-model="dnumber1"> -->
21      <input type="text" :value="dnumber1" @input="num1Input">
22      <h2>props:{{number2}}</h2>
23      <h2>data:{{dnumber2}}</h2>
24      <!-- <input type="text" v-model="dnumber2">-->
25
26      <input type="text" :value="dnumber2" @input="num2Input">
27      </div>
28  </template>
29  <script src="../js/vue.js"></script>
30
31 <script>
32 // 父组件
33 const app = new Vue({
34     // 挂载要管理的元素
35     el: '#app',
36     data: {
37         // 定义属性
38         num1: 1,
39         num2: 0
40     },
41     // 实现方法
42     methods: {
43         num1change(value){
44             this.num1 = parseFloat(value)
45         },
46         num2change(value){
47             this.num2 = parseFloat(value)
48         }
49     },
50     // 注册组件
51     components: {
52         // 子组件
53         cpn1:{
54             // 定义模板
55             template: '#cpn',
56             // 父子之间的通信
57             props:{
58                 number1: Number,
59                 number2: Number
60             },
61             // 添加一个data属性,从而实现双向绑定
62             data(){
63                 return{
64                     dnumber1: this.number1,
65                     dnumber2: this.number1
66                 }
67             },
68             methods: {
69                 num1Input(event){
70                     // 将input中的value赋值到dnumber中
71                     this.dnumber1 = event.target.value;
72                     // 让父组件可以修改值,发出一个事件
73                     this.$emit('num1change', this.dnumber1);
74
75                     // 修改dnumber2的值
76                     this.dnumber2 = this.dnumber1 * 100;
77                     this.$emit('num2change', this.dnumber2)
78                 }
79             }
80         }
81     }
82 })
```

```

78 },
79     num2Input(event){
80         // 将input中的value赋值到dnumber中
81         this.dnumber2 = event.target.value;
82         this.$emit('num2change', this.dnumber2);
83
84         // 修改dnumber1的值
85         this.dnumber1 = this.dnumber2 / 100;
86         this.$emit('num1change', this.dnumber1)
87     }
88 }
89 }
90 })
91 </script>
92 </body>
93 </html>

```

执行结果

The screenshot shows a browser window with the title "组件通信(双向绑定)". The address bar indicates the URL is "localhost:63342/Vue.js/03-vue组件化开发/1-组件化基础/11-组". The developer tools sidebar has "查看器" selected. In the main pane, there are two sections of output:

- props: 1111**
- data:1111**
- 1111** (in a text input field)
- props:111100**
- data:111100**
- 111100** (in a text input field)

The code editor on the left shows the Vue.js template and script blocks. A red box highlights the section of the template where the props and data are being used.

方式二

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>组件通信(双向绑定)</title>
6  </head>
7  <body>
8
9  <div id="app">
10     <cpn :number1="num1">

```

```
11      :number2="num2"
12      @num1change="num1change"
13      @num2change="num2change"/>
14  </div>
15
16 <template id="cpn">
17   <div>
18     <h2>props:{{number1}}</h2>
19     <h2>data:{{dnumber1}}</h2>
20     <input type="text" v-model="dnumber1">
21     <h2>props:{{number2}}</h2>
22     <h2>data:{{dnumber2}}</h2>
23     <input type="text" v-model="dnumber2">
24   </div>
25 </template>
26
27 <script src="../js/vue.js"></script>
28 <script>
29   const app = new Vue({
30     el: '#app',
31     data: {
32       num1: 1,
33       num2: 0
34     },
35     methods: {
36       num1change(value) {
37         this.num1 = parseFloat(value)
38       },
39       num2change(value) {
40         this.num2 = parseFloat(value)
41       }
42     },
43     components: {
44       cpn: {
45         template: '#cpn',
46         props: {
47           number1: Number,
48           number2: Number,
49           name: ''
50         },
51         data() {
52           return {
53             dnumber1: this.number1,
54             dnumber2: this.number2
55           }
56         },
57         watch: {
58           dnumber1(newValue) {
59             this.dnumber2 = newValue * 100;
60             this.$emit('num1change', newValue);
61           },
62           dnumber2(newValue) {
63             this.number1 = newValue / 100;
64             this.$emit('num2change', newValue);
65           }
66         }
67       }
68     }
69   }
70 </script>
```

```
69    })
70  </script>
71
72 </body>
73 </html>
```

执行结果

The screenshot shows a browser window with the title "组件通信(双向绑定)". The address bar indicates the URL is "localhost:63342/Vue.js/03-vue组件化开发/1-组件化基础/11-组". The developer tools sidebar has "查看器" selected. In the "Elements" tab, the DOM tree is displayed. A specific section of the code is highlighted with a red box:

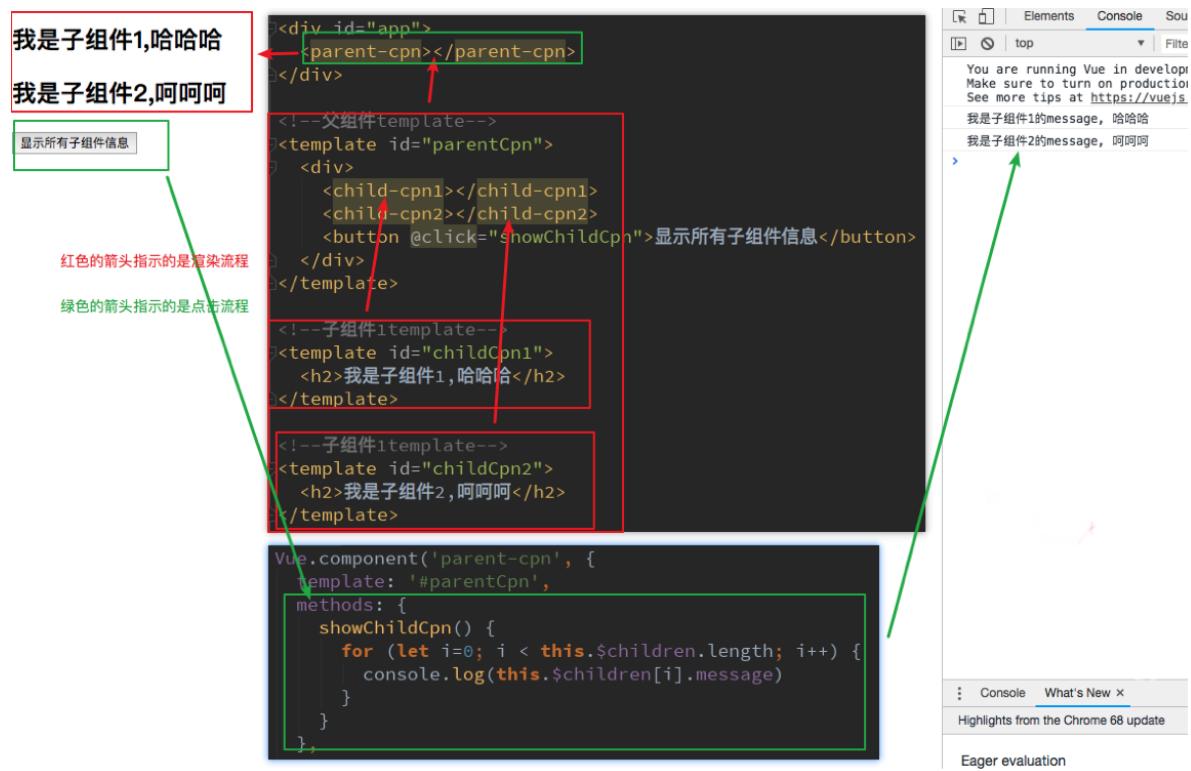
```
<h2>props: 1111</h2>
<h2>data:1111</h2>
<input type="text"> [event]
<h2>props:111100</h2>
<h2>data:111100</h2>
<input type="text"> [event]
```

On the right side of the developer tools, there are two text boxes. The top one contains "props: 1111" and the bottom one contains "data:1111". Below these, another set of text boxes shows "props:111100" and "data:111100".

9.5 父组件直接访问子组件

9.5.1 \$children的访问

- this.\$children是一个数组类型，它包含所有子组件对象。
- 能通过一个遍历，取出所有子组件的message状态。



9.5.2 \$children的缺陷

- 通过\$children访问子组件时，是一个数组类型，访问其中的子组件必须通过索引值。
- 但是当子组件过多，我们需要拿到其中一个时，往往不能确定它的索引值，甚至还可能会发生变化。
- 有时候，想明确获取其中一个特定的组件，这个时候就可以使用\$refs。

9.5.3 \$refs的使用

- \$refs和ref指令通常是一起使用的。
- 首先，通过ref给某一个子组件绑定一个特定的ID。
- 其次，通过this.\$refs.ID就可以访问到该组件了。

```

1 <html lang="en">
2   <head>
3     <meta charset="UTF-8">
4     <title>组件访问(父访问子)</title>
5   </head>
6   <body>
7     <!--2.定义一个div元素-->
8     <div id="app">
9       <cpn1></cpn1>
10      <cpn1></cpn1>
11      <cpn1 ref="aaa"></cpn1>
12      <button @click="btnClick">按钮</button>
13    </div>
14
15   <!--子组件模板-->
16   <template id="cpn">
17     <div>
18       <h2>我是子组件</h2>
19     </div>
20   </template>
21
22   <script src="../js/vue.js"></script>

```

```
23 <script>
24 // 创建对象
25 const app = new Vue({
26   // 挂载要管理的元素
27   el: '#app',
28   // 定义数据
29   data: {
30     message: 'hello world!'
31   },
32   methods: {
33     btnClick(){
34       // 1.$children
35       /*
36         console.log(this.$children);
37         for(let c of this.$children){
38           console.log(c.name);
39           c.showMessage();
40         }
41         console.log(this.$children[1].name);
42       */
43
44       // 2.$refs ==> 对象类型, 默认这是一个空对象
45       console.log(this.$refs.aaa.name);
46     }
47   },
48   // 创建子组件
49   components: {
50     cpn1: {
51       template: '#cpn',
52       data(){
53         return{
54           name: '我是子组件的name'
55         }
56       },
57       methods:{
58         showMessage(){
59           console.log("showMessage!!!!");
60         }
61       }
62     }
63   }
64 })
65 </script>
66 </body>
67 </html>
```

执行结果



9.6 子组件直接访问父组件

如果想在子组件中直接访问父组件，可以通过\$parent。

The diagram illustrates the structure of a parent-child component relationship:

- Parent Component Structure:**
 - HTML: ``
 - Template: ``
 - Child Component Structure:
 - HTML: `显示父组件信息`
 - Script: `showParent() { console.log(this.\$parent.message); }`

A red arrow points from the "显示父组件信息" button in the child component's template to the `showParent` method in its script, indicating the flow of control.

Console Output: The right side shows the browser's developer tools console output:

```
You are running Vue in development mode. Make sure to turn on production mode when deploying for production. See more tips at https://vuejs.org/guide/deployment.html
我是父组件, 嘿嘿嘿
我是父组件, 嘿嘿嘿
```

注意事项

- 尽管在Vue开发中，允许通过\$parent来访问父组件，但是在真实开发中尽量不要这样做。
- 子组件应该尽量避免直接访问父组件的数据，因为这样耦合度太高了。
- 如果将子组件放在另外一个组件之内，很可能该父组件没有对应的属性，往往会引起问题。
- 另外，更不好做的是通过\$parent直接修改父组件的状态，那么父组件中的状态将变得飘忽不定，很不利于调试和维护。

代码示例

```
1 <!DOCTYPE html>
```

```
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>组件访问(子访问父)</title>
6   </head>
7   <body>
8
9   <div id="app">
10    <cpn></cpn>
11  </div>
12
13 <template id="cpn">
14   <div>
15     <h2>我是cpn组件</h2>
16     <ccpn></ccpn>
17   </div>
18 </template>
19
20 <template id="ccpn">
21   <div>
22     <h2>我是子组件</h2>
23     <button @click="btnClick">按钮</button>
24   </div>
25 </template>
26
27 <script src="../../js/vue.js"></script>
28 <script>
29   const app = new Vue({
30     el: '#app',
31     data: {
32       message: 'hello vue.js!!!'
33     },
34     components: {
35       cpn: {
36         template: '#cpn',
37         data() {
38           return {
39             name: '我是cpn组件的name'
40           }
41         },
42         components: {
43           ccpn: {
44             template: '#ccpn',
45             methods: {
46               btnClick() {
47                 // 1.访问父组件$parent
48                 // console.log(this.$parent);
49                 // console.log(this.$parent.name);
50
51                 // 2.访问根组件$root
52                 console.log(this.$root);
53                 console.log(this.$root.message);
54               }
55             }
56           }
57         }
58       }
59     }
60   }
61 
```

```
60  })
61 </script>
62 </body>
63 </html>
```

执行结果



10- slot

10.1 为什么使用slot

组件的插槽

组件的插槽也是为了让封装的组件更加具有扩展性，让使用者可以决定组件内部的一些内容到底展示什么。

slot基本使用

在子组件中，使用特殊的元素就可以为子组件开启一个插槽。

该插槽插入什么内容取决于父组件如何使用。

中的内容表示，如果没有在该组件中插入任何其他内容，就默认显示该内容。

代码示例

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>slot-插槽的基本使用</title>
6  </head>
7  <body>
```

```
8 <!--2.定义一个div元素-->
9   <div id="app">
10    <cpn></cpn>
11    <cpn><span>php是最好的语言!!!</span></cpn>
12    <cpn><i>python一出，谁与争锋!!</i></cpn>
13  </div>
14
15  <template id="cpn">
16    <div>
17      <h2>我是组件</h2>
18      <p>我是子组件模板</p>
19      <!--设定默认值-->
20      <slot><button>按钮</button></slot>
21    </div>
22  </template>
23
24  <script src="../js/vue.js"></script>
25  <script>
26    // 创建对象
27    const app = new Vue({
28      // 挂载要管理的元素
29      el: '#app',
30
31      // 定义数据
32      data: {
33        message: 'hello world!'
34      },
35      components: {
36        cpn:{ 
37          template:'#cpn'
38        }
39      }
40    })
41  </script>
42 </body>
43 </html>
```

执行结果

```

<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <!--2.定义一个div元素-->
    <div id="app">
      <div>
        <h2>我是组件</h2>
        <p>我是子组件模板</p>
        <button>按钮</button>
      </div>
      <div>
        <h2>我是组件</h2>
        <p>我是子组件模板</p>
        <span>php是最好的语言!!!</span>
      </div>
      <div>
        <h2>我是组件</h2>
        <p>我是子组件模板</p>
        <i>python一出，谁与争锋!!</i>
      </div>
    </div>
    <template id="cpn"></template>
    <script src="../js/vue.js"></script>
    <script></script>
  </body>
</html>

```

10.2 具名插槽slot

当子组件的功能复杂时，子组件的插槽可能并非是一个。

- 比如封装一个导航栏的子组件，可能就需要三个插槽，分别代表左边、中间、右边。
- 外面在给插槽插入内容时，如何区分插入的是哪一个呢？这个时候，就需要给插槽起一个名字。

代码示例

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>具名插槽的使用</title>
6  </head>
7  <body>
8    <!--2.定义一个div元素-->
9    <div id="app">
10      <cpn><span slot="center">标题</span></cpn>
11      <cpn><button slot="left">返回</button></cpn>
12    </div>
13
14    <!--子组件模板-->
15    <template id="cpn">
16      <div>
17        <slot name="left"><span>左边</span></slot>
18        <slot name="center"><span>中间</span></slot>

```

```

19         <slot name="right"><span>右边</span></slot>
20     </div>
21 </template>

22
23 <script src="../../js/vue.js"></script>
24 <script>
25     // 创建对象
26     const app = new Vue({
27         // 挂载要管理的元素
28         el: '#app',
29
30         // 定义数据
31         data: {
32             message: 'hello world!'
33         },
34         components: {
35             cpn: {
36                 template: '#cpn'
37             }
38         }
39     })
40 </script>
41 </body>
42 </html>

```

执行结果

```

<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <!--2. 定义一个div元素-->
    <div id="app">...</div>
    <!--子组件模板-->
    <template id="cpn">...</template>
    <script src="../../js/vue.js"></script>
    <script>...</script>
  </body>
</html>

```

10.3 编译作用域

父组件模板的所有东西都会在父级作用域内编译，子组件模板的所有东西都会在子级作用域内编译。

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">

```

```
5   <title>编译的作用域</title>
6 </head>
7 <body>
8
9 <div id="app">
10  <!--使用变量时，先看在哪个模板里面-->
11  <cpn v-show="isShow"></cpn>
12  <!-- <cpn v-for="item in names"></cpn> -->
13 </div>
14
15 <template id="cpn">
16  <div>
17    <h2>我是子组件</h2>
18    <p>我是内容，哈哈哈</p>
19    <!-- <button v-show="isShow">按钮</button> -->
20  </div>
21 </template>
22
23 <script src="../js/vue.js"></script>
24 <script>
25  /*实例*/
26  const app = new Vue({
27    el: '#app',
28    data: {
29      message: '你好啊',
30      isShow: true
31    },
32    /*组件*/
33    components: {
34      cpn: {
35        template: '#cpn',
36        data() {
37          return {
38            isShow: false
39          }
40        }
41      },
42    }
43  })
44 </script>
45
46 </body>
47 </html>
```

执行结果

The screenshot shows a browser window with developer tools open. The title bar says '编译的作用域'. The address bar shows 'localhost:63342/Vue.js/03-vue组件化开发/2-组件化高'. The toolbar includes icons for back, forward, search, and various developer tools like '查看器' (Inspector). The main area shows the HTML code:

```
<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <div id="app">
      <div>
        <h2>我是子组件</h2>
        <p>我是内容，哈哈哈</p>
      </div>
    </div>
    <template id="cpn"></template>
    <script src="../js/vue.js"></script>
    <script></script>
  </body>
</html>
```

The content within the first nested div is highlighted with a red box. To the right, the rendered output is shown:

我是子组件
我是内容, 哈哈哈

10.4 作用域插槽

父组件替换插槽的标签，但是内容由子组件来提供。

代码示例

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>作用域插槽</title>
6  </head>
7  <body>
8    <!--作用域插槽：父组件替换插槽的标签，但是内容由子组件来提供-->
9    <div id="app">
10      <cpn></cpn>
11      <!--&lt;!&ndash;目的是获取子组件中的starts&ndash;&gt;-->
12      <cpn>
13        &lt;!&ndash;方式1&ndash;&gt;
14        <template slot-scope="slot">
15          <span v-for="item in slot.data">-{{item}}</span>
16        </template>
17      </cpn>-->
18
19      <!--目的是获取子组件中的starts-->
20      <cpn>
21        <!--方式2-->
22        <template slot-scope="slot"><!--引用插槽对象-->
23          <span>{{slot.data.join(' - ')}}</span>
24        </template>
25      </cpn>
26    </div>
```

```
27      <!--子组件模板-->
28      <template id="cpn">
29          <div>
30              <!--子组件传递给父组件-->
31              <slot :data="starts">
32                  <ul>
33                      <li v-for="item in starts">{{item}}</li>
34                  </ul>
35              </slot>
36          </div>
37      </template>
38      <script src="../js/vue.js"></script>
39      <script>
40          // 创建对象
41          const app = new Vue({
42              // 挂载要管理的元素
43              el: '#app',
44
45              // 定义数据
46              data: {
47                  message: 'hello world!'
48              },
49              components: {
50                  cpn: {
51                      template: '#cpn',
52                      data(){
53                          return {
54                              starts: ['kobe', 'Jmaes', 'Curry', 'Duncan']
55                          }
56                      }
57                  }
58              }
59          })
60      </script>
61  </body>
62</html>
```

执行结果

The screenshot shows the browser's developer tools with the '查看器' (Inspector) tab selected. The URL is `localhost:63342/Vue.js/03-vue组件化开发/2-组件化高级/03-作用于插槽案列`. The DOM tree is displayed, with a red box highlighting a specific section of the component's template. The template includes a parent component's logic and a child component's output, demonstrating how a parent component can replace a slot placeholder with its own content.

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <!--作用域插槽：父组件替换插槽的标签，但是内容由子组件来提供-->
    <div id="app">
      <div>
        <ul>...</ul>
      </div>
      <div>
        <span>kobe - Jmaes - Curry - Duncan</span>
      </div>
    </div>
    <!--子组件模板-->
    <template id="cpn">...</template>
    <script src="../js/vue.js"></script>
    <script>...</script>
  </body>
</html>
```

11- 前端模块化

11.1 export基本使用

export指令用于导出变量

```
1 | export let name = 'kobe';
2 | export let age = 18;
3 | export let flag = true;
```

上面的代码还有另外一种写法

```
1 | let name = 'kobe';
2 | let age = 18;
3 | let flag = true;
4 | export {name, age, flag}
```

导出函数或类

```
1 | function test(num){
2 |   console.log(num);
3 |
4 | class Person{
5 |   constructor(name, age){
6 |     this.name = name;
7 |     this.age = age;
8 |   }
9 |
10 |   run(){
11 |     console.log(this.name + '在学习');
```

```
12     }
13 }
14
15 export{test, Person}
```

export default

一个模块中包含某个的功能，并不希望给这个功能命名，而且让导入者可以自己来命名。

```
1 // info1.js
2 export default function(){
3     console.log('default function');
4 }
```

来到main.js中，这样使用就可以了

```
1 import function from './info1.js'
```

需要注意

export default在同一个模块中，不允许同时存在多个。

11.2 import基本使用

使用export指令导出了模块对外提供的接口，下面我们就可以通过import命令来加载对应的这个模块了

首先，我们需要在HTML代码中引入两个js文件，并且类型需要设置为module。

```
1 <script src="info2.js" type="module"></script>
2 <script src="main.js" type="module"></script>
```

import指令用于导入模块中的内容，比如main.js的代码。

```
1 import {name, age, flag} from "./info1.js"
2 console.log(name, age, flag);
```

11.3 代码示例

info1.js

```
1 // 定义变量
2 let name = 'kobe';
3 let age = 18;
4 let flag = true;
5
6 // 定义函数
7 function sum(num1, num2) {
8     return num1 + num2
9 }
10
11 // 条件判断
12 if (flag){
13     console.log(sum(20, 30));
14 }
15
```

```

16 // 1. 导出模块方式
17 export {
18   flag, sum,
19 }
20
21 // 2. 导出方式2
22 export let num1 = 100;
23 export let height = 1.71
24
25 // 3. 导出函数/类
26 export function mul(num1, num2) {
27   return num1 * num2
28 }
29
30 export class Person {
31   run(){
32     console.log('python是最优雅的语言!!!');
33   }
34 }
35
36 // 5.export default
37 const address = '广东省广州市'
38 export default address

```

info2.js

```

1 // 导入模块
2 import {sum} from './info1.js'
3 // 定义变量
4 let name = 'Curry';
5 let flag = false;
6
7 console.log(sum(100, 200));
8
9 // 3. 导入 export 的 function/class
10 import {mul, Person} from "./info1.js";
11 console.log(mul(30, 50));

```

main.js

```

1 // 1. 导入的{}中定义变量
2 import {flag, sum} from "./info1.js";
3
4 // 条件判断
5 if(flag){
6   console.log('kobe是mvp');
7   console.log(sum(20, 30));
8 }
9
10 // 2. 直接导入export 定义的变量
11 import {num1, height} from "./info1.js";
12
13 // 3. 导入函数和类对象
14 import {mul, Person} from "./info1.js";
15 console.log(mul(30, 50));
16

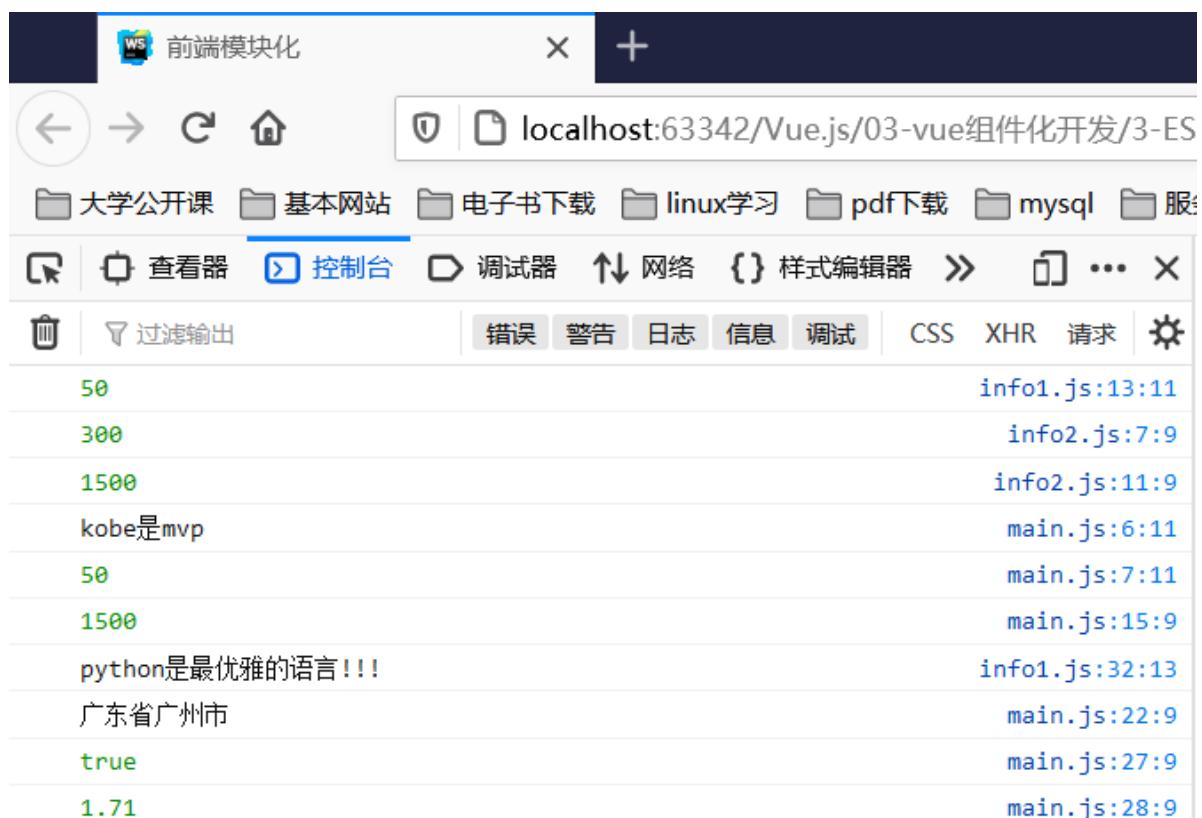
```

```
17 const p = new Person();
18 p.run();
19
20 // 4. 导入export default中的内容
21 import address from "./info1.js";
22 console.log(address);
23
24 // 5. 统一全部导入
25 import * as A from './info1.js'
26 // 打印结果
27 console.log(A.flag);
28 console.log(A.height);
```

index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>前端模块化</title>
6 </head>
7 <body>
8   <!--引入js文件-->
9   <script src="info1.js" type="module"></script>
10  <script src="info2.js" type="module"></script>
11  <script src="main.js" type="module"></script>
12 </body>
13
14 </html>
```

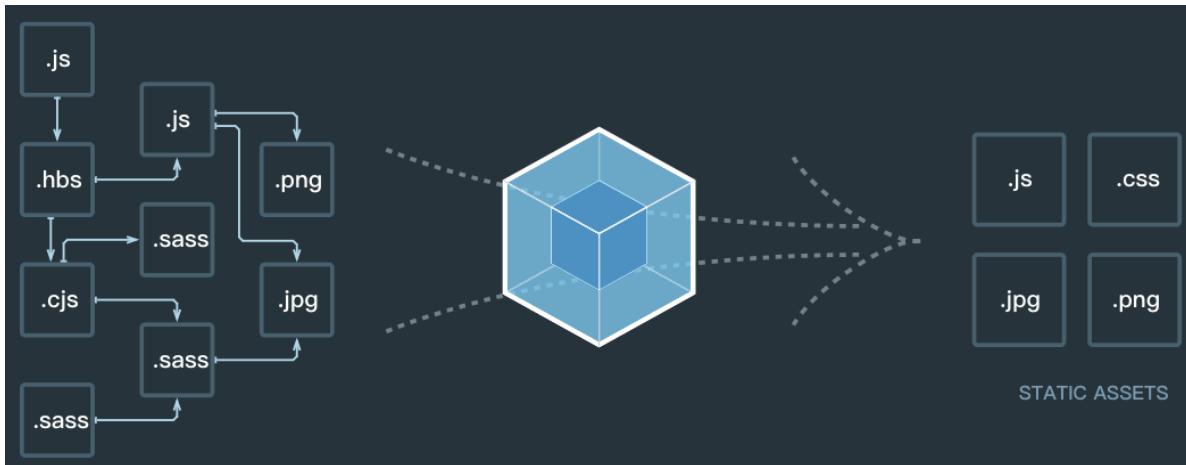
执行结果



12 - webpack 详解

12.1 什么是Webpack?

从本质上来说，webpack是一个现代的JavaScript应用的静态模块打包工具。



前端模块化

- 在ES6之前，我们要想进行模块化开发，就必须借助于其他的工具，让我们可以进行模块化开发。
- 并且在通过模块化开发完成了项目后，还需要处理模块间的各种依赖，并且将其进行整合打包。
- 而webpack其中一个核心就是让我们可能进行模块化开发，并且会帮助我们处理模块间的依赖关系。
- 而且不仅仅是JavaScript文件，CSS、图片、json文件等等在webpack中都可以被当做模块来使用。

打包

就是将webpack中的各种资源模块进行打包合并成一个或多个包(Bundle)。

并且在打包的过程中，还可以对资源进行处理，比如压缩图片，将scss转成css，将ES6语法转成ES5语法，将TypeScript转成JavaScript等等操作。

webpack安装

安装webpack首先需要安装Node.js，Node.js自带了软件包管理工具npm。NPM官方的管理的包都是从<http://npmjs.com>下载的，但是这个网站在国内速度很慢。这里推荐使用淘宝 NPM 镜像<http://npm.taobao.org/>，淘宝 NPM 镜像是一个完整npmjs.com 镜像，同步频率目前为 10分钟一次，以保证尽量与官方服务同步。

设置镜像和存储地址：

```
1 # 由于npm代码仓库的服务器在国外，由于Great Firewall的缘故，这时可以使用淘宝的npm代码仓库，通过npm安装cnpm
2 npm install -g cnpm --registry=https://registry.npm.taobao.org
3 # 安装成功后，可以通过以下命令查看cnpm版本：
4 cnpm -v
5 #设置npm下载包时保存在本地的地址(建议英文目录)，通过cnpm来操作下载速度会得到很大提升，但包的版本不一定是最新的。
6 cnpm config set prefix "E:\\Develop\\repo_npm"
7 #查看cnpm配置信息
8 cnpm config list
```

查看自己的node版本

```
1 | node -v
```

全局安装webpack(这里我先指定版本号3.6.0，因为vue cli2依赖该版本)

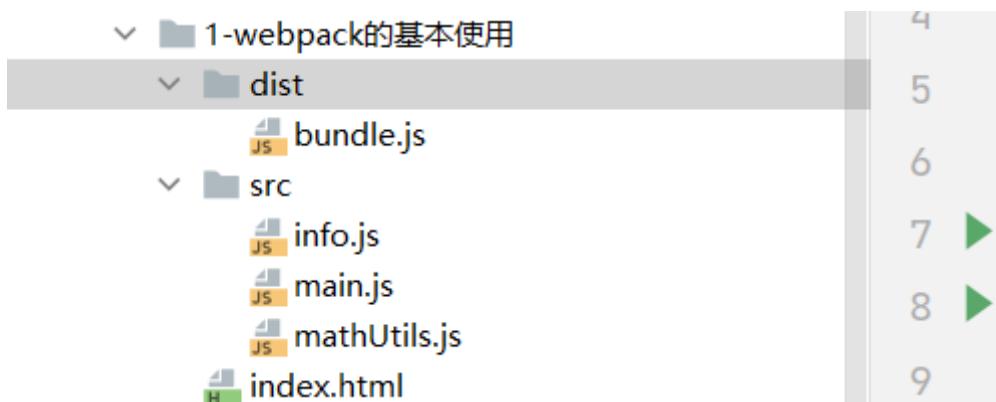
```
1 | cnpm install webpack@3.6.0 -g
```

查看webpack版本

```
1 | webpack --version
```

12.2 Webpack 初体验

项目目录



文件和文件夹解析:

- dist文件夹：用于存放之后打包的文件。
- src文件夹：用于存放我们写的源文件
 - main.js：项目的入口文件。具体内容查看下面详情。
 - mathUtils.js：定义了一些数学工具函数，可以在其他地方引用，并且使用。
- index.html：浏览器打开展示的首页html，package.json：通过npm init生成的，npm包管理的文件。

mathUtils.js

```
1 | function add(num1, num2) {  
2 |   return num1 + num2  
3 | }  
4 |  
5 | function mul(num1, num2) {  
6 |   return num1 * num2  
7 | }  
8 |  
9 | // 模块导出  
10| module.exports = {  
11|   add,  
12|   mul  
13| }
```

info.js

```
1 // 导出模块
2 export const name = 'Guardwhy';
3 export const age = 18;
4 export const height = 1.71
```

main.js

```
1 // 1. 使用commonjs的模块化导入
2 const {add, mul} = require('./mathutils')
3
4 // 输出结果
5 console.log(add(20, 30));
6 console.log(mul(20, 30));
7
8 // 2. 使用es6的模块化规范
9 import {name, age, height} from "./info";
10
11 console.log(name);
12 console.log(age);
13 console.log(height);
```

index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>webpack基本使用</title>
6 </head>
7 <body>
8 <!--引入js文件-->
9 <script src="dist/bundle.js"></script>
10 </body>
11 </html>
```

webpack的指令打包

```
1 webpack src/main.js dist/bundle.js
```

```
E:\workspace\Web\06-Vue.js\Vue.js\04-webpack的使用\1-webpack的基本使用>webpack src/main.js dist/bundle.js
Hash: 362419044c935d8b7502
Version: webpack 3.6.0
Time: 42ms
          Asset      Size  Chunks             Chunk Names
bundle.js  3.68 kB     0  [emitted]  main
[0] ./src/main.js 296 bytes {0} [built]
[1] ./src/mathUtils.js 163 bytes {0} [built]
[2] ./src/info.js 100 bytes {0} [built]
```

```
E:\workspace\Web\06-Vue.js\Vue.js\04-webpack的使用\1-webpack的基本使用>|
```

使用打包后的文件

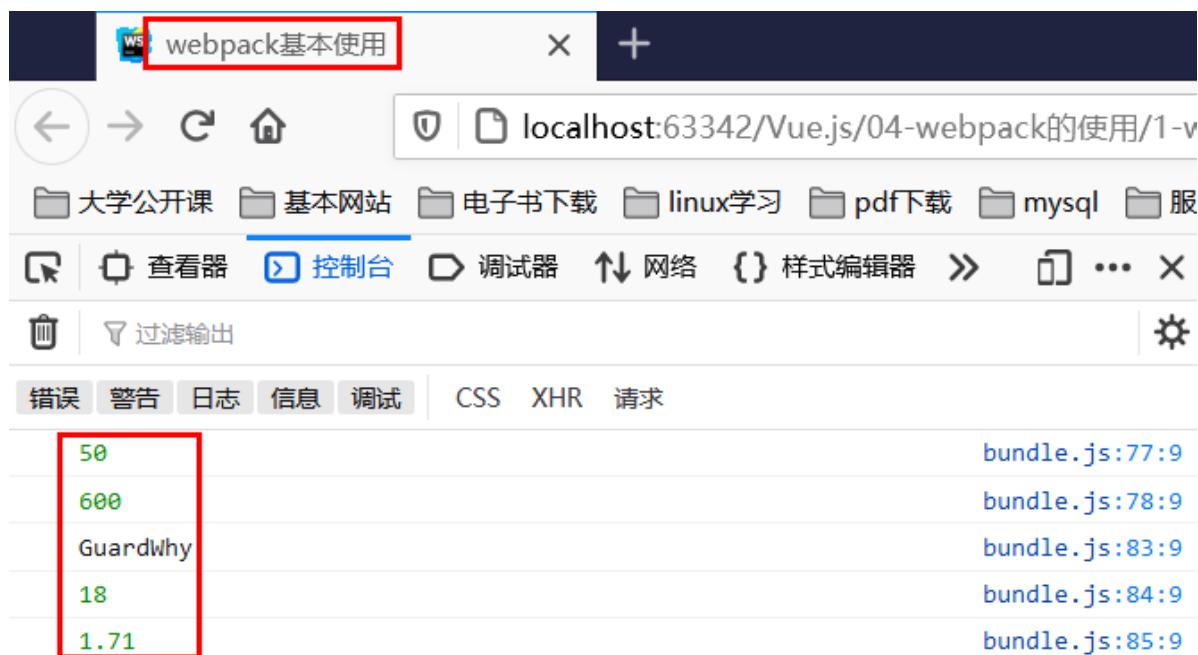
打包后会在dist文件下，生成一个bundle.js文件

bundle.js文件，是webpack处理了项目直接文件依赖后生成的一个js文件，只需要将这个js文件在index.html中引入即可。

index.html

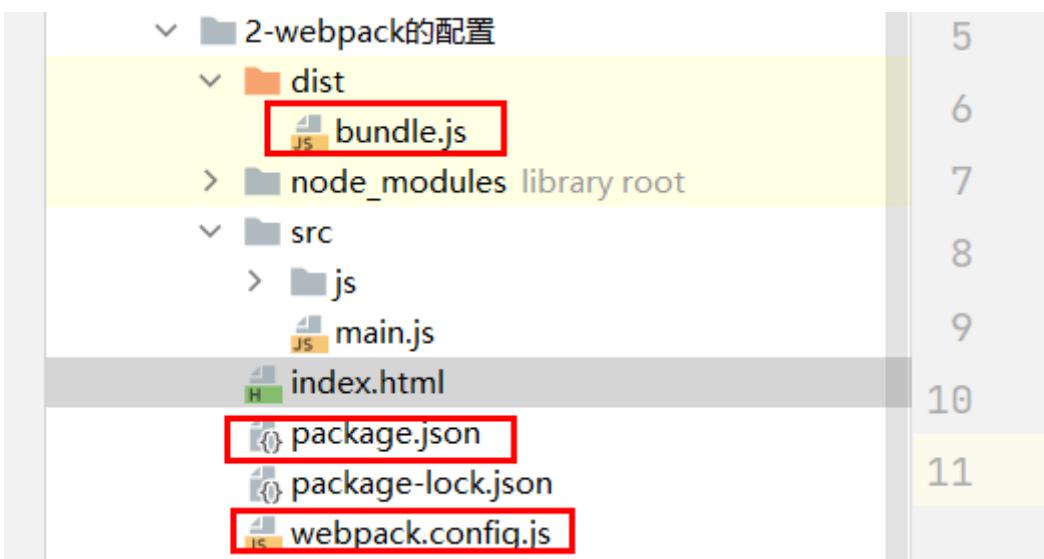
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>webpack基本使用</title>
6 </head>
7 <body>
8   <!--引入js文件-->
9   <script src="dist/bundle.js"></script>
10 </body>
11 </html>
```

执行结果



12.3 webpack的配置

项目目录



配置文件

webpack.config.js

```
1 // 1. 获得路径
2 const path = require('path');
3
4 // 支持导出
5 module.exports = {
6   entry: './src/main.js',
7   output: {
8     path: path.resolve(__dirname, 'dist'),
9     filename: 'bundle.js'
10   },
11 }
```

局部安装webpack

因为一个项目往往依赖特定的webpack版本，全局的版本可能跟这个项目的webpack版本不一致，导出打包出现问题。

所以通常一个项目，都有自己局部的webpack。

```
1 | cnpm install webpack@3.6.0 --save-dev
```

配置package.json

打开项目的命令所在位置，执行 `npm init` 命令，可得到 `package.json` 文件，可以在 `package.json` 的 `scripts` 中定义自己的执行脚本。

```
1 {
2   "name": "guardwhy",
3   "version": "1.0.0",
4   "main": "index.js",
5   "scripts": {
6     "test": "echo \\\"Error: no test specified\\\" && exit 1",
7     "build": "webpack"
8   },
9   "author": "guardwhy",
10  "license": "ISC",
11  "devDependencies": {
12    "webpack": "^3.6.0"
13  },
14  "dependencies": {},
15  "description": ""
16 }
```

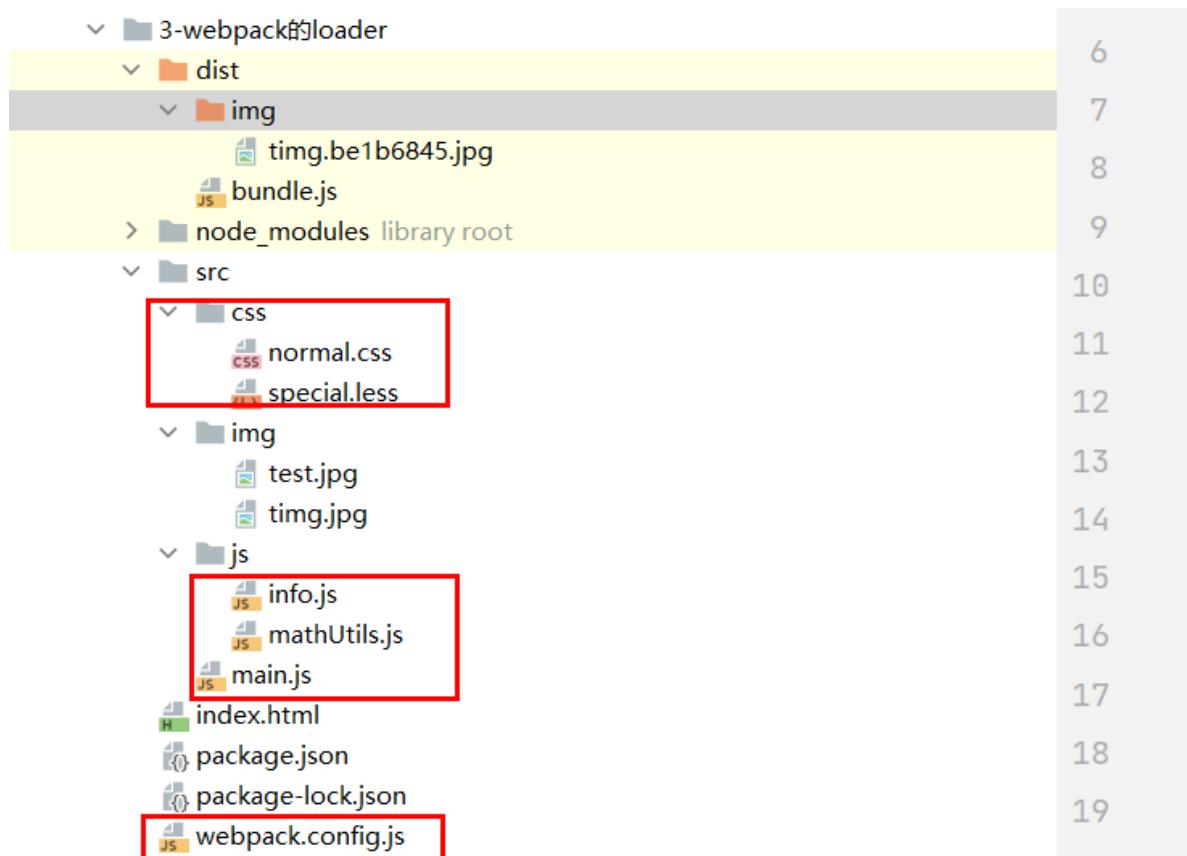
当执行 `npm run build` 命令时候会直接生成 `bundle.js`。

```
E:\workspace\Web\06-Vue.js\Vue.js\04-webpack的使用\2-webpack的配置>npm run build
```

```
> guardwhy@1.0.0 build E:\workspace\Web\06-Vue.js\Vue.js\04-webpack的使用\2-webpack的配置
> webpack
```

```
Hash: 145981deb7c214778d2
Version: webpack 3.6.0
Time: 41ms
Asset      Size  Chunks             Chunk Names
bundle.js  3.69 kB     0  [emitted]  main
  [0] ./src/main.js 302 bytes {0} [built]
  [1] ./src/js/mathUtils.js 163 bytes {0} [built]
```

12.4 什么是loader?



webpack用来做什么呢?

在之前的实例中，主要是用webpack来处理写的js代码，并且webpack会自动处理js之间相关的依赖。在开发中不仅仅有基本的js代码处理，也需要加载css、图片，也包括一些高级的将ES6转成ES5代码，将TypeScript转成ES5代码，将scss、less转成css，将.jsx、.vue文件转成js文件。对于webpack本身的能力来说，对于这些转化是不支持的。那怎么办呢？给webpack扩展对应的loader就可以啦。

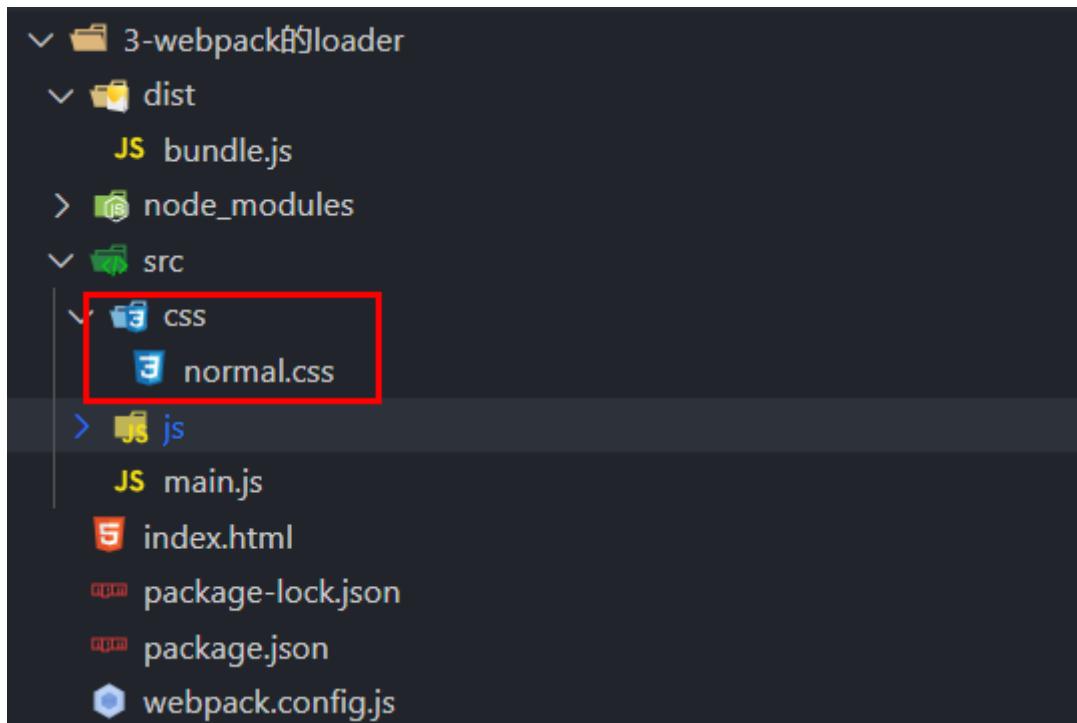
loader使用过程

- 步骤一：通过npm安装需要使用的loader。
- 步骤二：在webpack.config.js中的modules关键字下进行配置。

12.4.1 css文件处理

项目开发过程中，必然需要添加很多的样式，而样式往往写到一个单独的文件中。

在src目录中，创建一个css文件，其中创建一个normal.css文件，也可以重新组织文件的目录结构，将零散的js文件放在一个js文件夹中。



normal.css中的代码非常简单，就是将body设置为red。

```
1 body {  
2     background-color: red;  
3 }
```

但是，这个时候normal.css中的样式会生效吗？当然不会，因为压根就没有引用它。

webpack也不可能找到它，因为只有一个入口，webpack会从入口开始查找其他依赖的文件。

在入口文件中引用

main.js

```
1 // 1. 使用commonjs的模块化导入  
2 const {add, mul} = require('./js/mathUtils')  
3  
4 // 输出结果  
5 console.log(add(20, 30));  
6 console.log(mul(20, 30));  
7  
8 // 2. 使用es6的模块化规范  
9 import {name, age, height} from "./js/info";  
10  
11 console.log(name);  
12 console.log(age);  
13 console.log(height);  
14  
15 // 依赖css文件  
16 require('./css/normal.css')
```

执行 `npm run build` 命令后出现错误！！！

```
linux@Guardwhy MINGW64 /e/workspace/Web/06-Vue.js/04-webpack/3-webpack的loader
$ npm run build

> guardwhy@1.0.0 build E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader
> webpack

Hash: 146c4d7faad9d414b8ae
Version: webpack 3.6.0
Time: 41ms
    Asset      Size  Chunks             Chunk Names
bundle.js  4.1 kB        0  [emitted]  main
  [0] ./src/main.js 353 bytes {0} [built]
  [1] ./src/js/mathUtils.js 163 bytes {0} [built]
  [2] ./src/js/info.js 100 bytes {0} [built]
  [3] ./src/css/normal.css 304 bytes {0} [built] [failed] [1 error]

ERROR in ./src/css/normal.css
Module parse failed: E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader\src\css\normal.css Unexpected token (1:5)
You may need an appropriate loader to handle this file type.
body {
  background-color: red;
  /*background: url("../img/timg.jpg");*/
@ ./src/main.js 16:0-27
npm ERR! code ELIFECYCLE
```

加载*.css文件和格式必须有对应的loader

文档目录: <https://webpack.docschina.org/loaders/#styling>

Loaders | webpack 中文文档

webpack.docschina.org/loaders/#styling

中文文档 参与贡献 投票 博客 印记中文 API 概念 配置 指南 LOADER

Loaders

- 文件
- JSON
- 语法转换
- 模板
- 样式**
- Linting 和测试
- 框架
- Awesome

样式

- `style-loader` 将模块导出的内容作为样式并添加到 DOM 中
- `css-loader` 加载 CSS 文件并解析 import 的 CSS 文件，最终返回 CSS 代码
- `less-loader` 加载并编译 LESS 文件
- `sass-loader` 加载并编译 SASS/SCSS 文件
- `postcss-loader` 使用 PostCSS 加载并转换 CSS/SSS 文件
- `stylus-loader` 加载并编译 Stylus 文件

安装CSS-loader

```
1 | cnpm install --save-dev css-loader@3.6.0
```

安装style-loader

```
1 | cnpm install --save-dev style-loader@1.3.0
```

安装以后可以在 `package.json` 文件中查看到版本号。

```
{  
  "name": "guardwhy",  
  "version": "1.0.0",  
  "main": "index.js",  
  ▶ 调试  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1",  
    "build": "webpack"  
  },  
  "author": "guardwhy",  
  "license": "ISC",  
  "devDependencies": {  
    "css-loader": "^4.2.0",  
    "style-loader": "^1.3.0",  
    "webpack": "^3.6.0"  
  },  
  "dependencies": {},  
  "description": ""  
}
```

配置webpack.config.js

```
1 // 1.获得路径  
2 const path = require('path');  
3  
4 // 支持导出  
5 module.exports = {  
6   entry: './src/main.js',  
7   output: {  
8     path: path.resolve(__dirname, 'dist'),  
9     filename: 'bundle.js',  
10    publicPath: 'dist/'  
11  },  
12  module: {  
13    rules: [  
14      { test: /\.css$/,  
15        /*  
16         css-loader只负责将css文件进行加载  
17         style-loader负责将样式添加到DOM中  
18         使用多个loader时，是从右向左  
19         */  
20        use: ['style-loader', 'css-loader']  
21      }  
22    ]  
23  }  
24}
```

执行**npm run build**命令，可能会出现以下的问题。

```
linux@Guardwhy MINGW64 /e/workspace/Web/06-Vue.js/04-webpack/3-webpack的loader
$ npm run build

> guardwhy@1.0.0 build E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader
> webpack

(node:11380) UnhandledPromiseRejectionWarning: TypeError: this.getResolve is not a function
    at Object.loader (E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader\node_modules\_css-loader@5.2.4@css-loader\dist\index.js:62:27)
    at LOADER_EXECUTION (E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader\node_modules\_loader-runner@2.4.0@loader-runner\lib\LoaderRunner.js:119:14)
    at runSyncOrAsync (E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader\node_modules\_loader-runner@2.4.0@loader-runner\lib\LoaderRunner.js:120:4)
    at iterateNormalLoaders (E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader\node_modules\_loader-runner@2.4.0@loader-runner\lib\LoaderRunner.js:232:2)
    at Array.<anonymous> (E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader\node_modules\_loader-runner@2.4.0@loader-runner\lib\LoaderRunner.js:205:4)
    at Storage.finished (E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader\node_modules\_enhanced-resolve@3.4.1@enhanced-resolve\lib\CachedInputFilesystem.js:4:9)
    at E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader\node_modules\_enhanced-resolve@3.4.1@enhanced-resolve\lib\CachedInputFilesystem.js:77:9
    at E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader\node_modules\_graceful-fs@4.2.6@graceful-fs\graceful-fs.js:123:16
    at FSReqWrap.readfileAfterClose [as oncomplete] (internal/fs/read_file_context.js:53:3)
(node:11380) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing inside of an async function without a catch block or was not handled with .catch(). (rejection id: 1)
(node:11380) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In the future, promise rejections that are not handled will terminate the Node.js code.
```

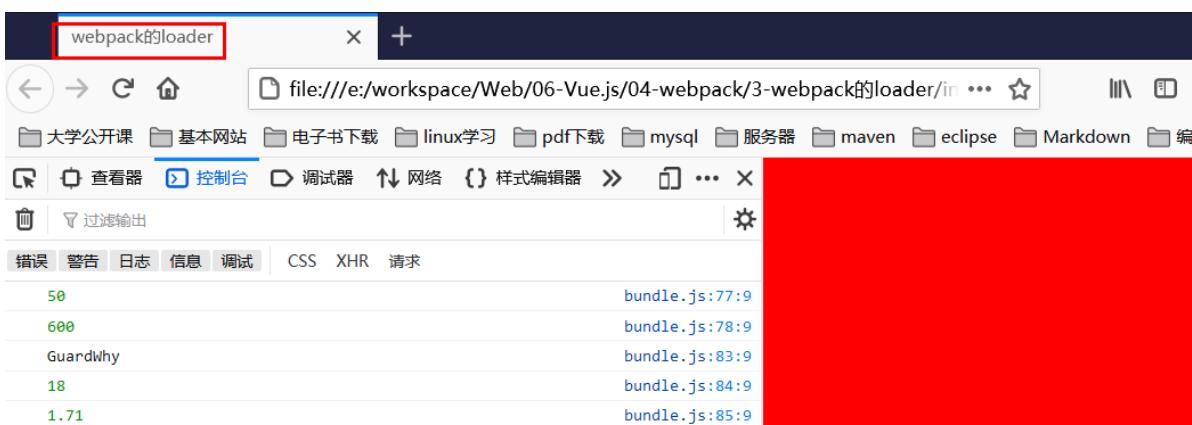
通过查看错误提示，发现会不会有可能是css-loader的版本太高了，所以把css-loader的版本由4.2.0改为了3.6.0！！！

```
linux@Guardwhy MINGW64 /e/workspace/Web/06-Vue.js/04-webpack/3-webpack的loader
$ npm run build

> guardwhy@1.0.0 build E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader
> webpack

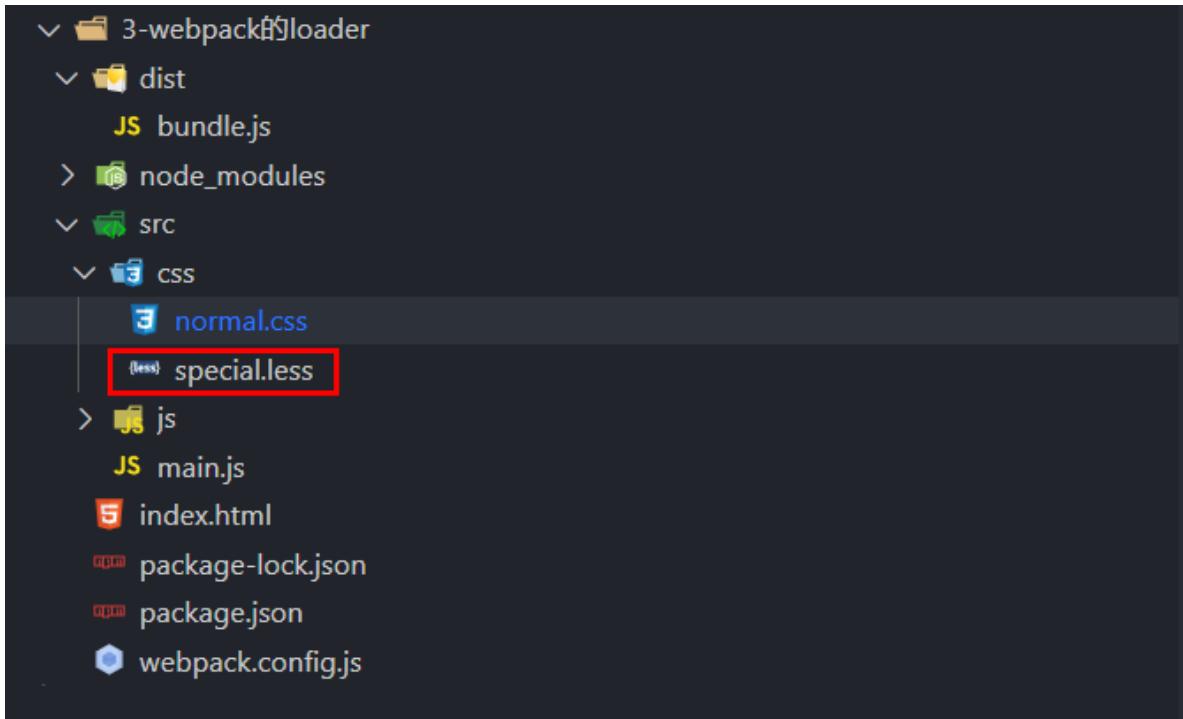
Hash: d29e63990b53247873df
Version: webpack 3.6.0
Time: 253ms
    Asset      Size  Chunks             Chunk Names
bundle.js  14 kB          0  [emitted]  main
[0] ./src/main.js 353 bytes {0} [built]
[1] ./src/js/mathUtils.js 163 bytes {0} [built]
[2] ./src/js/info.js 100 bytes {0} [built]
[3] ./src/css/normal.css 564 bytes {0} [built]
[5] ./node_modules/_css-loader@3.6.0@css-loader/dist/cjs.js!./src/css/normal.css 294 bytes {0} [built]
+ 2 hidden modules
```

执行结果



12.4.2 less文件处理

项目目录



在入口文件中引用

main.js

```
1 // 1. 使用commonjs的模块化导入
2 const {add, mul} = require('./js/mathutils')
3
4 // 输出结果
5 console.log(add(20, 30));
6 console.log(mul(20, 30));
7
8 // 2. 使用es6的模块化规范
9 import {name, age, height} from "./js/info";
10
11 console.log(name);
12 console.log(age);
13 console.log(height);
14
15 // 依赖css文件
16 require('./css/normal.css')
17
18 // 依赖less文件
19 require('./css/special.less')
20 document.write('<h3>kobe是mvp!!!</h3>')
```

加载*.less文件和格式必须有对应的loader

文档目录: <https://webpack.docschina.org/loaders/#styling>

安装less-loader

```
1 | cnpm install less-loader@5.0.0 --save-dev
```

安装less

```
1 | cnpm install less@3.13.1--save-dev
```

安装以后可以在package.json文件中查看到版本号。

```
{  
  "name": "guardwhy",  
  "version": "1.0.0",  
  "main": "index.js",  
  ▶ 调试  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1",  
    "build": "webpack"  
  },  
  "author": "guardwhy",  
  "license": "ISC",  
  "devDependencies": {  
    "css-loader": "^3.6.0",  
    "less": "^3.13.1",  
    "less-loader": "^5.0.0",  
    "style-loader": "^1.3.0",  
    "webpack": "^3.6.0"  
  },  
  "dependencies": {},  
  "description": ""  
}
```

配置webpack.config.js

```
1 // 1. 获得路径  
2 const path = require('path');  
3  
4 // 支持导出  
5 module.exports = {  
6   entry: './src/main.js',  
7   output: {  
8     path: path.resolve(__dirname, 'dist'),  
9     filename: 'bundle.js'  
10  },  
11  module: {  
12    rules: [  
13      { test: /\.css$/,  
14        /*  
15         css-loader只负责将css文件进行加载  
16         style-loader负责将样式添加到DOM中  
17         使用多个loader时，是从右向左  
18         */  
19        use: ['style-loader', 'css-loader']  
20      },  
21      {  
22        test: /\.less$/,  
23        use: [{  
24          loader: "style-loader" // creates style nodes from JS strings
```

```
25      }, {
26        loader: "css-loader" // translates CSS into CommonJS
27      }, {
28        loader: "less-loader" // compiles Less to CSS
29      }]
30    }
31  ]
32 }
33 }
```

执行npm run build命令。

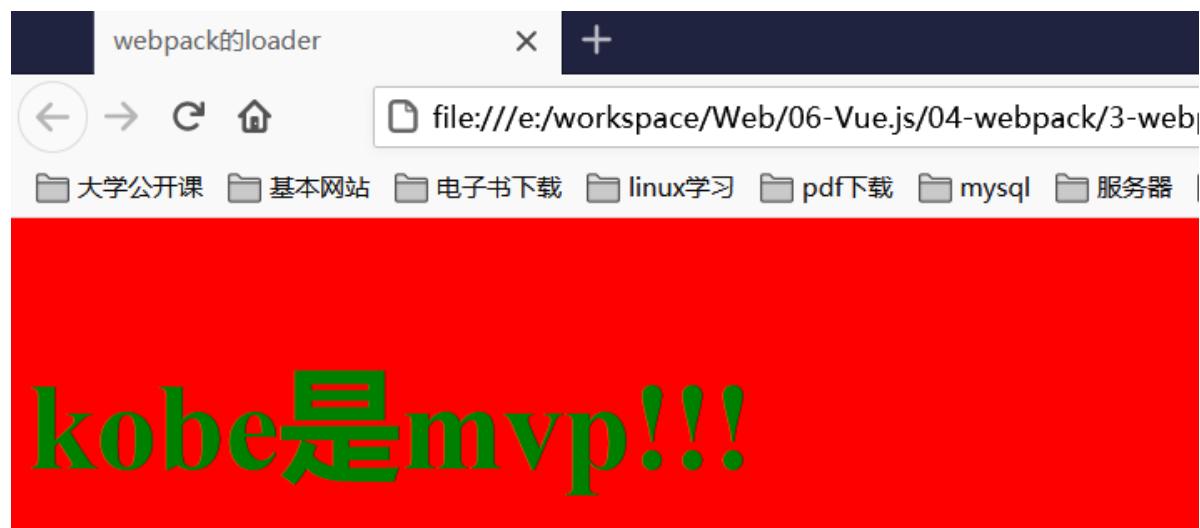
```
linux@Guardwhy MINGW64 /e/workspace/Web/06-Vue.js/04-webpack/3-webpack的loader
$ npm run build

> guardwhy@1.0.0 build E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader
> webpack

Hash: 58427050f731c2eff737
Version: webpack 3.6.0
Time: 327ms
 Asset      Size  Chunks             Chunk Names
bundle.js  14.9 kB          {0} [emitted]  main
[2] ./src/main.js 447 bytes {0} [built]
[3] ./src/js/mathUtils.js 163 bytes {0} [built]
[4] ./src/js/info.js 100 bytes {0} [built]
[5] ./src/css/normal.css 564 bytes {0} [built]
[6] ./node_modules/_css-loader@3.6.0@css-loader/dist/cjs.js!./src/css/normal.css 294 bytes {0} [built]
[7] ./src/css/special.less 628 bytes {0} [built]
[8] ./node_modules/_css-loader@3.6.0@css-loader/dist/cjs.js!./node_modules/_less-loader@5.0.0@less-loader/
+ 2 hidden modules

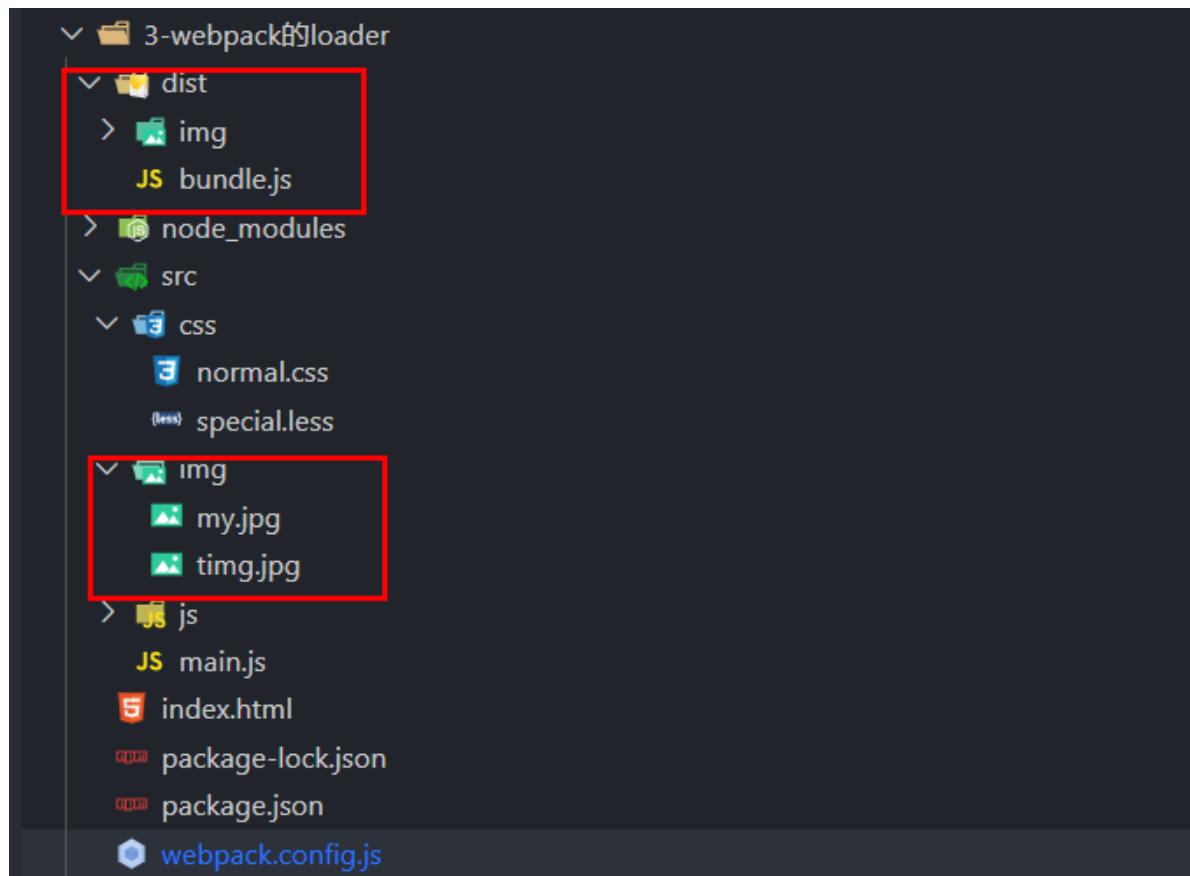
linux@Guardwhy MINGW64 /e/workspace/Web/06-Vue.js/04-webpack/3-webpack的loader
```

执行结果



12.4.3 图片文件处理

项目目录



normal.css中的样式

```
1 body {  
2     background: url("../img/timg.jpg");  
3 }
```

加载图片文件和格式必须有对应的loader

加载图片文件和格式必须有对应的loader

Loaders | webpack 中文文档 +

webpack.docschina.org/loaders/#files

中文文档 参与贡献 投票 博客 印记中文 API 概念 配置 指南 LOADER 迁移 PLUGIN

Loaders

- 文件
- JSON
- 语法转换
- 模板
- 样式
- Linting 和测试
- 框架
- Awesome

文件

- raw-loader 用于加载文件的原始内容 (utf-8)
- val-loader 将代码作为模块执行，并将其导出为 JS 代码
- url-loader 与 file-loader 类似，但是如果文件大小小于一个设置的值，则会返回 data URL
- file-loader 将文件保存至输出文件夹中并返回 (相对) URL
- ref-loader 用于手动建立文件之间的依赖关系

文档目录: <https://webpack.docschina.org/loaders/#files>

安装url-loader

```
1 | cnpm install url-loader@2.0.0 --save-dev
```

安装file-loader

```
1 | cnpm install file-loader@2.0.0 --save-dev
```

安装以后可以在package.json文件中查看到版本号。

```
"devDependencies": {
    "css-loader": "^3.6.0",
    "file-loader": "^6.0.0",
    "less": "^3.13.1",
    "less-loader": "^5.0.0",
    "style-loader": "^1.3.0",
    "url-loader": "^4.0.0",
    "webpack": "^3.6.0"
},
"dependencies": {},
"description": ""
}
```

配置webpack.config.js

```
1 // 1. 获得路径
2 const path = require('path');
3
4 // 支持导出
5 module.exports = {
6   entry: './src/main.js',
7   output: {
8     path: path.resolve(__dirname, 'dist'),
9     filename: 'bundle.js',
10    publicPath: 'dist/'
11  },
12  module: {
13    rules: [
14      { test: /\.css$/,
15        /*
16         css-loader只负责将css文件进行加载
17         style-loader负责将样式添加到DOM中
18         使用多个loader时，是从右向左
19         */
20        use: ['style-loader', 'css-loader']
21      },
22      {
23        test: /\.less$/,
24        use: [
25          {
26            loader: "style-loader" // creates style nodes from JS strings
27          },
28          {
29            loader: "css-loader" // translates CSS into CommonJS
30          },
31          {
32            loader: "less-loader" // compiles Less to CSS
33          }
34        ],
35        options: {
36          loader: 'url-loader',
37          options: {
38            // 当加载的图片，小于limit时会将图片编译成base64字符串形式
39            //当加载的图片，大于limit时需要使用file-loader模块进行加载
40          }
41        }
42      }
43    ]
44  }
45}
```

```

40         limit: 13000,
41         name: 'img/[name].[hash:8].[ext]'
42     }
43 }
44 ]
45 }
46 ]
47 }
48 }

```

执行npm run build命令。

```

linux@Guardwhy MINGW64 /e/workspace/Web/06-Vue.js/04-webpack/3-webpack的loader
$ npm run build

> guardwhy@1.0.0 build E:\workspace\Web\06-Vue.js\04-webpack\3-webpack的loader
> webpack

Hash: 58427050f731c2eff737
Version: webpack 3.6.0
Time: 327ms
 Asset      Size  Chunks             Chunk Names
bundle.js  14.9 kB       0  [emitted]  main
[2] ./src/main.js 447 bytes {0} [built]
[3] ./src/js/mathUtils.js 163 bytes {0} [built]
[4] ./src/js/info.js 100 bytes {0} [built]
[5] ./src/css/normal.css 564 bytes {0} [built]
[6] ./node_modules/_css-loader@3.6.0@css-loader/dist/cjs.js!./src/css/normal.css 294 bytes {0} [built]
[7] ./src/css/special.less 628 bytes {0} [built]
[8] ./node_modules/_css-loader@3.6.0@css-loader/dist/cjs.js!./node_modules/_less-loader@5.0.0@less-loader/
+ 2 hidden modules

linux@Guardwhy MINGW64 /e/workspace/Web/06-Vue.js/04-webpack/3-webpack的loader

```

执行结果



12.4.4 ES6语法处理

如果希望将ES6的语法转成ES5，那么就需要使用babel，而在webpack中，直接使用babel对应的loader就可以了。

安装babel-loader

```
1 | cnpm install --save-dev babel-loader@7 babel-core babel-preset-es2015
```

安装以后可以在package.json文件中查看到版本号。

```
"devDependencies": {
  "babel-core": "^6.26.3",
  "babel-loader": "^7.1.5",
  "babel-preset-es2015": "^6.24.1",
  "css-loader": "^3.6.0",
  "file-loader": "^6.0.0",
  "less": "^3.13.1",
  "less-loader": "^5.0.0",
  "style-loader": "^1.3.0",
  "url-loader": "^4.0.0",
  "webpack": "^3.6.0"
},
"dependencies": {}},
```

配置webpack.config.js

```
1 // 1. 获得路径
2 const path = require('path');
3
4 // 支持导出
5 module.exports = {
6   entry: './src/main.js',
7   output: {
8     path: path.resolve(__dirname, 'dist'),
9     filename: 'bundle.js',
10    publicPath: 'dist/'
11  },
12  module: {
13    rules: [
14      { test: /\.css$/,
15        /*
16         css-loader只负责将css文件进行加载
17         style-loader负责将样式添加到DOM中
18         使用多个loader时，是从右向左
19         */
20        use: ['style-loader', 'css-loader']
21      },
22      {
23        test: /\.less$/,
24        use: [
25          {
26            loader: "style-loader" // creates style nodes from JS strings
27          },
28          {
29            loader: "css-loader" // translates CSS into CommonJS
30          },
31          {
32            loader: "less-loader" // compiles Less to CSS
33          }
34        ]
35      },
36    ]
37  }
38}
```

```
33     test: /\.png|jpg|gif$/,
34     use: [
35         {
36             loader: 'url-loader',
37             options: {
38                 limit: 13000,
39                 name: 'img/[name].[hash:8].[ext]'
40             }
41         }
42     ],
43 },
44 {
45     test: /\.js$/,
46     exclude: /(node_modules|bower_components)/,
47     use: [
48         {
49             loader: 'babel-loader',
50             options: {
51                 presets: ['es2015']
52             }
53         }
54     ]
55 }
56 }
```

重新打包，查看bundle.js文件，发现其中的内容变成了ES5的语法。

12.5 webpack配置Vue

12.5.1 引入vue.js

安装Vue，在实际项目中也会使用vue的，所以并不是开发时依赖。

```
1 | cnpm install vue@2.5.11 --save
```

main.js

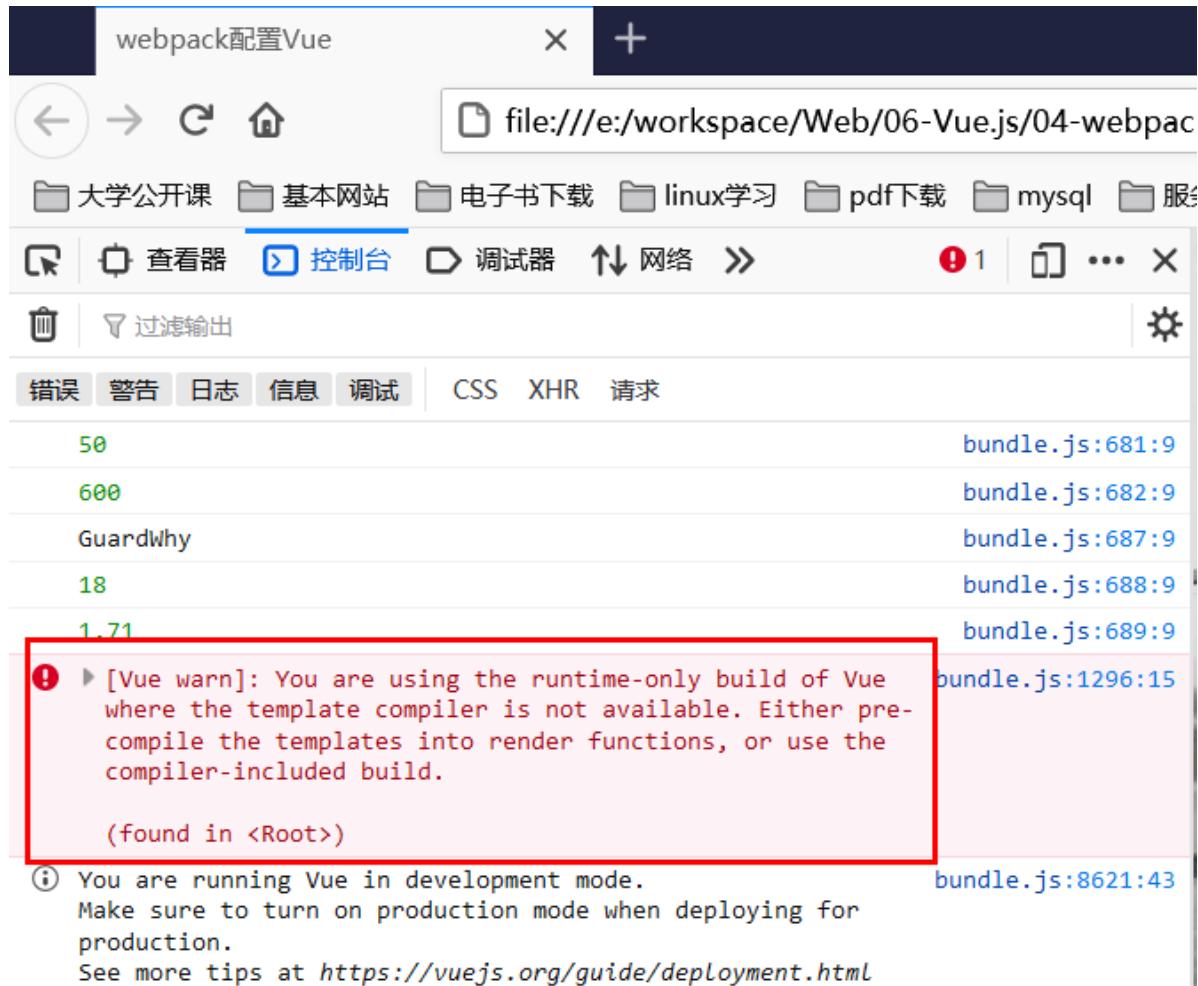
```
1 import Vue from 'vue'
2
3 new Vue({
4     el: '#app',
5     data: {
6         message: 'hello vue.js!!!'
7     }
8 })
```

index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>webpack配置Vue</title>
6 </head>
7 <body>
```

```
9 <div id = "app">
10   {{message}}
11 </div>
12 <!--引入js文件-->
13 <script src="dist/bundle.js"></script>
14 </body>
15 </html>
```

执行npm run build 命令结果，报错！！！



解决方案修改webpack的配置

webpack.config.js

```
1 // 以下内容
2 resolve: {
3   // alias: 别名
4   extensions: ['.js', '.css', '.vue'],
5   alias: {
6     'vue$': 'vue/dist/vue.esm.js'
7   }
8 }
```

执行结果



12.5.2 el和template区别

正常运行之后，来考虑另外一个问题

如果希望将data中的数据显示在界面中，就必须是修改index.html
但是html模板在之后的开发中，并不希望手动的来频繁修改。

定义template属性

在前面的Vue实例中，我们定义了el属性，用于和index.html中的#app进行绑定，让Vue实例之后可以管理它其中的内容。

这里，可以将div元素中的{{message}}内容删掉，只保留一个基本的id为div的元素。

el和template模板的关系是什么呢？

知道el用于指定Vue要管理的DOM，可以帮助解析其中的指令、事件监听等。

而如果Vue实例中同时指定了template，那么template模板的内容会替换掉挂载的对应el的模板。

这样做之后就不需要在以后的开发中再次操作index.html，只需要在template中写入对应的标签即可。

代码示例

main.js

```

1 import vue from 'vue'
2
3 new vue({
4   el: '#app',
5   template:
6     <div>
7       <h3>{{message}}</h3>
8       <button @click= "btnClick">按钮</button>
9       <h3>{{name}}</h3>
10    </div>
11    ,
12    data:{
13      message:'hello vue.js!!!!',
14      name: 'guardwhy'
15    },
16    methods:{
17      btnClick(){
18        }
19      }
20    }
21  })

```

index.html

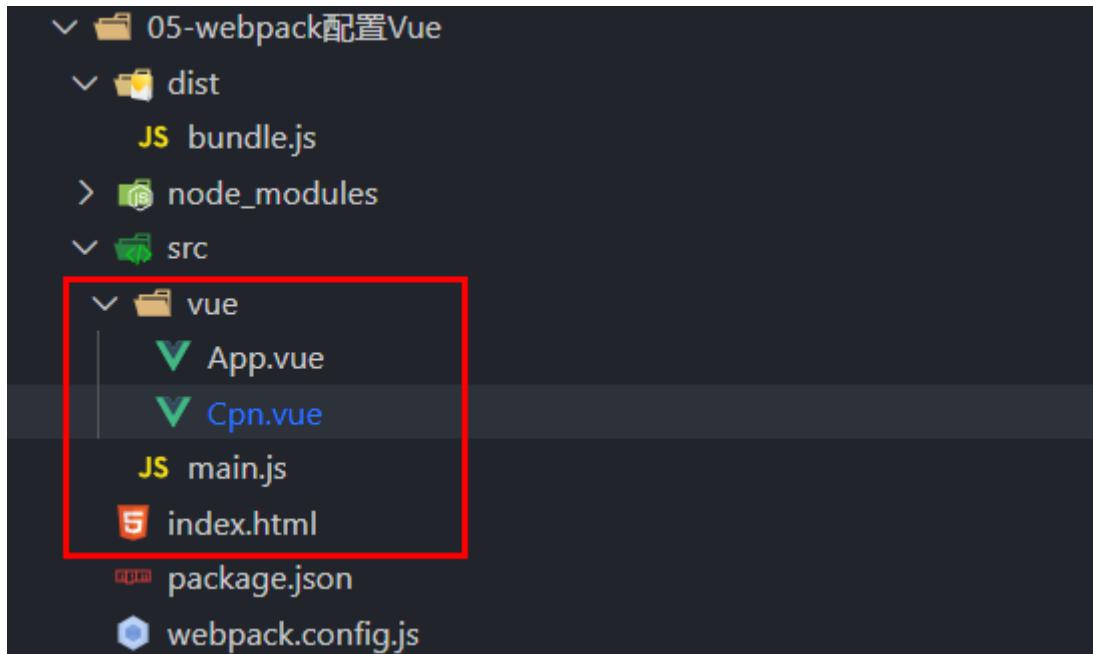
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>webpack配置Vue</title>
6 </head>
7 <body>
8
9 <div id = "app">
10</div>
11<!--引入js文件-->
12<script src="dist/bundle.js"></script>
13</body>
14</html>
```

执行结果



12.5.3 Vue组件化开发

项目目录



安装vue-loader和vue-template-compiler

```
1 | cnpm install vue-loader vue-template-compiler --save-dev
```

安装以后可以在package.json文件中查看到版本号。

```
12 "devDependencies": {
13     "babel-core": "^6.26.3",
14     "babel-loader": "^7.1.5",
15     "babel-preset-es2015": "^6.24.1",
16     "vue-loader": "^13.0.0",
17     "vue-template-compiler": "^2.6.12",
18     "webpack": "^3.6.0"
19 },
20 "dependencies": {
21     "vue": "^2.6.12"
22 }
```

修改webpack.config.js的配置文件

```
1 // 1. 获得路径
2 const path = require('path');
3 // 引入webpack
4 const webpack = require('webpack');
5
6 // 支持导出
7 module.exports = {
8     entry: './src/main.js',
9     output: {
10         path: path.resolve(__dirname, 'dist'),
11         filename: 'bundle.js',
12         publicPath: 'dist/'
13     },
14 }
```

```
14 module: {
15   rules: [
16     {
17       test: /\.js$/,
18       exclude: /(node_modules|bower_components)/,
19       use: [
20         {
21           loader: 'babel-loader',
22           options: {
23             presets: ['es2015']
24           }
25         },
26         {
27           test: /\.vue$/,
28           use: ['vue-loader']
29         }
30       ]
31     },
32   ],
33   resolve: {
34     // alias: 别名
35     extensions: ['.js', '.css', '.vue'],
36     alias: {
37       'vue$': 'vue/dist/vue.esm.js'
38     }
39   }
40 }
41 }
```

代码示例

main.js

```
1 import Vue from 'vue'
2 import App from "./vue/App.vue"
3
4 new Vue({
5   el: '#app',
6   template: '<App/>',
7   components: {
8     App
9   }
10 })
```

App.vue

```
1 <template>
2   <div>
3     <h3 class="title">{{message}}</h3>
4     <button @click="btnClick">按钮</button>
5     <h3>{{name}}</h3>
6     <Cpn></Cpn>
7   </div>
8 </template>
9
10 <script>
```

```
1 // 引入组件
2 import Cpn from './Cpn'
3 export default {
4   name: "App",
5   // 注册Cpn组件
6   components: {
7     Cpn
8   },
9   data(){
10   return {
11     message: 'hello webpack!!!!',
12     name: 'guardwhy'
13   }
14 },
15 methods: {
16   btnClick(){
17   }
18 }
19 }
20 </script>
21
22 <style scoped>
23   .title {
24     color: green;
25   }
26 </style>
```

Cpn.vue

```
1 <template>
2   <div>
3     <h3>我是NBA球星</h3>
4     <p>我是三分王，我是Curry</p>
5     <h3>{{name}}</h3>
6   </div>
7 </template>
8
9 <script>
10 export default {
11   name: "Cpn",
12   data(){
13     return{
14       name: 'hello vue.js!!!'
15     }
16   }
17 }
18 </script>
19
20 <style scoped>
21
22 </style>
```

index.html

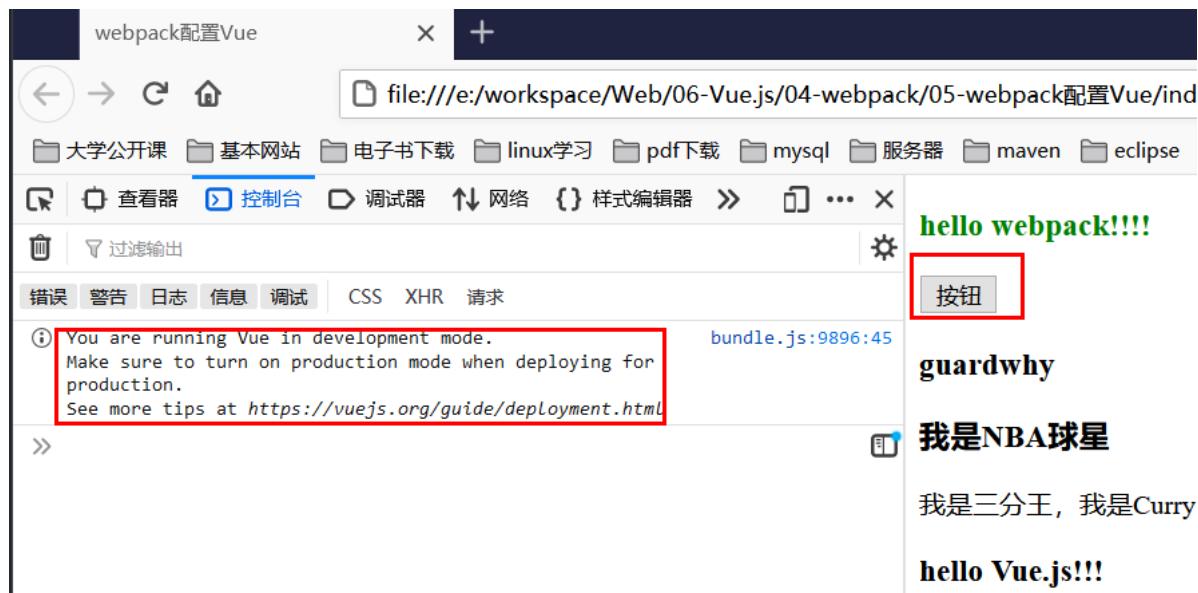
```
1 <!DOCTYPE html>
```

```

2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>webpack配置Vue</title>
6   </head>
7   <body>
8
9   <div id = "app">
10  </div>
11  <!--引入js文件-->
12  <script src="dist/bundle.js"></script>
13  </body>
14 </html>

```

执行命令 `npm run build`，执行成功！！！



12.6 认识plugin

plugin是什么？

- plugin是插件的意思，通常是用于对某个现有的架构进行扩展。
- webpack中的插件，就是对webpack现有功能的各种扩展，比如打包优化，文件压缩等等。

loader和plugin区别

- loader主要用于转换某些类型的模块，它是一个转换器。
- plugin是插件，它是对webpack本身的扩展，是一个扩展器。

plugin的使用过程：

- 步骤一：通过npm安装需要使用的plugins(某些webpack已经内置的插件不需要安装)。
- 步骤二：在webpack.config.js中的plugins中配置插件。

12.6.1 打包html的plugin

HtmlWebpackPlugin插件具体作用

自动生成一个index.html文件(可以指定模板来生成)。
将打包的js文件，自动通过script标签插入到body中。

安装HtmlWebpackPlugin插件

```
1 | cnpm install html-webpack-plugin --save-dev
```

安装以后可以在package.json文件中查看到版本号。

```
"devDependencies": {
    "babel-core": "^6.26.3",
    "babel-loader": "^7.1.5",
    "babel-preset-es2015": "^6.24.1",
    "html-webpack-plugin": "^5.3.1", // 盒子
    "vue-loader": "^13.0.0",
    "vue-template-compiler": "^2.6.12",
    "webpack": "^3.6.0"
},
"dependencies": {
    "vue": "^2.6.12"
}
```

修改webpack.config.js文件

```
1 // 引入插件
2 const HtmlWebpackPlugin = require('html-webpack-plugin');
3
4 // 支持导出
5 module.exports = {
6   entry: './src/main.js',
7   output: {
8     path: path.resolve(__dirname, 'dist'),
9     filename: 'bundle.js',
10    // publicPath: 'dist/'
11  },
12  plugins: [
13    // 添加插件
14    new HtmlWebpackPlugin({
15      template: 'index.html'
16    }),
17  ]
18 }
```

index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>webpack配置Vue</title>
6 </head>
7 <body>
8
9 <div id = "app">
10 </div>
11 </body>
12 </html>
```

使用命令 npm run build 执行操作，发现报错！！！

```
linux@Guardwhy MINGW64 /e/workspace/Web/06-Vue.js/04-webpack/05-webpack配置Vue
$ npm run build

> guardwhy@1.0.0 build E:\workspace\Web\06-Vue.js\04-webpack\05-webpack配置Vue
> webpack

E:\workspace\Web\06-Vue.js\04-webpack\05-webpack配置Vue\node_modules\_webpack@3.6.0@webpack\bin\webpack.js:14
      throw e;
      ^

TypeError: Cannot read property 'initialize' of undefined
    at HtmlWebpackPlugin.apply (E:\workspace\Web\06-Vue.js\04-webpack\05-webpack配置Vue\node_modules\_webpack@3.6.0@webpack\lib\HTMLWebpackPlugin.js:14:10)
    at Compiler.apply (E:\workspace\Web\06-Vue.js\04-webpack\05-webpack配置Vue\node_modules\_tapable@1.0.3\Compiler.js:57:10)
    at webpack (E:\workspace\Web\06-Vue.js\04-webpack\05-webpack配置Vue\node_modules\_webpack@3.6.0@webpack\bin\webpack.js:14:10)
    at processOptions (E:\workspace\Web\06-Vue.js\04-webpack\05-webpack配置Vue\node_modules\_webpack@3.6.0@webpack\lib\webpack.js:14:10)
    at yargs.parse (E:\workspace\Web\06-Vue.js\04-webpack\05-webpack配置Vue\node_modules\_yargs@13.2.0\yargs.js:14:10)
    at Object.Yargs.self.parse (E:\workspace\Web\06-Vue.js\04-webpack\05-webpack配置Vue\node_modules\_yargs@13.2.0\yargs.js:14:10)
    at Object.<anonymous> (E:\workspace\Web\06-Vue.js\04-webpack\05-webpack配置Vue\node_modules\_yargs@13.2.0\yargs.js:14:10)
    at Module._compile (internal/modules/cjs/loader.js:701:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:712:10)
    at Module.load (internal/modules/cjs/loader.js:600:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:539:12)
    at Function.Module._load (internal/modules/cjs/loader.js:531:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:754:12)
    at startup (internal/bootstrap/node.js:283:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:622:3)

npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! guardwhy@1.0.0 build: `webpack`
npm ERR! Exit status 1
```

解决方案！！！

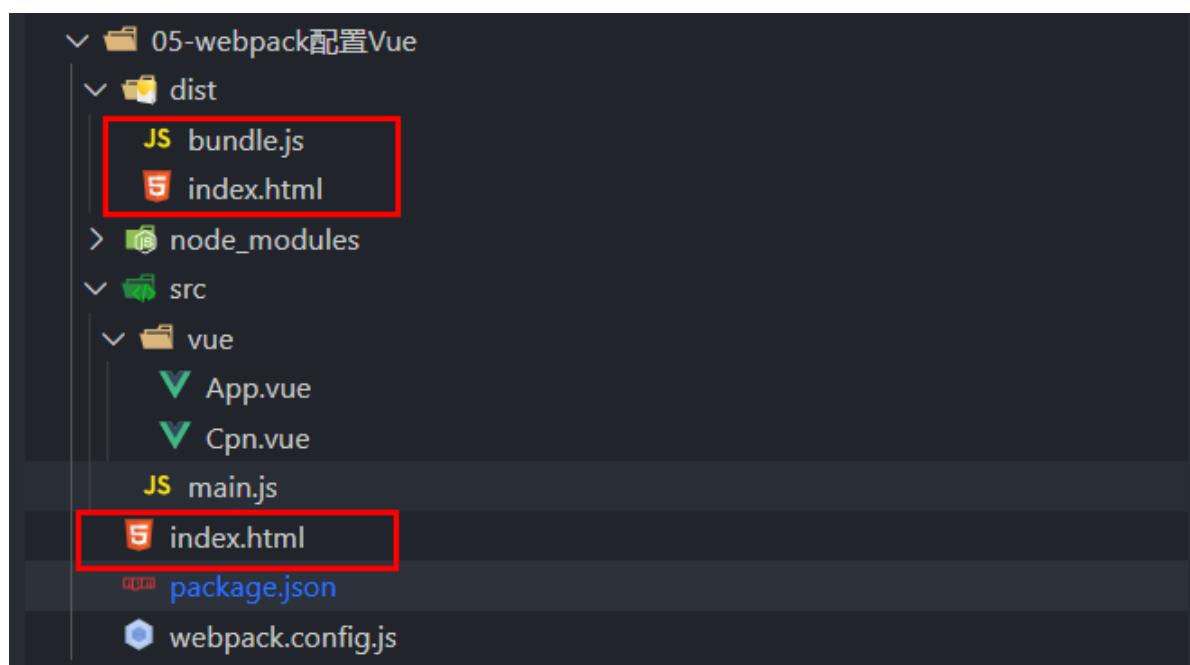
HtmlWebpackPlugin插件版本太高了，将HtmlWebpackPlugin版本设置为2.0.0

```
"devDependencies": {
    "babel-core": "^6.26.3",
    "babel-loader": "^7.1.5",
    "babel-preset-es2015": "^6.24.1",
    "html-webpack-plugin": "^2.0.0", highlight
    "vue-loader": "^13.0.0",
    "vue-template-compiler": "^2.6.12",
    "webpack": "^3.6.0"
},
"dependencies": {
    "vue": "^2.6.12"
}
```

终端执行命令

```
1 | cnpm install
```

操作结果



12.6.2 js压缩的Plugin

安装uglifyjs-webpack-plugin插件

```
1 | cnpm install uglifyjs-webpack-plugin@1.1.1 --save-dev
```

修改webpack.config.js文件

```
1 // 添加压缩js插件
2 const UglifyjsWebpackPlugin = require('uglifyjs-webpack-plugin');
3
4 // 支持导出
```

```
5 module.exports = {
6   entry: './src/main.js',
7   output: {
8     path: path.resolve(__dirname, 'dist'),
9     filename: 'bundle.js',
10    // publicPath: 'dist/'
11  },
12  plugins: [
13    new UglifyjsWebpackPlugin()
14  ]
15 }
```

13- Vue CLI详解

13.1 什么是Vue CLI

CLI是Command-Line Interface, 翻译为命令行界面, 但是俗称脚手架。

Vue CLI是一个官方发布 vue.js 项目脚手架, 使用 vue-cli 可以快速搭建Vue开发环境以及对应的 webpack 配置。

13.2 Vue CLI使用前提条件

13.2.1 安装NodeJS

可以直接在官方网站中下载安装, 网址: <http://nodejs.cn/download/>

默认情况下自动安装Node和NPM, Node环境要求8.9以上或者更高版本。

检测安装的版本

```
1 | node -v
```

13.2.2 什么是NPM呢?

NPM的全称是Node Package Manager, 是一个NodeJS包管理和分发工具, 已经成为了非官方的发布 Node模块(包) 的标准。

注意事件

NPM官方的管理的包都是从 <http://npmjs.com> 下载的, 但是这个网站在国内速度很慢。这里推荐使用淘宝 NPM 镜像 <http://npm.taobao.org/>, 淘宝 NPM 镜像是一个完整npmjs.com 镜像, 同步频率目前为 10分钟一次, 以保证尽量与官方服务同步。

设置镜像和存储地址:

```
1 # 由于npm代码仓库的服务器在国外, 由于Great Firewall的缘故, 这时可以使用淘宝的npm代码仓库, 通过npm安装cnpm
2 npm install -g cnpm --registry=https://registry.npm.taobao.org
3 # 安装成功后, 可以通过以下命令查看cnpm版本:
4 cnpm -v
5 #设置npm下载包时保存在本地的地址(建议英文目录), 通过cnpm来操作下载速度会得到很大提升, 但包的版本不一定是最新的。
6 cnpm config set prefix "E:\\Develop\\repo_npm"
7 #查看cnpm配置信息
8 cnpm config list
```

查看自己的npm版本！！！

```
PS C:\Users\linux> cnpm -v
cnpm@6.2.0 (E:\Develop\repo_npm\node_modules\cnpm\lib\parse_argv.js)
npm@6.14.13 (E:\Develop\repo_npm\node_modules\cnpm\node_modules\npm\lib\npm.js)
node@10.15.3 (C:\Program Files\nodejs\node.exe)
npminstall@3.28.0 (E:\Develop\repo_npm\node_modules\cnpm\node_modules\npminstall\lib\index.js)
prefix=E:\Develop\repo_npm
win32 x64 10.0.19042
registry=https://registry.nlark.com
PS C:\Users\linux> npm -v
6.4.1
PS C:\Users\linux>
```

13.2.3 安装Webpack

Vue.js官方脚手架工具就使用了webpack模板，对所有的资源会压缩等优化操作。它在开发过程中提供了一套完整的功能，能够使得我们开发过程中变得高效。

Webpack的全局安装

```
1 | cnpm install webpack@3.6.0 -g
```

13.3 Vue CLI2的使用

13.3.1 Vue CLI安装

目前主流版本是 2.x 和 3.x 版本(4.0 由于版本过高 很多插件不支持),安装3.x 以上的版本是因为该版本既可以创建2.x项目与3.x 项目。

安装Vue脚手架

```
1 | npm install -g @vue/cli@3.12.1
```

输入 vue -V 查看版本

```
linux@Guardwhy MINGW64 /e/workspace/Web/06-Vue.js/05-Vue CLI
$ vue -V
3.12.1
```

```
linux@Guardwhy MINGW64 /e/workspace/Web/06-Vue.js/05-Vue CLI
$ []
```

拉取 2.x 模板 (旧版本)

Vue CLI >= 3 和旧版使用了相同的 vue 命令，所以 Vue CLI 2 (@vue/cli) 被覆盖了。如果仍然需要使用旧版本的 vue init 功能，可以全局安装一个桥接工具。

```
1 | cnpm install -g @vue/cli-init
2 | # `vue init` 的运行效果将会跟 `vue-cli@2.x` 相同
3 | vue init webpack my-project
```

13.3.2 Vue CLI2详解

使用Vue CLI2创建工程

```
1 | vue init webpack vuecli2_test01
```

1.会根据这个名称创建一个文件夹，存放之后项目的内容
该名称也会作为默认的项目名称，但是不能包含大写字符

2.项目名称

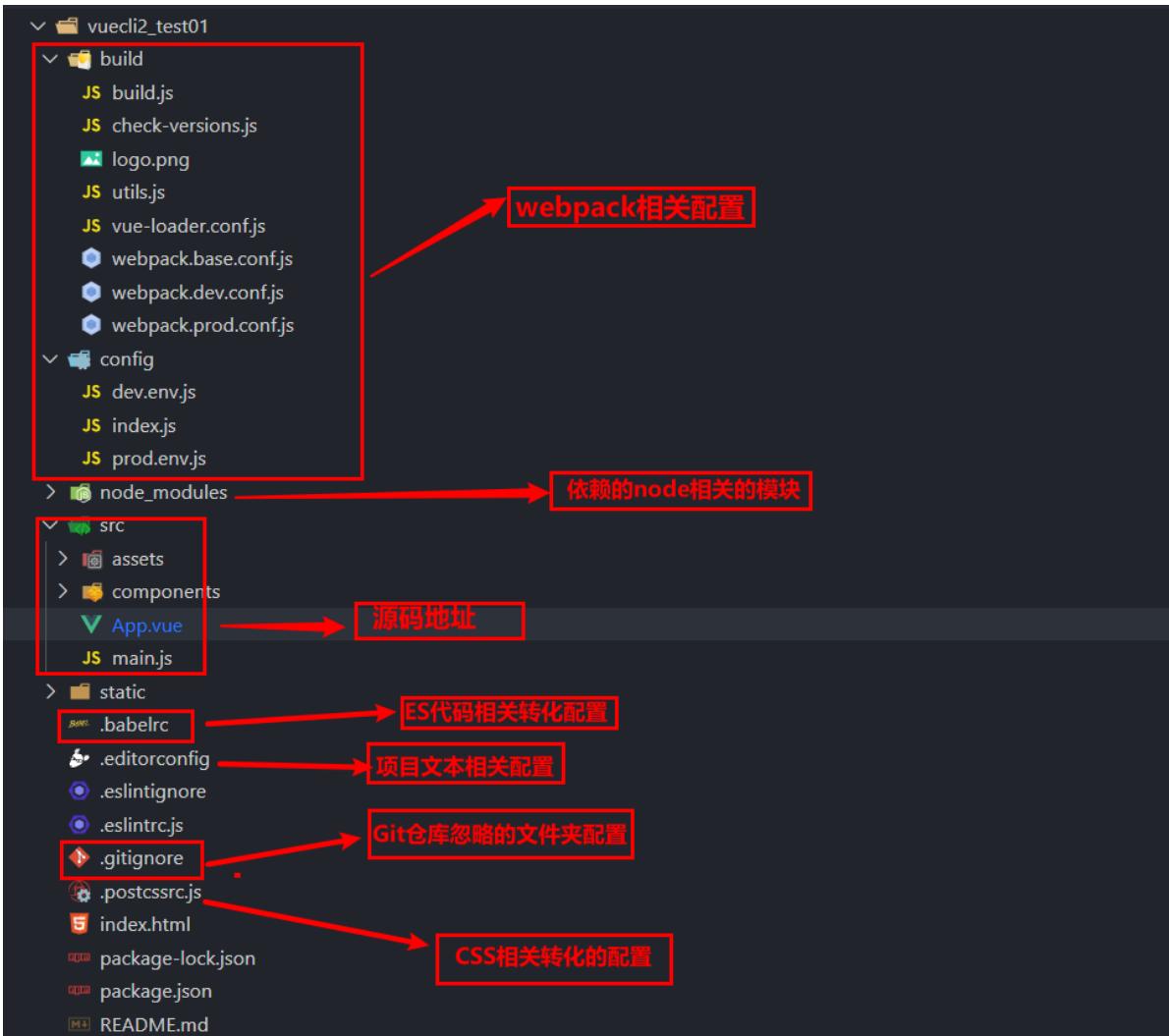
3.作者信息

选择使用npm

```
linux@GuardWhy MINGW64 /e/workspace/Web/06-Vue.js/05-Vue CLI
$ vue init webpack vuecli2_test01
? Project name vuecli2_test01
? Project description test vuecli2
? Author guardwhy <hxy1625309592@foxmail.com>
? Vue build standalone
? Install vue-router? No
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Set up unit tests No
? Setup e2e tests with Nightwatch? No
? Should we run `npm install` for you after the project has been created? (recommended)

vue-cli · Generated "vuecli2_test01".
# Installing project dependencies ...
```

目录结构详解



Runtime-Compiler和Runtime-only的区别

运行时+编译器(Runtime + Compiler)版本 vs 只含有运行时版本(Runtime-only)

如果你需要在客户端编译模板（例如，向 `template` 选项传入一个字符串，或者需要将模板中的非 DOM 的 HTML 挂载到一个元素），你需要带有编译器的版本，因而需要完整构建版本。

```
// 这种情况需要编译器(compiler)
new Vue({
  template: '<div>{{ hi }}</div>'
})

// 这种情况不需要
new Vue({
  render (h) {
    return h('div', this.hi)
  }
})
```

在使用 `vue-loader` 或 `vueify` 时，`*.vue` 文件中的模板会在构建时(build time)预编译(pre-compile)为 JavaScript。最终生成的 bundle 中你不再需要编译器(compiler)，因此可以直接使用只含有运行时的构建版本(runtime-only)。

由于只含有运行时构建版本(runtime-only)比完整构建版本(full-build)轻量大约 30%，你应该尽可能使用只含有运行时的构建版本。如果你还是希望使用完整构建版本，则需要在打包器中配置别名：

由于运行时版本的构建比其全面版本的重量轻约30%，因此你可以随时使用它。如果你仍然希望使用完整版本，则需要在捆绑程序中配置别名：

简单总结

如果在之后的开发中，你依然使用`template`，就需要选择Runtime-Compiler。如果之后的开发中，使用的是`.vue`文件夹开发，那么可以选择Runtime-only。

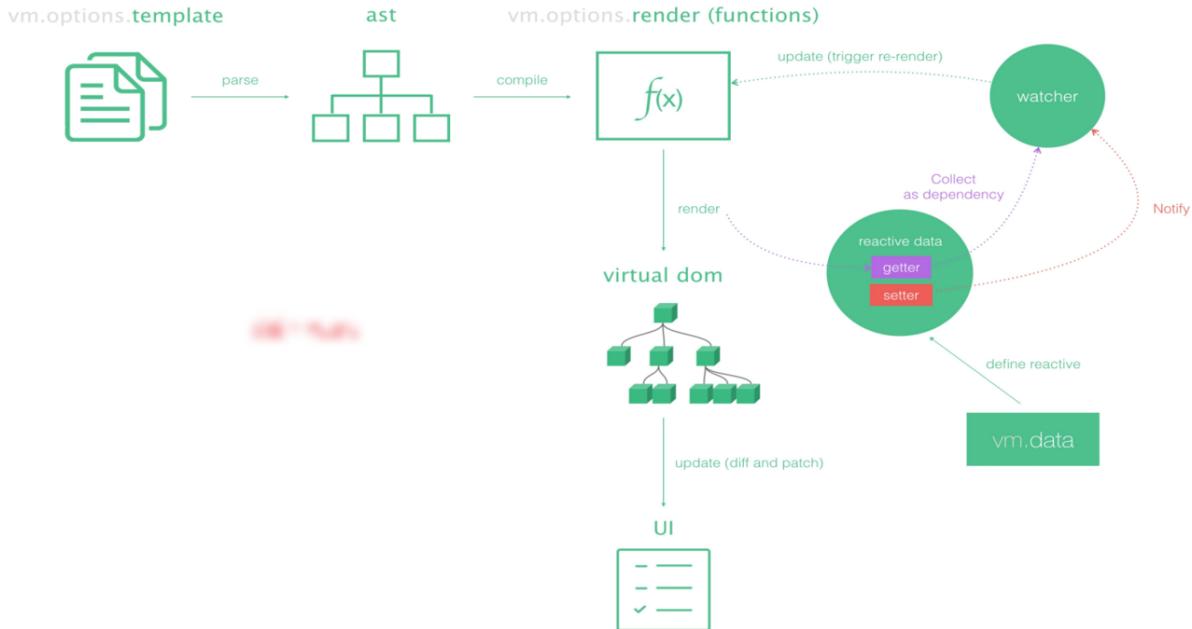
```
JS main.js
runtimecompiler02 > src > JS mainjs > ...
1 import Vue from 'vue'
2 import App from './App'
3
4 Vue.config.productionTip = false
5
6 /* eslint-disable no-new */
7 new Vue({
8   el: '#app',
9   components: { App },
10  template: '<App/>'
11})
12

JS main.js
runtimeonly03 > src > JS mainjs > ...
1 import Vue from 'vue'
2 import App from './App'
3
4 Vue.config.productionTip = false
5
6 /* eslint-disable no-new */
7 new Vue({
8   el: '#app',
9   render: h => h(App)
10})
11
```

Vue程序运行过程

Runtime-Compiler执行过程, `template -> ast ->render ->virtual dom ->UI`

Runtime-only执行过程, `render -> virtual dom ->UI`, Runtime-only代码量更少，性能更加高效！！！



render函数的使用

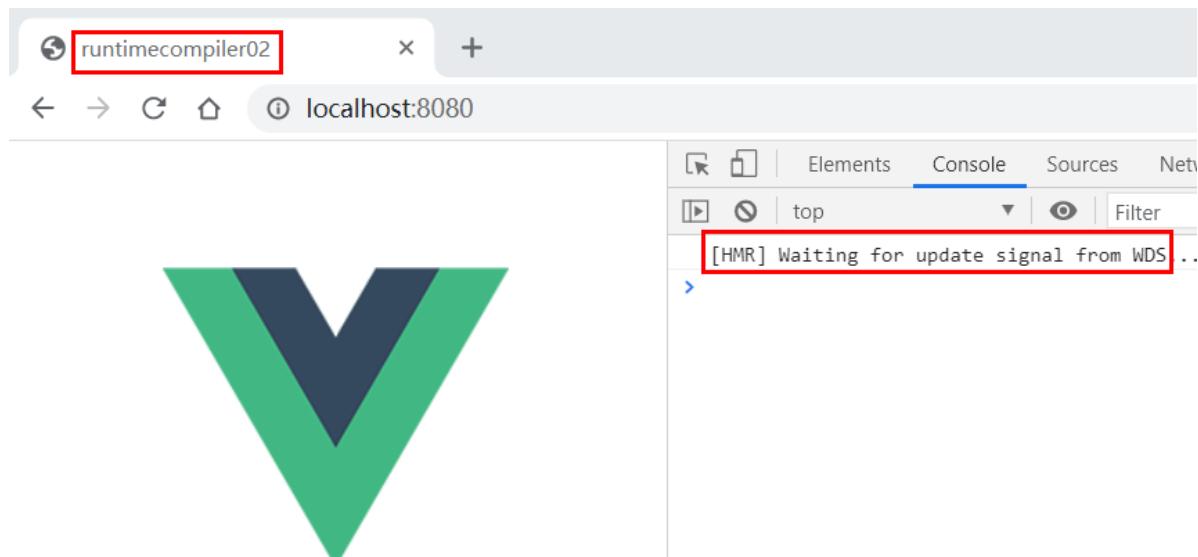
Runtime-Compiler中的入口

```

1 import vue from 'vue'
2 import App from './App'
3
4 // 注册组件
5 const cpn = {
6   template: '<div>{{message}}</div>',
7   data(){
8     return {
9       message:'heTlo VueCLI! ! ! '
10    }
11  }
12}
13Vue.config.productionTip = false
14
15/* eslint-disable no-new */
16new vue({
17  el: '#app',
18  // components: { App },
19  // template: '<App/>'
20  render: function(createElement){
21    //方式 1.createElement('标签', {标签的属性}, ['数组'])
22    // return createElement('h3',{class:'st1'},['hello vue.js!!!!'])
23
24    // 方式2.传入组件对象
25    // return createElement(cpn)
26
27    return createElement(App)
28  }
29
30})

```

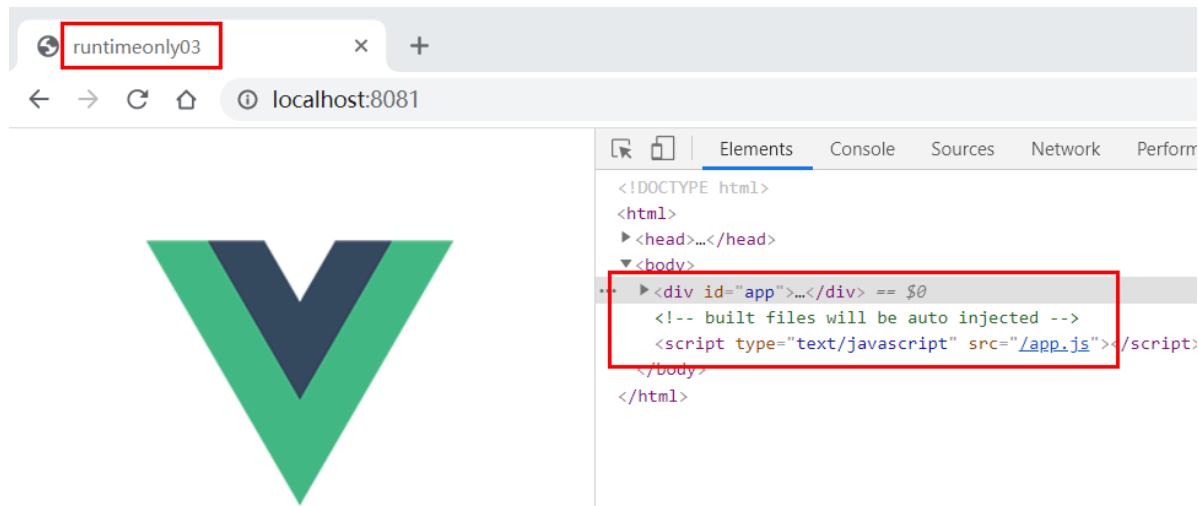
执行结果



Runtime-only

```
1 import vue from 'vue'
2 import App from './App'
3
4 vue.config.productionTip = false
5
6 /* eslint-disable no-new */
7 new vue({
8   el: '#app',
9   render: h => h(App)
10 })
```

执行结果



Vue文件中的template是怎么处理的?

是由vue-template-compiler处理的(render -> virtual dom -> UI)，可以在package.json文件中查看到版本号。

```

40   "semver": "^5.3.0",
41   "shelljs": "^0.7.6",
42   "uglifyjs-webpack-plugin": "^1.1.1",
43   "url-loader": "^0.5.8",
44   "vue-loader": "^13.3.0",
45   "vue-style-loader": "^3.0.1",
46   "vue-template-compiler": "^2.5.2",   
47   "webpack": "^3.6.0",
48   "webpack-bundle-analyzer": "^2.9.0",
49   "webpack-dev-server": "^2.9.1",

```

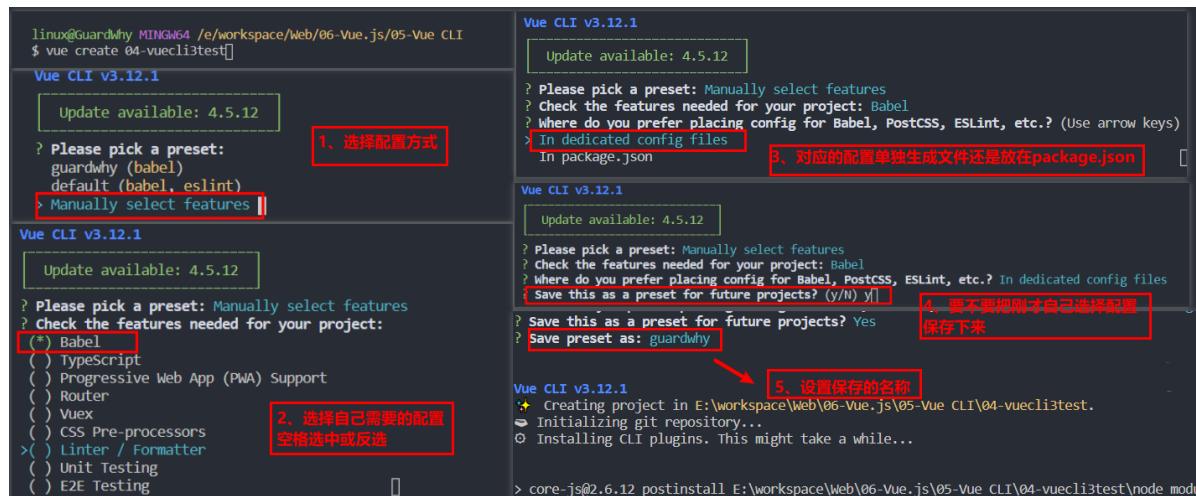
13.4 Vue CLI3 的使用

13.4.1 创建项目

使用Vue CLI3创建工程

1 | vue create vuecli3test

配置说明



目录结构详解

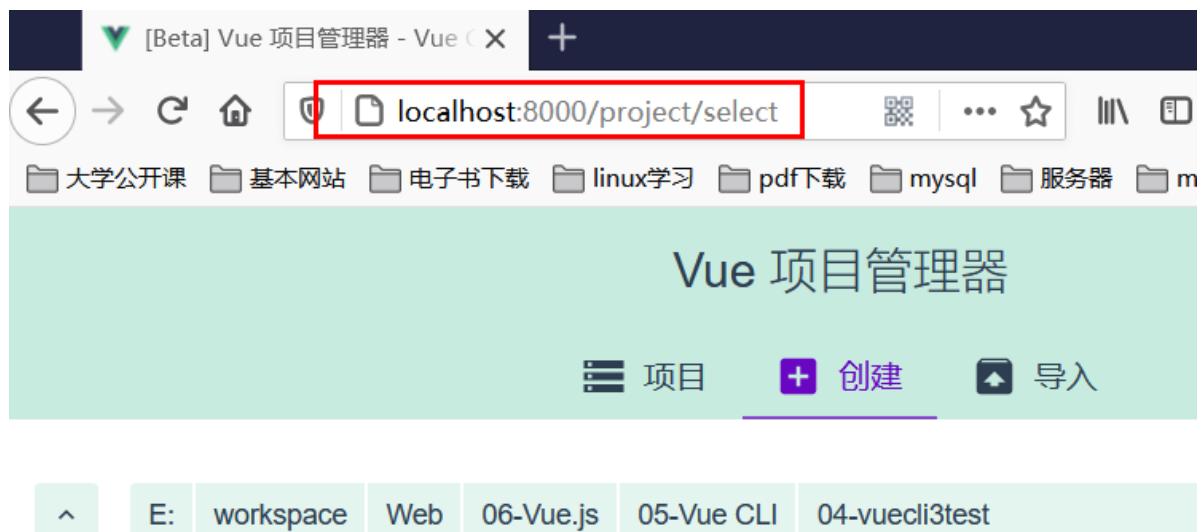


终端执行 `npm run serve`，执行成功！！！



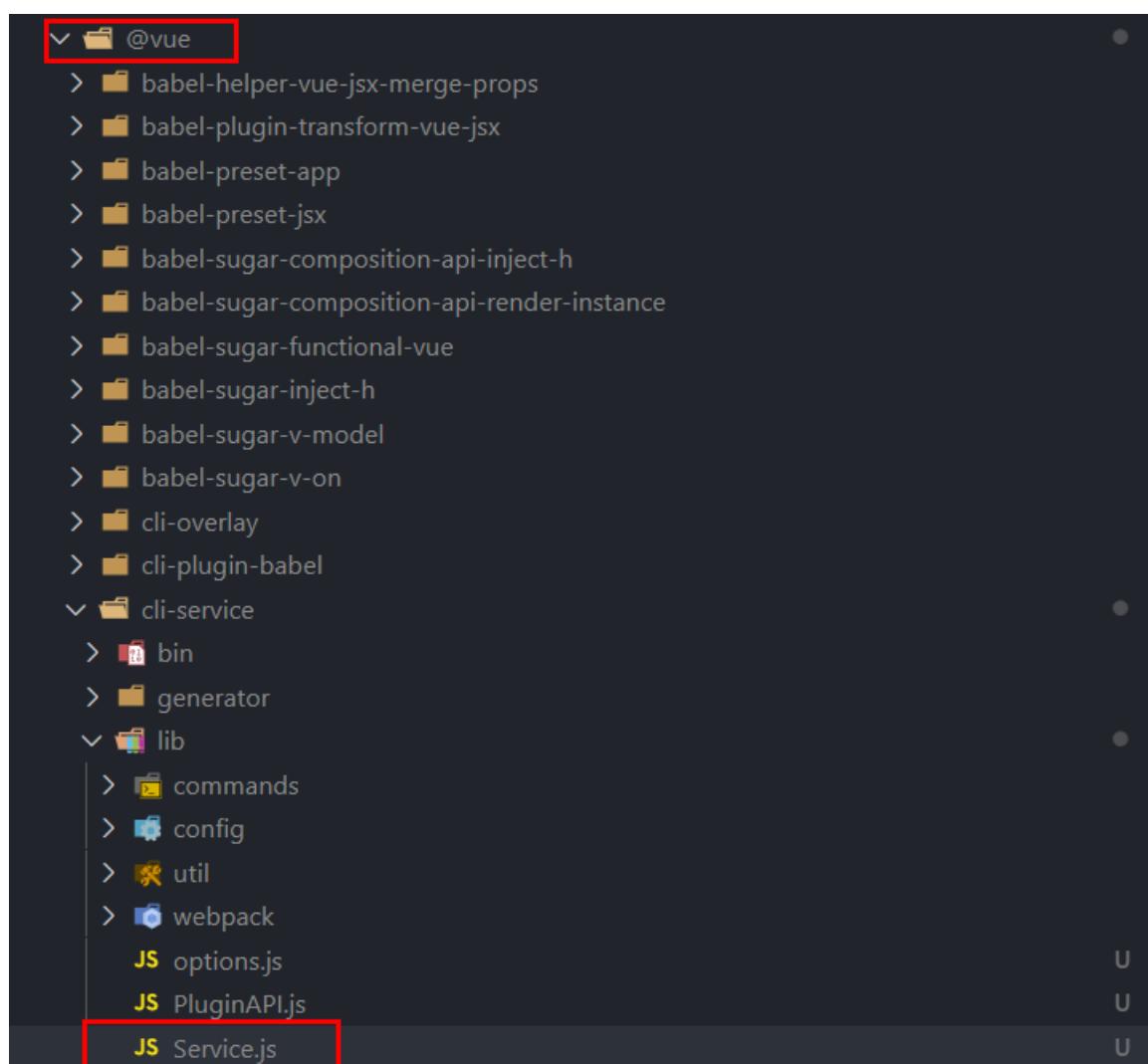
13.4.2 修改配置

方式1：终端执行 `vue ui` 命令，可直接在浏览器中就行修改。



node_modules

方式二：可以在Service.js修改！！



方式三：自定义配置，起别名建立一个vue.config.js文件

```
vue.config.js ×
04-vuecli3test > vue.config.js > [?] <unknown> > 🛡 configurWebpack
1  const path = require('path')
2  function resolve(dir){
3    return path.join(__dirname, dir)
4  }
5
6  module.exports ={
7    // 1. 基础的配置方式
8    configurWebpack:{
9      resolve:{
10        alias:{
11          'components': '@/components',
12          'pages': '@/pages'
13        }
14      }
15    }
16  }
```

14-Vue-router详解

14.1 什么是路由

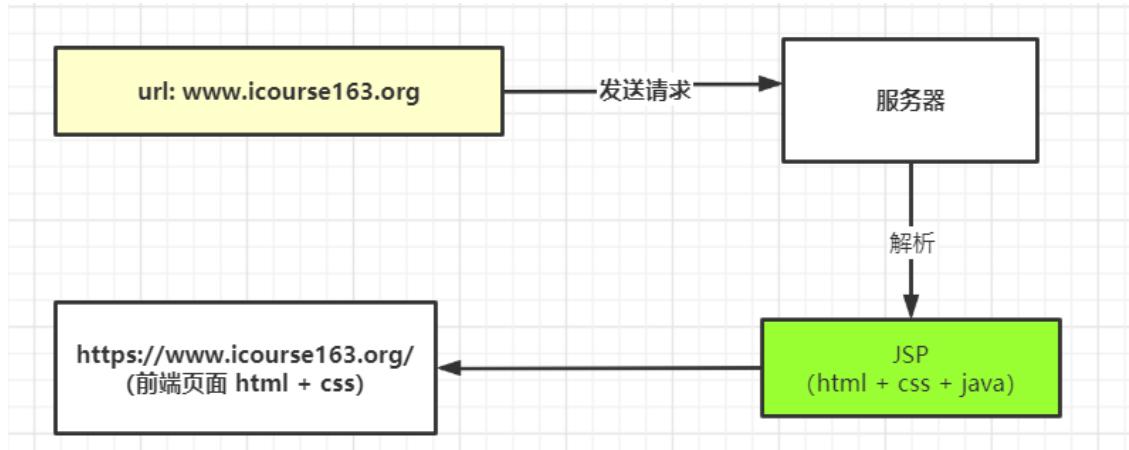
路由 (routing) 就是通过互联的网络把信息从源地址传输到目的地址的活动。路由中有一个非常重要的概念叫路由表，路由表本质上就是一个映射表，决定了数据包的指向。

后端路由渲染(JSP)

早期的网站开发整个HTML页面是由服务器来渲染的，服务器直接生产渲染好对应的HTML页面，返回给客户端进行展示。**后端处理URL和页面之间的映射关系！！**

具体流程

- 一个页面有自己对应的网址，也就是URL。
- URL会发送到服务器，服务器会通过正则对该URL进行匹配，并且最后交给一个控制器进行处理。
- 控制器进行各种处理，最终生成HTML或者数据，返回给前端（Java代码作用是从数据库中读取数据，并将它动态的放在页面中）。



后端渲染优点

当页面中需要请求不同的路径内容时,交给服务器来进行处理,服务器渲染好整个页面,并且将页面返回给客户顿。

这种情况下渲染好的页面,不需要单独加载任何的js和css,可以直接交给浏览器展示,这样也有利于SEO的优化。

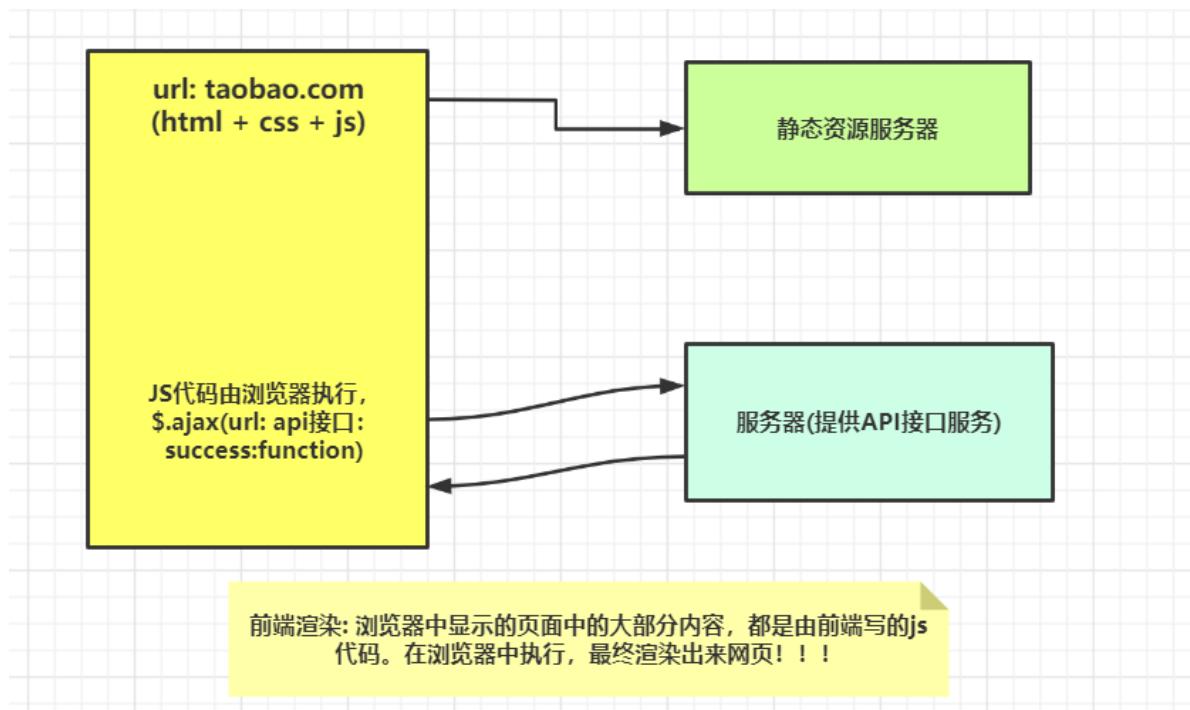
后端渲染缺点

- 是整个页面的模块由后端人员来编写和维护的。
- 另一种情况是前端开发人员如果要开发页面,需要通过PHP和Java等语言来编写页面代码。
- 而且通常情况下HTML代码和数据以及对应的逻辑会混在一起,编写和维护都是非常糟糕的事情。

前后端分离阶段

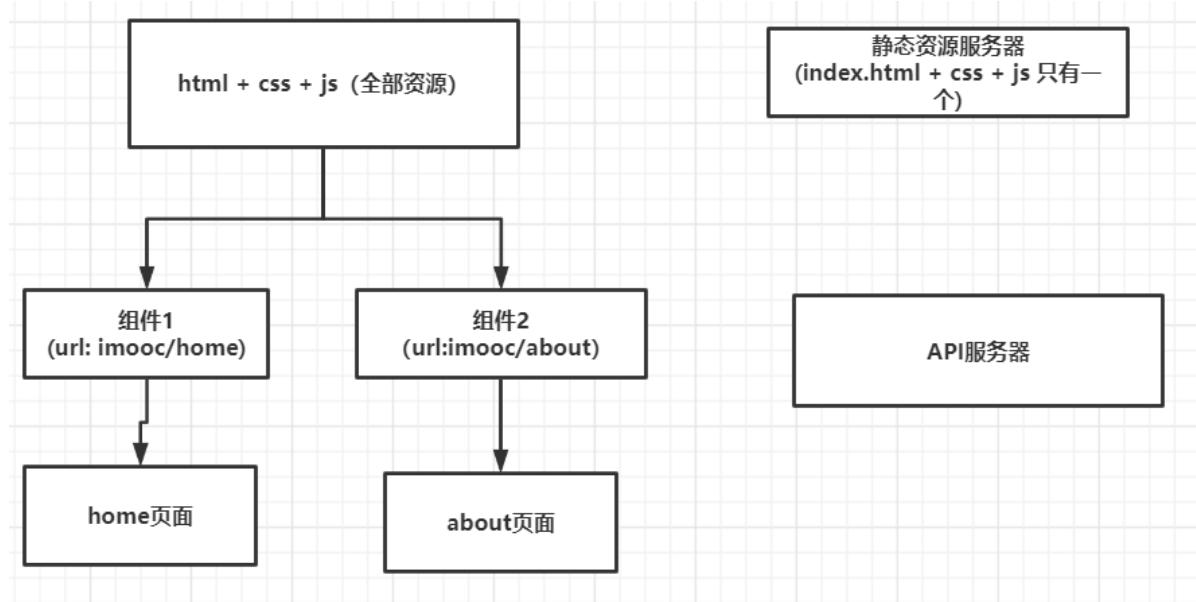
前后端分离优点

- 随着Ajax的出现,有了前后端分离的开发模式,后端只提供API来返回数据,前端通过Ajax获取数据,并且可以通过JavaScript将数据渲染到页面中。
- 这样做最大的优点就是前后端责任的清晰,后端专注于数据上,前端专注于交互和可视化上。
- 并且当移动端(iOS/Android)出现后,后端不需要进行任何处理,依然使用之前的一套API即可,目前很多的网站依然采用这种模式开发。



SPA页面阶段

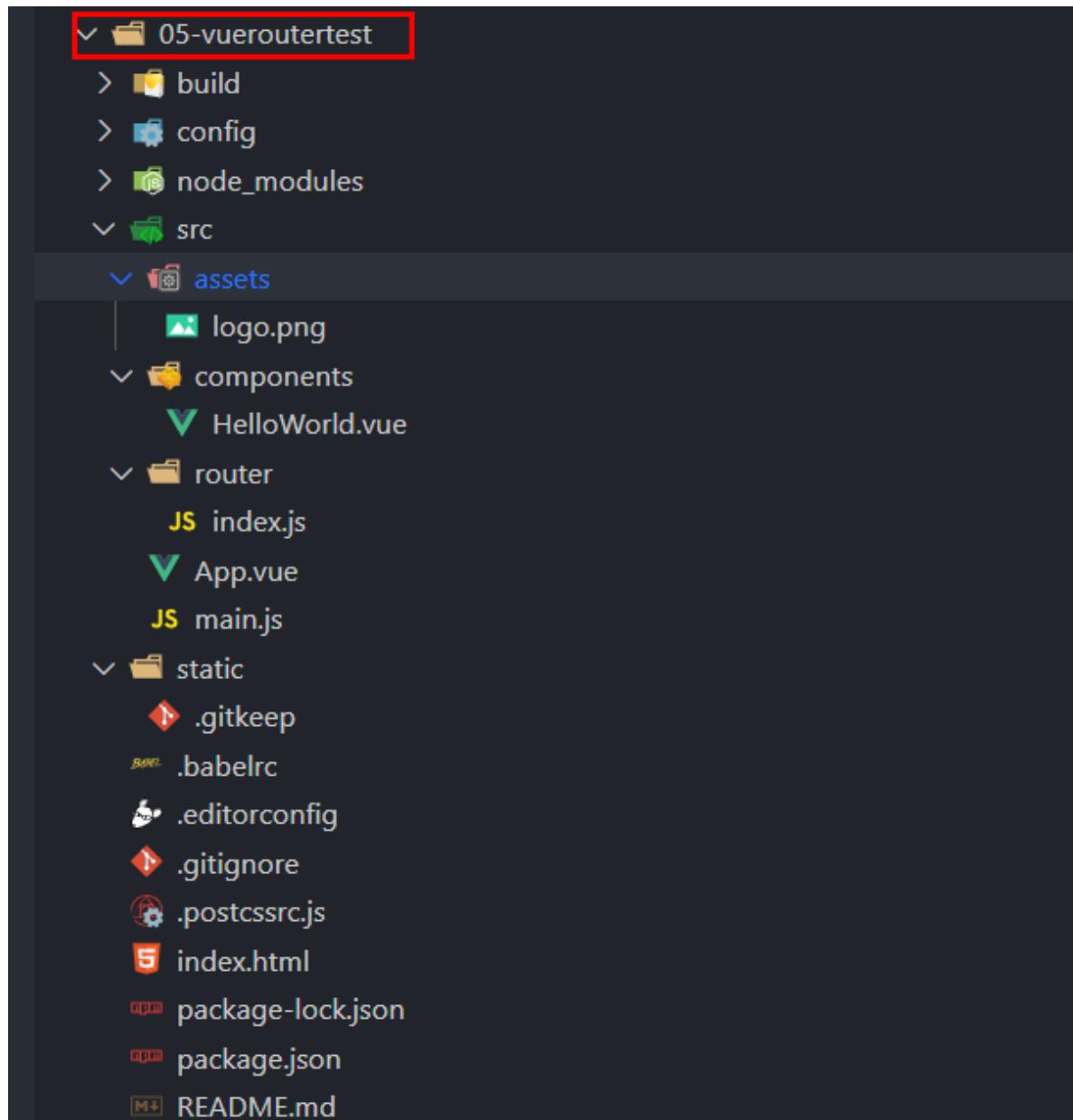
SPA页面是单页富应用,整个网页只有一个html页面!!!! 其实SPA最主要的特点就是在前后端分离的基础上加了一层前端路由。



14.2 认识vue-router

创建项目

```
1 | vue init webpack 05-vueroutertest
```



vue-router是Vue.js官方的路由插件，它和vue.js是深度集成的，适合用于构建单页面应用。

vue-router是基于路由和组件的

路由用于设定访问路径，将路径和组件映射起来，在vue-router的单页面应用中，页面的路径的改变就是组件的切换。

14.2.1 安装vue-router

步骤一：安装vue-router

```
1 | cnpm install vue-router --save
```

步骤二：在模块化工程中使用它(因为是一个插件，所以可以通过Vue.use()来安装路由功能)

- 第一步：导入路由对象，并且调用 Vue.use(VueRouter)。
- 第二步：创建路由实例，并且传入路由映射配置。

The screenshot shows the VS Code interface with the following details:

- Resource Explorer:** Shows the project structure:
 - 05-VUE CLI
 - 05-vueroutertest (selected)
 - src (selected)
 - router (selected)
 - index.js (selected)
- Editor:** The file `index.js` contains the following code:

```
1 // 配置路由相关的信息
2 import Vue from 'vue'
3 import VueRouter from 'vue-router'
4 import Router from 'vue-router'
5
6 // 1. 通过Vue.use(插件), 安装插件
7 Vue.use(VueRouter)
8
9 // 2. 定义路由
10 const routes = []
11
12 // 3. 创建router实例
13 const router = new VueRouter({
14   routes
15 })
16
17 // 4. 导出router实例
18 export default router
```

A red arrow points from the line `Vue.use(VueRouter)` in the editor back to the `Vue.use` line in the Resource Explorer.

- 第三步：在Vue实例中挂载创建的路由实例

The screenshot shows the VS Code interface with the following details:

- Resource Explorer:** Shows the project structure:
 - 05-VUE CLI
 - 05-vueroutertest (selected)
 - src (selected)
 - router (selected)
 - index.js (selected)
 - main.js (selected)
- Editor:** The file `main.js` contains the following code:

```
1 import Vue from 'vue'
2 import App from './App'
3 import router from './router'
4
5 Vue.config.productionTip = false
6
7 /* eslint-disable no-new */
8 new Vue({
9   el: '#app',
10  // 接收router对象, 进行挂载!!!
11  router,
12  render: h => h(App)
13})
14
```

Two red arrows point from the `import router from './router'` line and the `router` object in the `new Vue` configuration back to the `router` line in the Resource Explorer.

14.2.2 使用vue-router的步骤

第一步: 创建路由组件。

```

About.vue
1 <template>
2   <div>
3     <h3>这是关于内容</h3>
4     <p>python是最优雅的！！！</p>
5   </div>
6 </template>
7
8 <script>
9   export default {
10    name: "About"
11  }
12 </script>
13
14 <style scoped>
15
16 </style>
17

Home.vue
1 <template>
2   <div>
3     <h3>首页内容</h3>
4     <p>Vue.js python JavaScript </p>
5   </div>
6 </template>
7
8 <script>
9   export default {
10    name: "Home"
11  }
12 </script>
13
14 <style scoped lang="scss">
15
16 </style>
17

```

第二步: 配置路由映射: 组件和路径映射关系。

```

index.js
1 // 配置路由相关的信息
2 import Vue from 'vue'
3
4 // 导入组件
5 import Home from "../components/Home";
6 import About from "../components/About";
7 import VueRouter from 'vue-router';
8
9 // 1. 通过Vue.use(插件), 安装插件
10 Vue.use(VueRouter)
11
12 // 2. 定义路由
13 const routes = [
14   {
15     path: '/home',
16     component: Home
17   },
18   {
19     path: '/about',
20     component: About
21   },
22 ]
23
24 // 3. 创建router实例
25 const router = new VueRouter({
26   routes
27 })
28
29 // 4. 导出router实例
30 export default router

```

第三步: 使用路由: 通过 `<router-link>` 和 `<router-view>`。

- `<router-link>`: 该标签是一个vue-router中已经内置的组件, 它会被渲染成一个 `<a>` 标签。
- `<router-view>`: 该标签会根据当前的路径, 动态渲染出不同的组件。
- 网页的其他内容, 比如顶部的标题/导航, 或者底部的一些版权信息等会和 `<router-view>` 处于同一个等级。
- 在路由切换时, 切换的是 `<router-view>` 挂载的组件, 其他内容不会发生改变。

The screenshot shows the VS Code interface. On the left, the file tree displays a project structure with folders like .vscode, 02-runtimecompiler, 03-runtimeonly, 04-vuecli3test, 05-vueroutertest, build, config, node_modules, and src. Inside src, there are assets, components (with About.vue and Home.vue), router (index.js), and App.vue. The App.vue file is open in the editor, showing its template, script, and style sections. A red box highlights the router-link and router-view tags in the template.

```

<template>
  <div id="app">
    <h2>中国大学mooc</h2>
    <router-link to="/home">首页</router-link>
    <router-link to="/about">关于</router-link>
    <router-view></router-view>
    <h3>没有围墙的大学！！</h3>
  </div>
</template>

<script>
  export default {
    name: 'App'
  }
</script>

<style>
</style>

```

执行结果

Three browser tabs are shown, each displaying different parts of the application based on the URL:

- localhost:8080/#/**: Shows the main page with "中国大学mooc" and navigation links "首页 关于". A red box highlights the "About" link.
- localhost:8080/#/home**: Shows the "首页内容" section with "Vue.js python javaScript" and "没有围墙的大学！！". A red box highlights the "About" link in the navigation bar.
- localhost:8080/#/about**: Shows the "这是关于内容" section with "python是最优雅的！！！" and "没有围墙的大学！！". A red box highlights the "About" link in the navigation bar.

路由的默认路径

让路径默认跳到到首页, 并且 `<router-view>` 渲染首页组件。

index.js

```

{
  path: '/',
  // 重定向, 路由的默认路径
  redirect: '/home'
},

```

配置解析:

- 在routes中又配置了一个映射, path配置的是根路径: /
- redirect是重定向, 也就是我们将根路径重定向到/home的路径下。

HTML5的History模式

- 改变路径的方式有两种: URL的hash, HTML5的history。
- 默认情况下, 路径的改变使用的URL的hash。

如果希望使用HTML5的history模式, 非常简单, 进行如下配置即可。

The image shows two browser windows side-by-side. Both windows have a title bar '05-vueroutertest'. The left window's address bar is highlighted with a red box and contains 'localhost:8080/home'. The right window's address bar is also highlighted with a red box and contains 'localhost:8080/about'. Both windows display the same basic layout with a header containing the text '中国大学mooc' and a navigation bar with links like '首页 关于 首页内容'. The content area below the navigation bar contains text such as 'Vue.js python javaScript' and '没有围墙的大学！！'.

router-link其他属性

- tag可以指定 `<router-link>` 之后渲染成什么组件。
- replace不会留下history记录, 所以指定replace的情况下, 后退键返回不能返回到上一个页面中。
- 当 `<router-link>` 对应的路由匹配成功时, 会自动给当前元素设置一个`router-link-active`的class, 设置active-class可以修改默认的名称。但是通常不会修改类的属性, 会直接使用默认的`router-link-active`即可。

This screenshot shows a browser window with developer tools open, specifically the element inspector. The URL in the address bar is 'localhost:8080/home'. The element being inspected is a button with the class 'router-link-exact-active active'. This button is part of the 'app' component's DOM structure, which includes an 'h2' header and other buttons for 'About' and 'Home'. The 'Home' button is highlighted with a red box. To the right of the DOM tree, the application's content is displayed, showing the '中国大学mooc' header and the '首页内容' section with the text 'Vue.js python javaScript' and '没有围墙的大学！！'.

修改linkActiveClass

class具体的名称也可以通过router实例的属性进行修改。exact-active-class类似于active-class, 只是在精准匹配下才会出现的class。

The screenshot shows two code editors side-by-side. On the left, the file `index.js` contains Vue Router configuration:

```

17     redirect: '/home'
18   },
19   {
20     path: '/home',
21     component: Home
22   },
23   {
24     path: '/about',
25     component: About
26   },
27 ]
28
29 // 3. 创建router实例
30 const router = new VueRouter({
31   // 配置路由和组件之间的应用关系
32   routes,
33   mode:'history',
34   linkActiveClass: 'active' // 3.1 配置路由和组件之间的应用关系
35 })
36
37 // 4. 导出router实例
38 export default router

```

On the right, the file `App.vue` contains the template and script sections:

```

1 <template>
2   <div id="app">
3     <h2>中国大学mooc</h2>
4     <router-link to="/home" tag="button" replace>首页</router-link>
5     <router-link to="/about" tag="button" replace>关于</router-link>
6     <router-view></router-view>
7     <h3>没有围墙的大学！！</h3>
8   </div>
9 </template>
10
11 <script>
12   export default {
13     name: 'App'
14   }
15 </script>
16
17 <style>
18   .active{ // 3.1 配置路由和组件之间的应用关系
19     color: red;
20   }
21 </style>

```

A red arrow points from the `linkActiveClass` configuration in `index.js` to the `.active` class definition in `App.vue`.

14.2.3 路由代码跳转

有时候，页面的跳转可能需要执行对应的JavaScript代码，这个时候，就可以使用第二种跳转方式了。

The screenshot shows the `App.vue` file with a button click event handler:

```

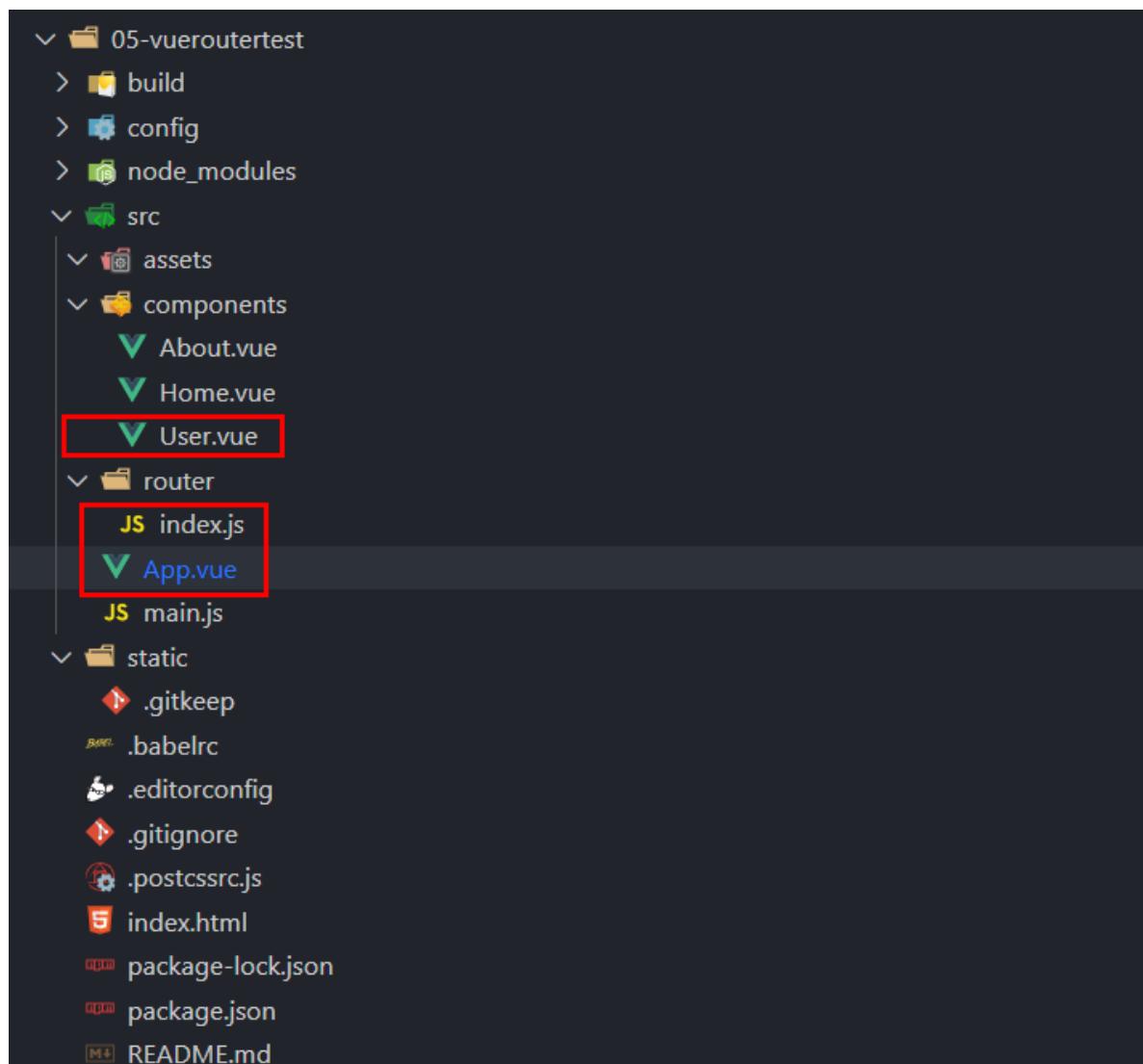
1 <template>
2   <div id="app">
3     <h2>中国大学mooc</h2>
4     <!-- <router-link to="/home" tag="button" replace>首页</router-link>
5     <router-link to="/about" tag="button" replace>关于</router-link>
6     <button @click="homeClick">首页</button>
7     <button @click="aboutClick">关于</button>
8     <router-view></router-view>
9     <h3>没有围墙的大学！！</h3>
10    </div>
11 </template>
12
13 <script>
14   export default {
15     name: 'App',
16     methods: {
17       homeClick(){ // 1. 通过代码的方式修改路由。
18         this.$router.replace('/home')
19         console.log('homeClick');
20       },
21       aboutClick(){ // 2. 通过代码的方式修改路由。
22         this.$router.replace('/about')
23         console.log('aboutClick')
24       },
25     },
26   }
27 </script>

```

A red box highlights the `homeClick` and `aboutClick` methods, which use `this.$router.replace` to change the route.

14.3 路由的懒加载

14.3.1 动态路由



在某些情况下，一个页面的path路径可能是不确定的，当进入用户界面时，希望是如下的路径。

/user/userid或/user/username，这种path和Component的匹配关系，称之为动态路由。

代码示例

User.vue

```
1 <template>
2   <div>
3     <h3>用户界面</h3>
4     <p>学习Vue.js!!!</p>
5     <h3>{{userId}}</h3>
6     <!--直接拿-->
7     <!-- <h2>$route.params.userId</h2> -->
8   </div>
9 </template>
10
11 <script>
12   export default {
13     name: "User",
14     computed: {
15       userId() {
16         // 拿到的是哪个路由处于活跃状态，就是拿到哪个路由！！
```

```

17         return this.$route.params.userid
18     }
19   }
20 }
21 </script>
22
23 <style scoped>
24
25 </style>

```

index.js

```

1 // 配置路由相关的信息
2 import Vue from 'vue'
3
4 // 导入组件
5 import User from "../components/User";
6 import VueRouter from 'vue-router';
7
8 // 1.通过Vue.use(插件), 安装插件
9 Vue.use(VueRouter)
10
11 // 2.定义路由
12 const routes = [
13   {
14     path: '/user/:userid',
15     component: User
16   }
17 ]
18
19 // 3.创建router实例
20 const router = new VueRouter({
21   // 配置路由和组件之间的应用关系
22   routes,
23   mode: 'history',
24   linkActiveClass: 'active'
25 })
26
27 // 4.导出router实例
28 export default router

```

APP.vue

```

1 <template>
2   <div id="app">
3     <h2>中国大学mooc</h2>
4     <router-link :to="'/user/' + userId" tag="button" replace>用户</router-
5       link>
6       <router-view></router-view>
7       <h3>没有围墙的大学！！</h3>
8     </div>
9   </template>
10
11 <script>
12   export default {
13     name: 'App',
14   }

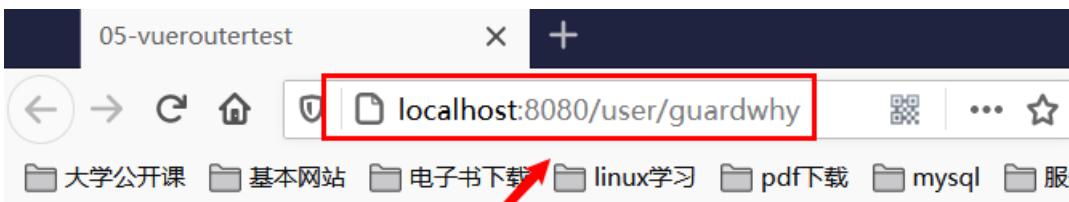
```

```

13     data() {
14       return {
15         userId: 'guardwhy'
16       }
17     }
18   }
19 </script>
20
21 <style>
22   .active{
23     color: red;
24   }
25 </style>

```

执行结果



中国大学mooc

首页 关于 **用户**

用户界面

学习Vue.js!!!

guardwhy

没有围墙的大学！！

14.3.2 路由的懒加载

当打包构建应用时，Javascript 包会变得非常大，影响页面加载。

如果能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了。

路由懒加载做了什么？

路由懒加载的主要作用就是将路由对应的组件打包成一个个的js代码块。

只有在这个路由被访问到的时候，才加载对应的组件。

懒加载的方式

方式一：结合Vue的异步组件和Webpack的代码分析。

```

1 | const Home = resolve => { require.ensure(['./components/Home.vue'], () => {
2 |   resolve(require('./components/Home.vue')) });

```

方式二：AMD写法

```
1 | const About = resolve => require(['../components/About.vue'], resolve);
```

方式三：在ES6中，我们可以有更加简单的写法来组织Vue异步组件和Webpack的代码分割。

```
1 | const Home = () => import('../components/Home.vue')
```

路由懒加载的效果

The screenshot shows two code editors side-by-side. The left editor contains the router configuration from the previous slide, while the right editor shows the resulting build output.

Left Editor (Router Configuration):

```
JS index.js
05-vueroutertest > src > router > JS index.js > [e] default
1 // 配置路由相关的信息
2 import Vue from 'vue'
3
4 // 导入组件
5 import Home from "../components/Home";
6 import About from "../components/About";
7 import User from "../components/User";
8 import VueRouter from 'vue-router';
9
10 // 1. 通过Vue.use(插件), 安装插件
11 Vue.use(VueRouter)
```

Right Editor (Build Output):

```
JS index.js
05-vueroutertest > src > router > JS index.js > ...
1 // 配置路由相关的信息
2 import Vue from 'vue'
3
4 // 导入组件
5 import VueRouter from 'vue-router';
6
7 // 路由懒加载
8 const Home = () => import('../components/Home')
9 const About = () => import('../components/About')
10 const User = () => import('../components/User')
11
12 // 1. 通过Vue.use(插件), 安装插件
13 Vue.use(VueRouter)
```

Build Output (dist folder):

- css
- js
 - app.03392e7e6b905d12485f.js
 - app.03392e7e6b905d12485f.js.map
 - manifest.2ae2e69a05c33dfc65f8.js
 - manifest.2ae2e69a05c33dfc65f8.js.map
 - vendor.6babab3bf80904d146eba.js
 - vendor.6babab3bf80904d146eba.js.map

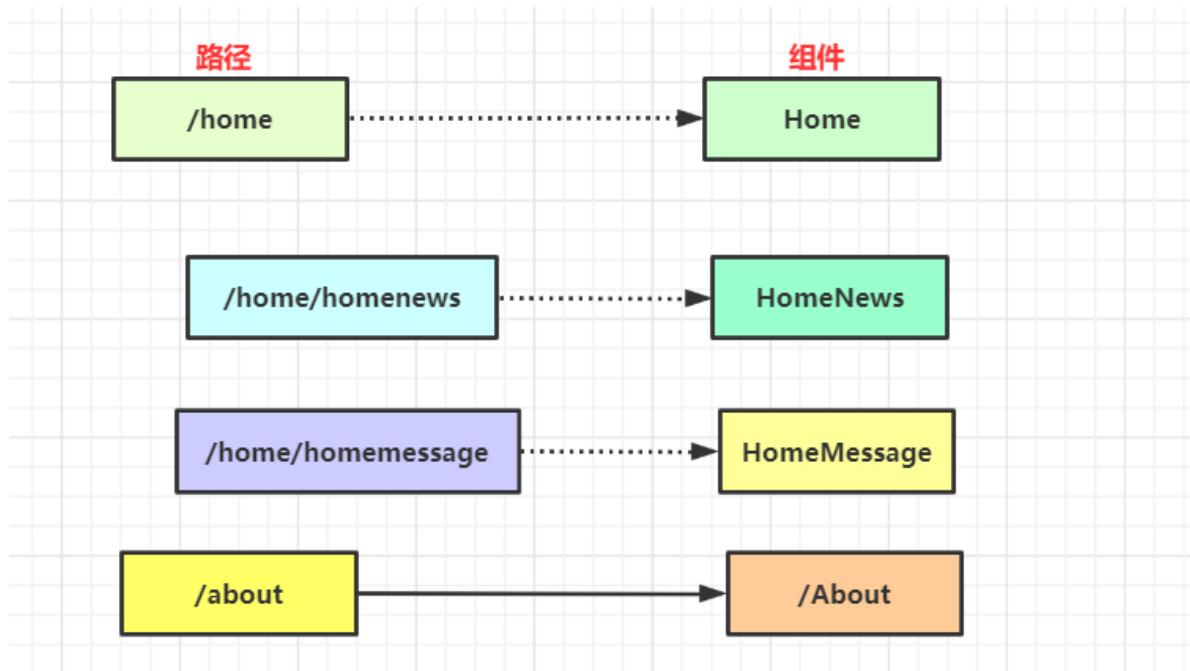
Build Output (dist/js folder):

- 引用后直接使用
- JS 0.6af5d79716f3df214018.js
- 0.6af5d79716f3df214018.js.map
- 1.789b1b850dba97405b11.js
- 1.789b1b850dba97405b11.js.map
- 2.fc7a9ae755b5e4db9699.js
- 2.fc7a9ae755b5e4db9699.js.map
- app.b62dd8439d0f825a25e8.js
- app.b62dd8439d0f825a25e8.js.map
- manifest.2be3e915f2186b0ec957.js
- manifest.2be3e915f2186b0ec957.js.map
- vendor.bf721f00db70d95068a0.js
- vendor.bf721f00db70d95068a0.js.map

14.4 嵌套路由

嵌套路由是一个很常见的功能。比如在home页面中，希望通过/home/news和/home/message访问一些内容，一个路径映射一个组件，访问这两个路径也会分别渲染两个组件。

路径和组件的关系如下



实现嵌套路由有两个步骤

创建对应的子组件，并且在路由映射中配置对应的子路由。

```
<template>
<div>
  <ul>
    <li>jdk16发布了！！！</li>
    <li>python最优雅！！</li>
    <li>C++人工智能！！</li>
    <li>linux好玩！！</li>
  </ul>
</div>
<script>
  export default {
    name: "HomeMessage"
  }
</script>
<style scoped>
```

```
<template>
<div>
  <ul>
    <li>hello Vue.js!!!</li>
    <li>PHP是最好的语言!!!</li>
    <li>JavaScript就是大流氓！！！</li>
    <li>Node.js好香！！</li>
  </ul>
</div>
<script>
  export default {
    name: "HomeNews"
  }
</script>
<style scoped>
```

index.js

```
// 路由懒加载
const Home = () => import('../components/Home')
const HomeNews = () => import('../components/HomeNews')
const HomeMessage = () => import('../components/HomeMessage')
const About = () => import('../components/About')
const User = () => import('../components/User')

// 1. 通过Vue.use(插件), 安装插件
Vue.use(VueRouter)

// 2. 定义路由
const routes = [
  {
    path: '',
    // 重定向, 路由的默认路径
    redirect: '/home'
  },
  {
    path: '/home',
    component: Home,
    children:[
      // 重定向, 默认路径
      {
        path: '',
        redirect: 'homenews'
      },
      {
        path: 'homenews',
        component: HomeNews
      },
      {
        path: 'homemessage',
        component: HomeMessage
      }
    ]
  }
],
```

在组件内部使用 `<router-view>` 标签。

```

    资源管理器
    ...
    Home.vue 05-vueroutertest\src\components
    05-VUE CLI
    ...
    src
        assets
            components
                About.vue
                Home.vue
                HomeMessage.vue
                HomeNews.vue
                User.vue
        router
            index.js
        App.vue

```

`<template>`

- 1 <div>
- 2 <h3>首页内容</h3>
- 3 <p>Vue.js python javaScript </p>
- 4 <!--添加router-Link-->
- 5 <router-link to="/home/homenews">新闻</router-link>
- 6 <router-link to="/home/homemessage">消息</router-link>
- 7 <router-view></router-view>
- 8 </div>
- 9 </template>
- 10 <script>
- 11 export default {
- 12 name:"Home"
- 13 }
- 14 </script>
- 15 <style scoped>
- 16 </style>

执行结果

05-vueroutertest x + localhost:8080/home/homenews

中国大学mooc

首页 关于 用户

新闻 消息

- hello Vuejs!!!
- PHP是最好的语言!!!
- JavaScript就是大流氓!!!
- Nodejs好香!!!

没有围墙的大学！！

05-vueroutertest x + localhost:8080/home/homemessage

中国大学mooc

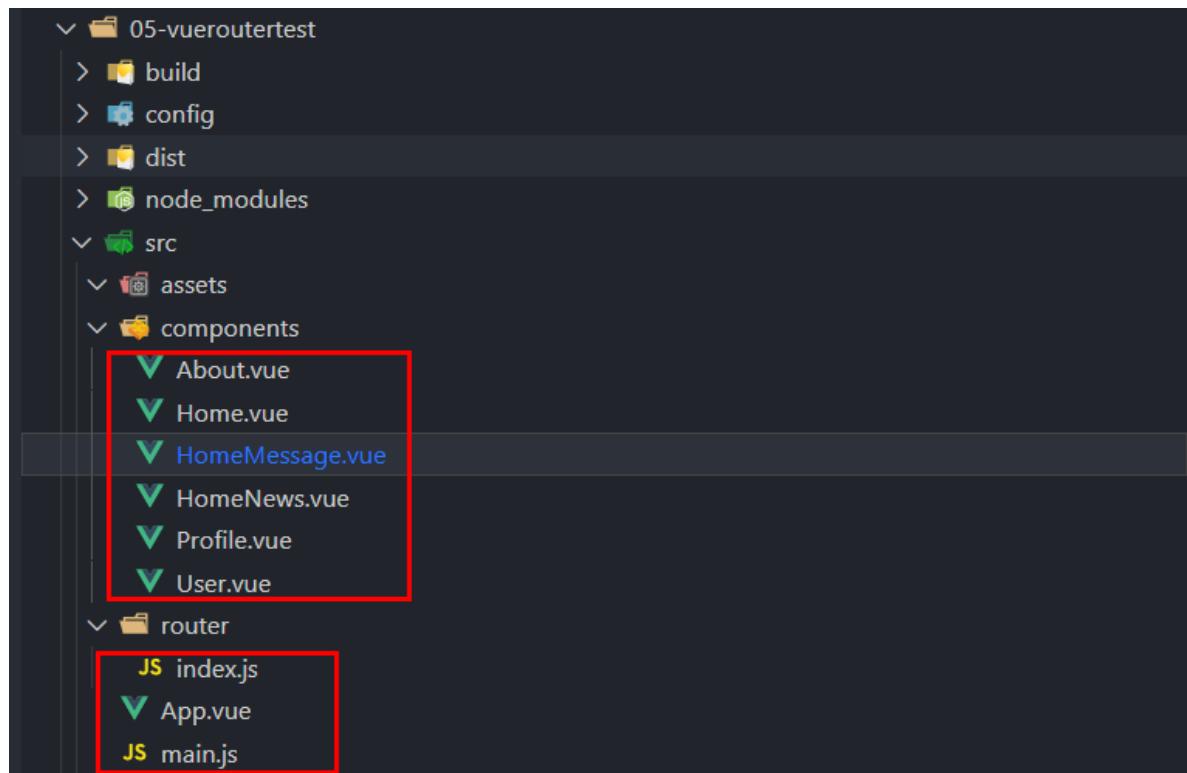
首页 关于 用户

新闻 消息

- jdk16发布了!!!
- python最优雅!!!
- C++人工智能!!!
- linux好玩!!!

没有围墙的大学！！

14.5 传递参数的方式



传递参数主要有两种类型: params和query。

params的类型

- 配置路由格式: /router/:id
- 传递的方式: 在path后面跟上对应的值
- 传递后形成的路径: /router/123, /router/abc。

query的类型

- 配置路由格式: /router, 也就是普通配置。
- 传递的方式: 对象中使用query的key作为传递方式。
- 传递后形成的路径: /router?id=123, /router?id=abc。

传递参数方式一:

Profile.vue

```
1 <template>
2   <div>
3     <p>用户基本档案</p>
4   </div>
5 </template>
6
7 <script>
8   export default {
9     name:"Profile"
10    }
11 </script>
12
13 <style scoped>
14 </style>
```

index.js

```
1 // 配置路由相关的信息
```

```

2 import Vue from 'vue'
3
4 // 导入组件
5 import VueRouter from 'vue-router';
6
7 // 路由懒加载
8 const Profile = () => import('../components/Profile')
9
10 // 1.通过Vue.use(插件), 安装插件
11 Vue.use(VueRouter)
12
13 // 2.定义路由
14 const routes = [
15   {
16     path: '/profile',
17     component: Profile
18   }
19 ]
20
21 // 3.创建router实例
22 const router = new VueRouter({
23   // 配置路由和组件之间的应用关系
24   routes,
25   mode: 'history',
26   linkActiveClass: 'active'
27 })
28
29 // 4.导出router实例
30 export default router

```

App.vue

The screenshot shows the VS Code interface with the following details:

- Resource Manager:** Shows the project structure: 05-vueroutertest > src > App.vue.
- Editor:** The file **App.vue** is open, displaying the following code:

```

<template>
  <div id="app">
    <h2>中国大学mooc</h2>
    <router-link to="/home" tag="button" replace>首页</router-link>
    <router-link to="/about" tag="button" replace>关于</router-link>
    <router-link :to="'/user/' + userId" tag="button" replace>用户</router-link>
    <router-link :to="{path: '/profile', query: {name: 'guardwhy', age: 28, height: 1.72}}">
      用户档案
    </router-link>
    <router-view></router-view>
    <h3>没有围墙的大学！！</h3>
  </div>
</template>

<script>
  export default {
    name: 'App',
    data() {
      return {
        userId: 'guardwhy'
      }
    },
    methods: {
    }
  }
</script>

```

传递参数方式二: JavaScript代码

App.vue

```

1 <template>
2   <div id="app">
3     <h2>中国大学mooc</h2>
4     <button @click="userClick">用户</button>
5     <button @click="profileClick">档案</button>

```

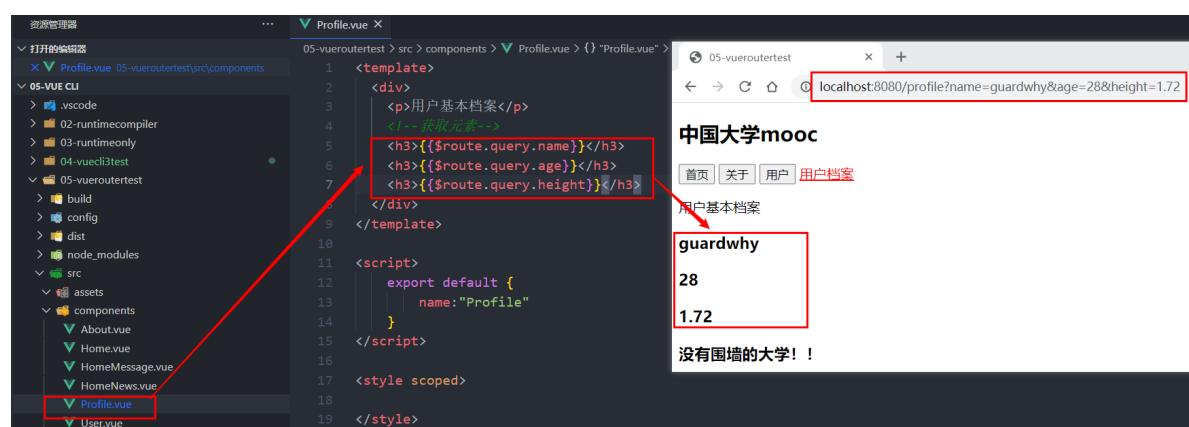
```

6      <router-view></router-view>
7          <h3>没有围墙的大学！！</h3>
8      </div>
9  </template>
10
11 <script>
12     export default {
13         name: 'App',
14         data() {
15             return {
16                 userId: 'guardwhy'
17             }
18         },
19         methods: {
20             userClick(){
21                 this.$router.push('/user/'+ this.userId)
22             },
23             profileClick(){
24                 this.$router.push({
25                     path: '/profile',
26                     query:{
27                         name: 'guardwhy',
28                         age : 28,
29                         height: 1.72
30                     }
31                 })
32             }
33         },
34     }
35 </script>
36
37 <style>
38     .active{
39         color: red;
40     }
41 </style>

```

获取参数

获取参数通过\$route对象获取的，在使用了 vue-router 的应用中，路由对象会被注入每个组件中，赋值为 this.\$route， 并且当路由切换时，路由对象会被更新。



14.6 导航守卫

14.6.1 为什么使用导航守卫?

普通的修改方式

- 能比较容易想到的修改标题的位置是每一个路由对应的组件.vue文件中。
- 通过mounted声明周期函数, 执行对应的代码进行修改即可。
- 但是当页面比较多时, 这种方式不容易维护(因为需要在多个页面执行类似的代码)。

什么是导航守卫?

vue-router提供的导航守卫主要用来监听监听路由的进入和离开的。

vue-router提供了beforeEach和afterEach的钩子函数, 它们会在路由即将改变前和改变后触发.

14.6.2 导航守卫使用

可以利用beforeEach来完成标题的修改

首先, 我们可以在钩子当中定义一些标题, 可以利用meta来定义。

资源管理器

... JS index.js X

05-vueroutertest > src > router > JS index.js > router.beforeEach

The screenshot shows the VS Code interface with the file structure on the left and the code editor on the right. The file structure includes .vscode, 02-runtimecompiler, 03-runtimeonly, 04-vuedcli3test, 05-vueroutertest (which contains build, config, dist, node_modules, and src), and components (About.vue, Home.vue, HomeMessage.vue, HomeNews.vue, Profile.vue, User.vue). The router folder contains index.js, App.vue, and main.js. The static folder contains .gitkeep, .babelrc, .editorconfig, .gitignore, .postcssrc.js, index.html, package-lock.json, package.json, and README.md. The code editor shows the router/index.js file with the following content:

```
// 2. 定义路由
const routes = [
  {
    path: '',
    // 重定向, 路由的默认路径
    redirect: '/home'
  },
  {
    path: '/home',
    component: Home,
    meta: {
      title: '首页'
    },
    children: [ ... ]
  },
  {
    path: '/about',
    component: About,
    meta: {
      title: '关于'
    }
  },
  {
    path: '/user/:userid',
    component: User,
    meta: {
      title: '用户'
    }
  },
  {
    path: '/profile',
    component: Profile,
    meta: {
      title: '档案'
    }
  }
]
```

Red arrows point from the highlighted 'index.js' files in the file tree to the corresponding 'meta: { title: ... }' blocks in the code editor.

其次, 利用导航守卫(前置钩子hook),修改我们的标题。

资源管理器

- ✓ 打开的编辑器
 - JS index.js 05-vueroutertest\src\router
- ✓ 05-VUE CLI
 - > .vscode
 - > 02-runtimecompiler
 - > 03-runtimeonly
 - > 04-vuecli3test
 - ✓ 05-vueroutertest
 - > build
 - > config
 - > dist
 - > node_modules
 - ✓ src
 - ✓ assets
 - ✓ components
 - ✓ About.vue
 - ✓ Home.vue
 - ✓ HomeMessage.vue
 - ✓ HomeNews.vue
 - ✓ Profile.vue
 - ✓ User.vue
 - ✓ router
 - ✓ JS index.js
 - ✓ App.vue
 - JS main.js

... JS index.js X

05-vueroutertest > src > router > JS index.js > ...

```

70
71 // 3. 创建router实例
72 const router = new VueRouter({
73   // 配置路由和组件之间的应用关系
74   routes,
75   mode: 'history',
76   linkActiveClass: 'active'
77 })
78
79 /*
80 导航钩子的三个参数解析:
81 to: 即将要进入的目标的路由对象。
82 from: 当前导航即将要离开的路由对象。
83 next: 调用该方法后, 才能进入下一个钩子。
84 */
85 router.beforeEach((to, from, next) =>{
86   // 从from跳转到to
87   document.title = to.matched[0].meta.title
88   next()
89 }
90
91 // 4. 导出router实例
92 export default router

```

执行结果

用户

关于

localhost:8080/user/guardwhy

localhost:8080/about

中国大学mooc

首页 关于 用户 用户档案

用户界面

学习Vue.js!!!

guardwhy

没有围墙的大学！！

中国大学mooc

首页 关于 用户 用户档案

这是关于内容

python是最优雅的！！！

没有围墙的大学！！

14.6.3 导航守卫补充

如果是后置钩子, 也就是afterEach, 不需要主动调用next()函数。

```

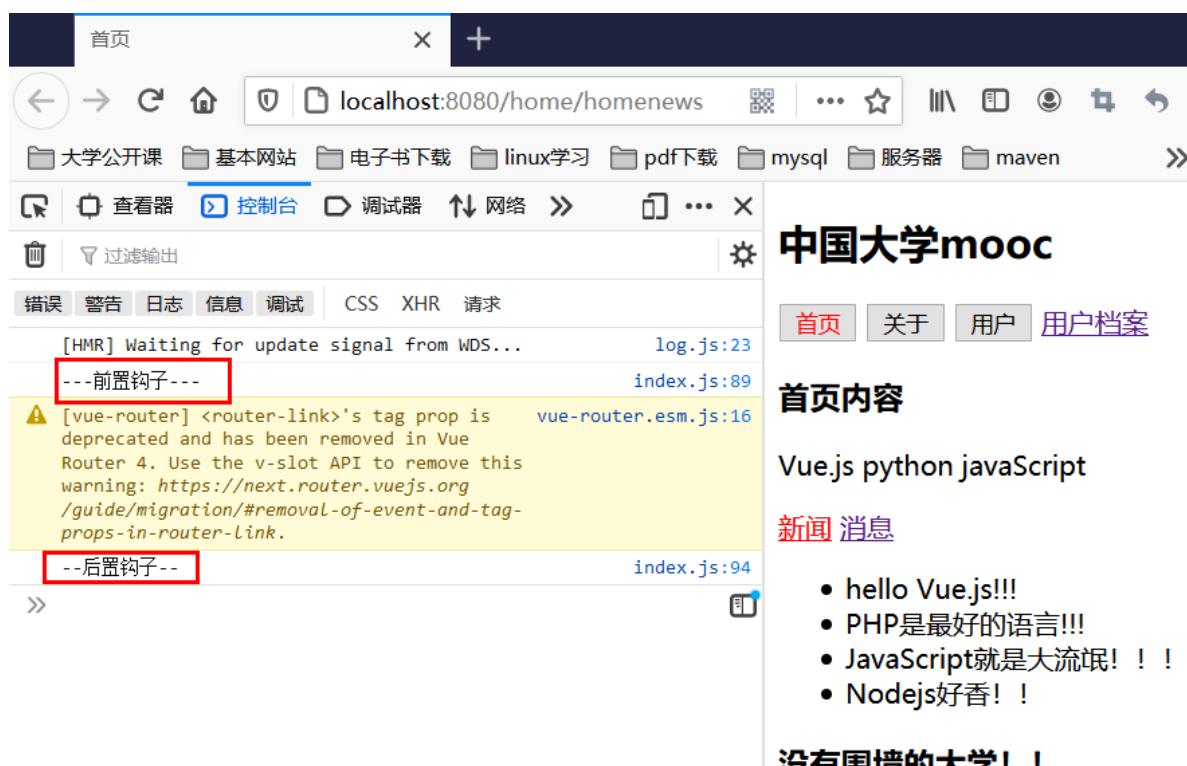
    /*
    导航钩子的三个参数解析:
    to: 即将要进入的目标的路由对象。
    from: 当前导航即将要离开的路由对象。
    next: 调用该方法后, 才能进入下一个钩子。
    */
    router.beforeEach((to, from, next) => {
      // 从from跳转到to
      document.title = to.matched[0].meta.title
      next()
      console.log("---前置钩子---")
    })

    // 后置钩子(hook)
    router.afterEach((to, from) => {
      console.log("--后置钩子--");
    })
  
```

// 4. 导出router实例

export default router

执行结果

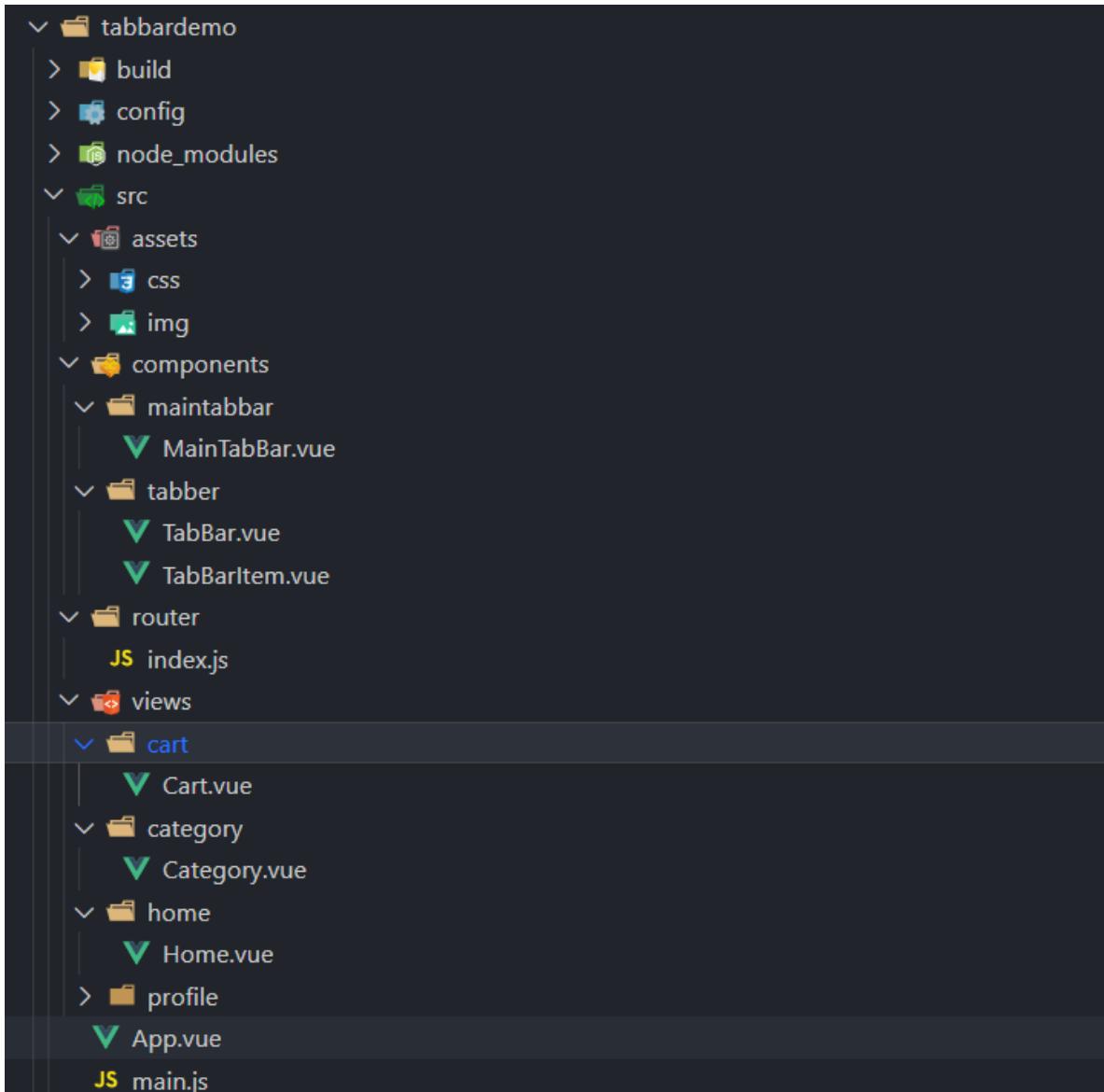


钩子函数参考文档: <https://router.vuejs.org/zh/guide/advanced/navigation-guards.html>

15- TabBar实现

15.1 项目目录

创建项目: `vue init webpack tabbardemo`



15.2 项目实现代码

项目所需的图片：<https://pan.baidu.com/s/1T0LxSsT4yGamy3gilKcNfw> 提取码：bo79

App.vue

```
1 <template>
2   <div id="app">
3     <!--使用组件-->
4     <router-view></router-view>
5     <main-tab-bar></main-tab-bar>
6   </div>
7 </template>
8
9 <script>
10 // 导入主键
11 import MainTabBar from "components/maintabbar/MainTabBar";
12
13 export default {
14   name: 'App',
15   // 注册组件
16   components:{
17     MainTabBar
```

```
18     }
19
20   }
21 </script>
22
23 <style>
24   @import "./assets/css/base"
25 </style>
```

MainTabBar.vue

```
1 <template>
2   <tab-bar>
3     <tab-bar-item path="/home" active-color="pink">
4       
5       
7         <div slot="item-text">首页</div>
8     </tab-bar-item>
9     <tab-bar-item path="/category" active-color="pink">
10       
11       
13         <div slot="item-text">分类</div>
14     </tab-bar-item>
15     <tab-bar-item path="/cart" active-color="pink">
16       
17       
19         <div slot="item-text">购物车</div>
20     </tab-bar-item>
21     <tab-bar-item path="/profile" active-color="deepPink">
22       
23       
25         <div slot="item-text">我的</div>
26   </tab-bar>
27 </template>
28
29 <script>
30 // 导入组件
31 import TabBarItem from 'components/tabber/TabBarItem'
32 import Tabbar from 'components/tabber/Tabbar'
33   export default {
34     name: "MainTabBar",
35     components: {
36       TabBarItem,
37       Tabbar
38     }
39   }
40 </script>
41
42 <style scoped>
43 </style>
```

TabBar.vue

```
1 <template>
2   <div id="tab-bar">
3     <slot></slot>
4   </div>
5 </template>
6
7 <script>
8   export default {
9     name: "TabBar"
10    }
11 </script>
12
13 <style scoped>
14   #tab-bar{
15     display: flex;
16     background-color: #ff6666;
17
18     position: fixed;
19     left: 0;
20     right: 0;
21     bottom: 0;
22     box-shadow: 0 -1px 1px rgba(100, 100, 100,.2);
23   }
24 </style>
```

TabBarItem.vue

```
1 <template>
2   <div class="tab-bar-item" @click="itemClick">
3     <div v-if="!isActive"><slot name="item-icon"></slot></div>
4     <div v-else><slot name="item-icon-active"></slot></div>
5     <div :style="activeStyle"><slot name="item-text"></slot></div>
6   </div>
7 </template>
8
9 <script>
10  export default {
11    name: "TabBarItem",
12    // 传递参数
13    props:{
14      path:String,
15      activeColor:{
16        type:String,
17        default:'red'
18      }
19    },
20    computed:{
21      isActive(){
22        return this.$route.path.indexOf(this.path) !==-1
23      },
24      activeStyle(){
25        return this.isActive ? {color : this.activeColor} : {}
26      }
27    },
28  </script>
```

```
28     methods: {
29         itemClick(){
30             this.$router.replace(this.path)
31         }
32     }
33 }
34 </script>
35
36 <style scoped>
37 .tab-bar-item{
38     flex: 1;
39     text-align: center;
40     height: 49px;
41     font-size: 14px;
42 }
43
44 .tab-bar-item img {
45     width: 24px;
46     height: 24px;
47     margin-top: 3px;
48     vertical-align: middle;
49     margin-bottom: 2px;
50 }
51 </style>
```

Cart.vue

```
1 <template>
2     <h3>购物车</h3>
3 </template>
4
5 <script>
6     export default {
7         name: "Cart"
8     }
9 </script>
10
11 <style scoped>
12 </style>
```

Category.vue

```
1 <template>
2     <h3>分类</h3>
3 </template>
4
5 <script>
6     export default {
7         name: "Category"
8     }
9 </script>
10
11 <style scoped>
12 </style>
13
```

Home.vue

```
1 <template>
2   <h3>首页</h3>
3 </template>
4
5 <script>
6   export default {
7     name: "Home"
8   }
9 </script>
10
11 <style scoped>
12
13 </style>
```

Profile.vue

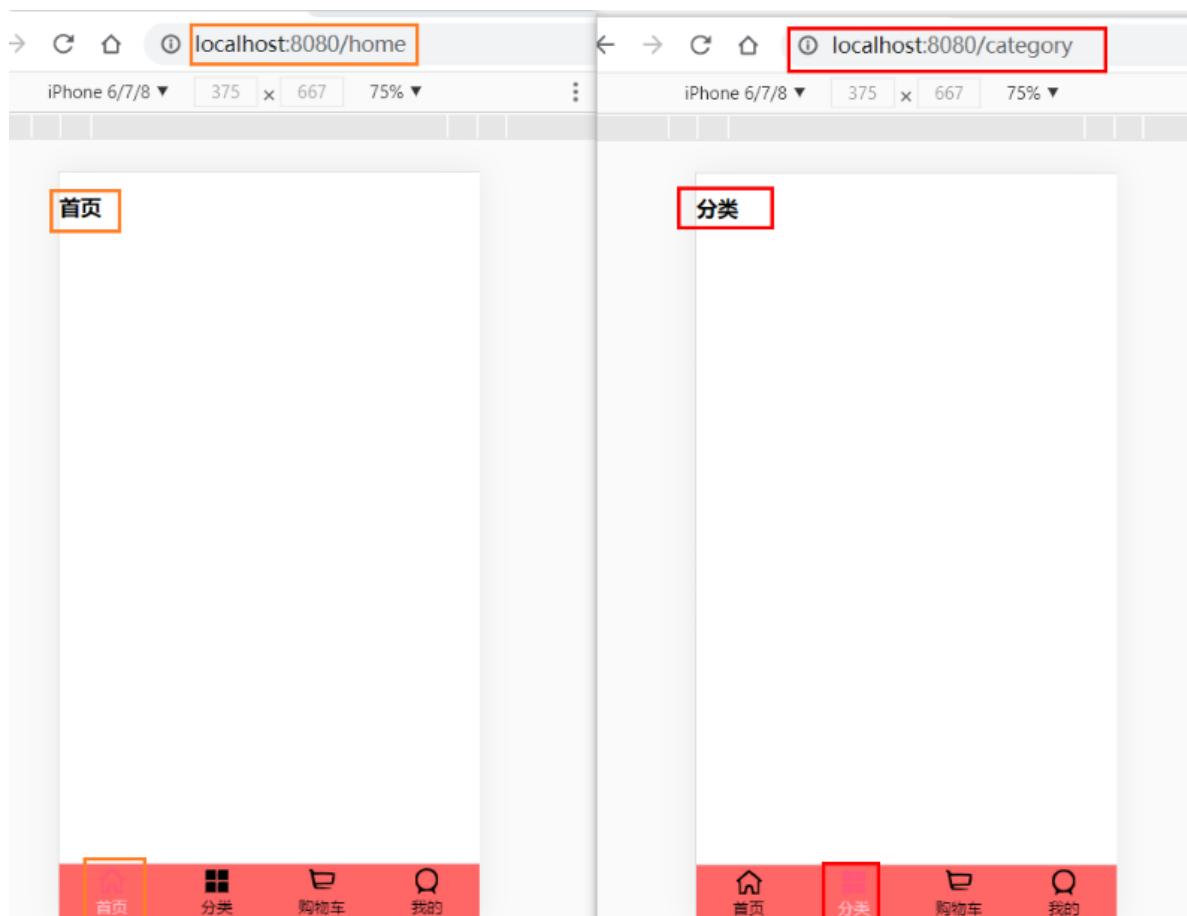
```
1 <template>
2   <h3>个人</h3>
3 </template>
4
5 <script>
6   export default {
7     name: "Profile"
8   }
9 </script>
10
11 <style scoped>
12
13 </style>
```

index.js

```
1 import Vue from 'vue'
2 import VueRouter from "vue-router";
3
4 const Home = () => import('views/home/Home')
5 const Category = () => import('views/category/Category')
6 const Cart = () => import('views/cart/Cart')
7 const Profile = () => import('views/profile/Profile')
8 // 1. 安装插件
9 Vue.use(VueRouter)
10 // 2. 创建路由对象
11 const routes = [
12   {
13     path: '',
14     redirect: '/home'
15   },
16   {
17     path: '/home',
18     component: Home
19   },
20   {
21     path: '/category',
22     component: Category
23 }
```

```
23 },
24 {
25   path: '/cart',
26   component: Cart
27 },
28 {
29   path: '/profile',
30   component: Profile
31 }
32 ]
33
34 const router = new VueRouter({
35   routes,
36   // 使用history
37   mode: 'history'
38 })
39
40 // 3. 导出router
41 export default router
```

15.3 执行结果



16-Vuex 详解

16.1 Vuex是做什么的?

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。

它采用 集中式存储管理 应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。

Vuex 也集成到 Vue 的官方调试工具 devtools extension，提供了诸如零配置的 time-travel 调试、状态快照导入导出等高级调试功能。

状态管理到底是什么?

可以简单的将其看成把需要多个组件共享的变量全部存储在一个对象里面。然后，将这个对象放在顶层的Vue实例中，让其他组件可以使用。

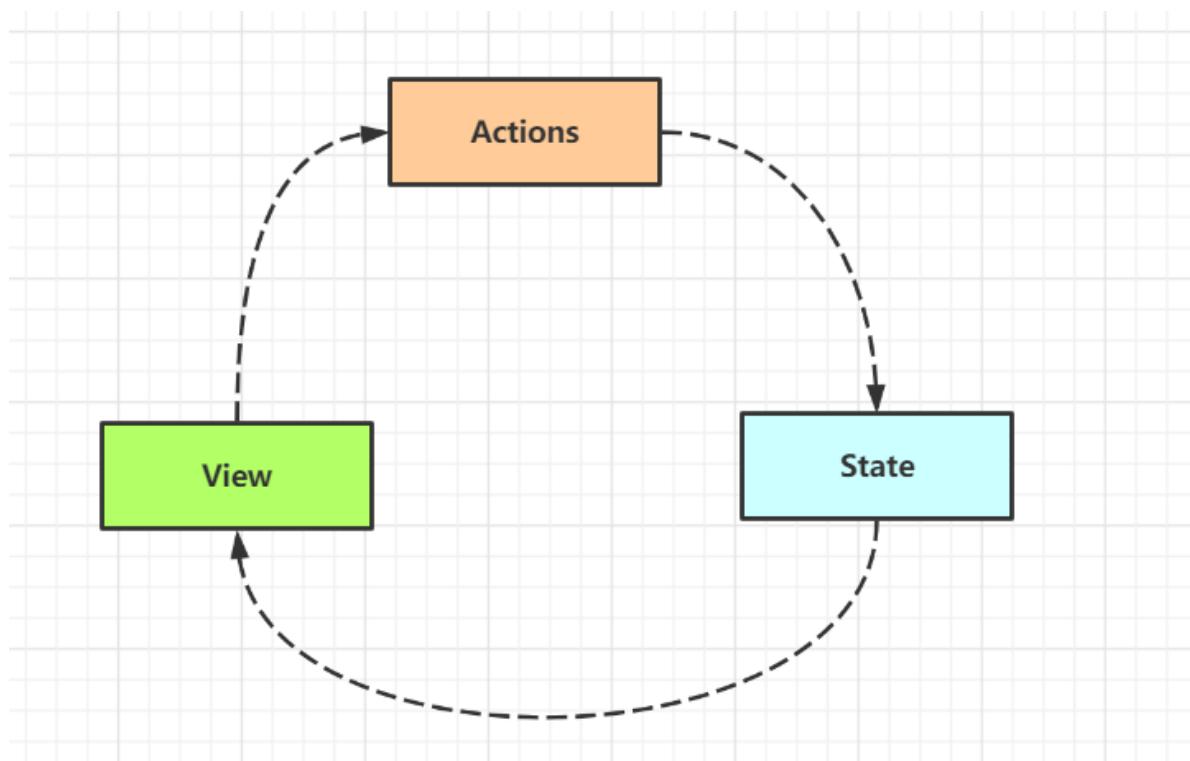
Vuex就是为了提供这样一个在多个组件间共享状态的插件。

管理什么状态呢?

如果你做过大型开放，一定遇到过多个状态，在多个界面间的共享问题。比如用户的登录状态、用户名称、头像、地理位置信息等等。比如商品的收藏、购物车中的物品等等。

这些状态信息，我们都可以放在统一的地方，对它进行保存和管理，而且它们还是响应式的。

16.2 单界面的状态管理



State：不用多说，就是我们的状态。（可以当做就是data中的属性）

View：视图层，可以针对State的变化，显示不同的信息。

Actions：这里的Actions主要是用户的各种操作：点击、输入等等，会导致状态的改变。

代码示例

counter需要某种方式被记录下来，也就是我们的State。

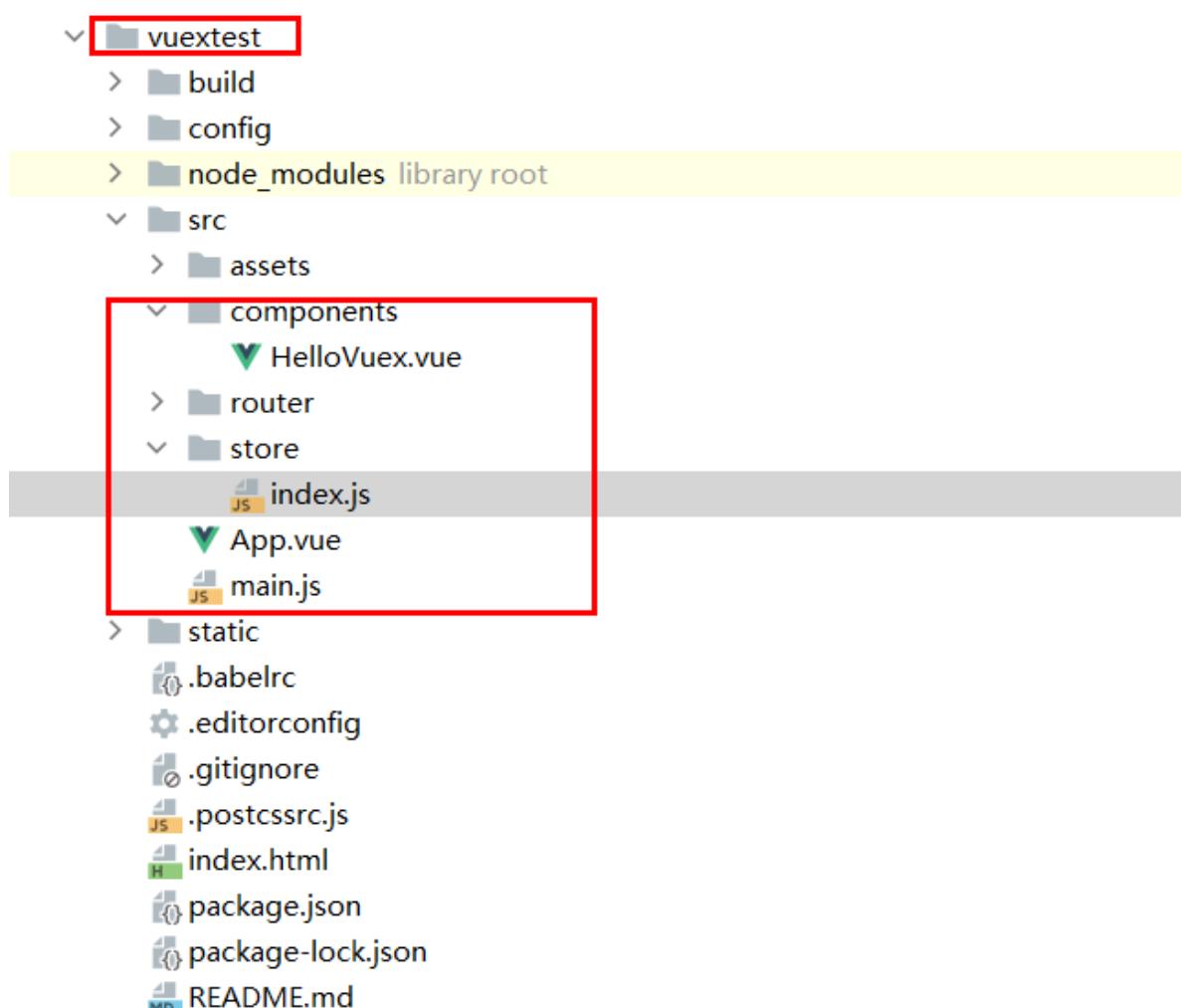
counter目前的值需要被显示在界面中，也就是View部分。

界面发生某些操作时（这里是用户的点击，也可以是用户的input），需要去更新状态，也就是我们的Actions

```
1 <template>
2   <div class="test">
3     <div>当前计数:{{counter}}</div>
4     <button @click="counter+=1">+1</button>
5     <button @click="counter-=1">-1</button>
6   </div>
7 </template>
8
9 <script>
10 export default {
11   name: "hello",
12   data(){
13     return {
14       counter:0
15     }
16   }
17 }
18 </script>
19
20 <style scoped>
21
22 </style>
```

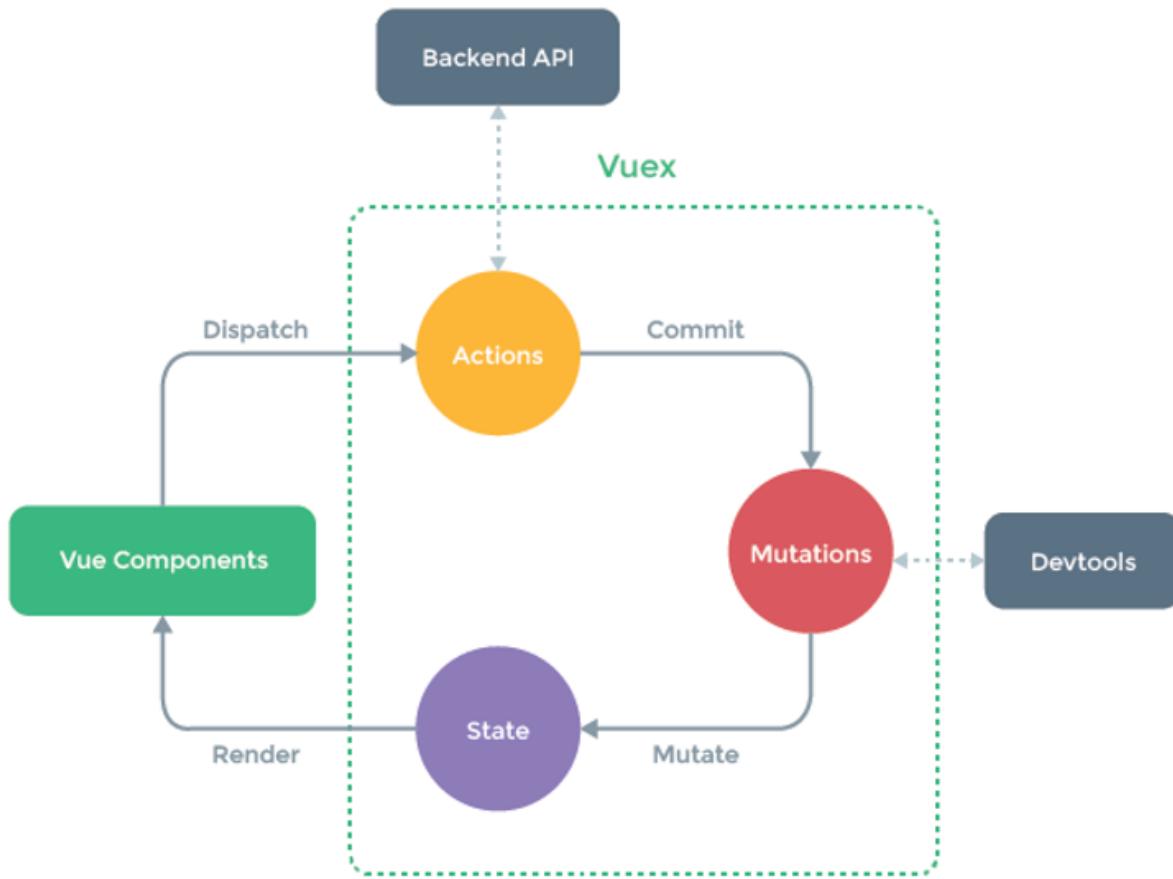
16.3 多界面状态管理

创建项目 vue init webpack vuetest



进入项目终端执行 `cnpm install vuex --save` 命令。

全局单例模式（大管家）



现在要做的就是将共享的状态抽取出来，交给我们的大管家，统一进行管理。

之后，每个试图，按照规定好的规定，进行访问和修改等操作。这就是Vuex背后的基本思想。

代码示例

先创建一个文件夹store，并且在其中创建一个index.js文件

index.js

```
1 // 导入vue
2 import Vue from 'vue'
3 import Vuex from 'vuex'

4 // 安装插件
5 Vue.use(Vuex)

6 // 创建对象
7 const store = new Vuex.Store({
8     // 放置状态相关的信息
9     state: {
10         counter: 1000
11     },
12     // 方法
13     mutations: {
14         increment(state){
15             state.counter++
16         },
17         decrement(state){
18             state.counter--
19         }
20     }
21 })
```

```
20     state.counter--
21   }
22 }
23 })
24
25 // 导出store
26 export default store
```

挂载到Vue实例中，导入store对象，并且放在new Vue中。

这样，在其他Vue组件中，就可以通过 `this.$store` 的方式，获取到这个store对象了。

main.js

```
1 import Vue from 'vue'
2 import App from './App'
3 import router from './router'
4 import store from "./store";
5
6 vue.config.productionTip = false
7
8 /* eslint-disable no-new */
9 new Vue({
10   el: '#app',
11   router,
12   store,
13   render: h => h(App)
14 })
```

HelloVuex.vue

```
1 <template>
2   <div>
3     <h3>{$store.state.counter}</h3>
4   </div>
5 </template>
6
7 <script>
8 export default {
9   name: "HelloVuex"
10 }
11 </script>
12
13 <style scoped>
14 </style>
```

APP.vue

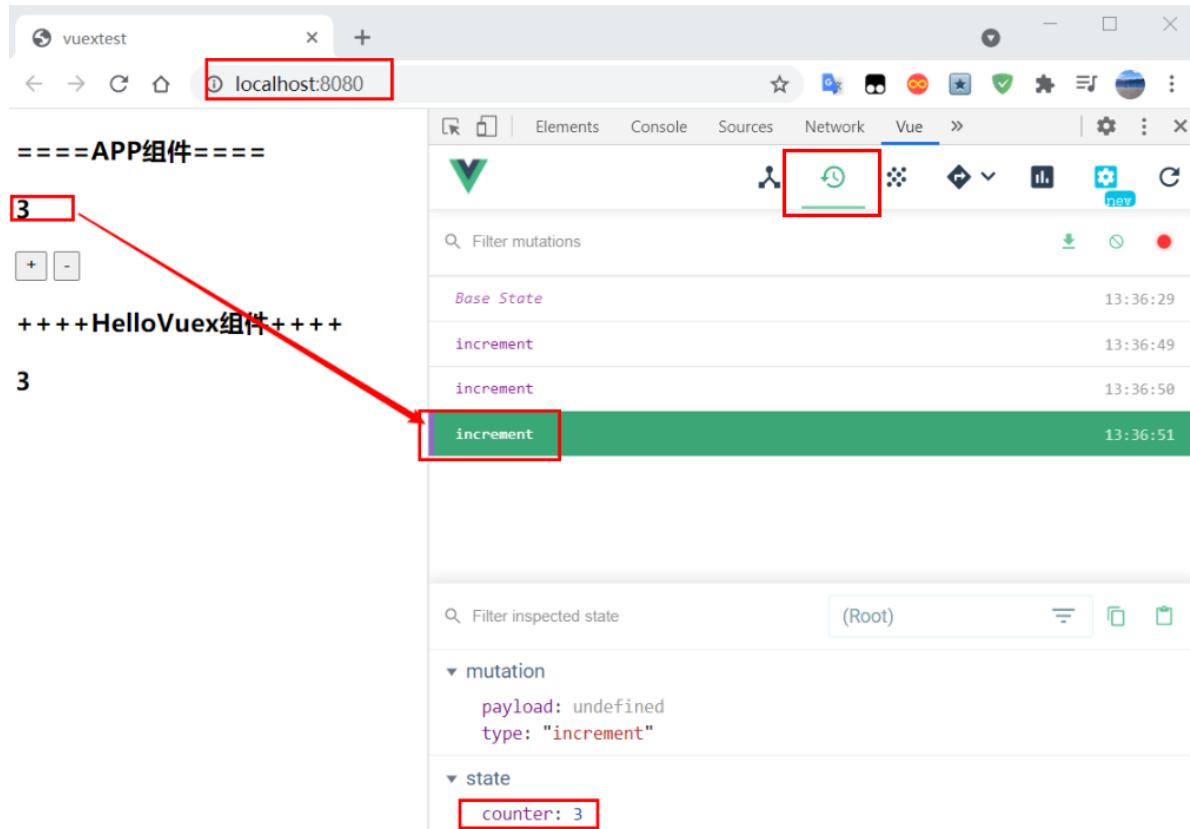
通过提交mutation的方式，而非直接改变`store.state.count`。

Vuex可以更明确的追踪状态的变化，所以不要直接改变`store.state.count`的值。

```
1 <template>
2   <div id="app">
3     <h3>====APP组件====</h3>
4     <h3>{$store.state.counter}</h3>
5     <button @click="addition">+</button>
```

```
6   <button @click="subtraction">-</button>
7
8   <h3>++++HelloVueX组件++++</h3>
9   <HelloVueX/>
10  </div>
11  </template>
12
13 <script>
14 import HelloVueX from "./components/HelloVueX";
15 export default {
16   name: 'App',
17   // 注册组件
18   components:{ 
19     HelloVueX
20   },
21   data(){
22     return{
23       message: 'hello vue.js!!!'
24     }
25   },
26   methods:{
27     addition(){
28       this.$store.commit('increment')
29     },
30     subtraction(){
31       this.$store.commit('decrement')
32     }
33   }
34 }
35 </script>
36
37 <style>
38 </style>
39
```

执行结果



16.4 Vuex核心概念

16.4.1 Getters基本使用

有时候，我们需要从store中获取一些state变异后的状态。

获取学生年龄大于20的个数

```

1 // 创建对象
2 const store = new Vuex.Store({
3   // 放置状态相关的信息
4   state: {
5     students: [
6       {id:1, name: 'guardwhy', age:18},
7       {id:2, name: 'kobe', age:41},
8       {id:3, name: 'James', age:35},
9       {id:4, name: 'Rondo', age:32}
10    ]
11  }
12})

```

可以在Store中定义getters

```

1 getters:{
2   arrayStu(state){
3     // 返回结果
4     return state.students.filter(s =>s.age > 20)
5   }
6 }

```

代码示例

index.js

```
1 // 导入vue
2 import Vue from 'vue'
3 import Vuex from 'vuex'
4
5 // 安装插件
6 Vue.use(Vuex)
7
8 // 创建对象
9 const store = new Vuex.Store({
10   // 放置状态相关的信息
11   state: {
12     students: [
13       {id:1, name: 'guardwhy', age:18},
14       {id:2, name: 'kobe', age:41},
15       {id:3, name: 'James', age:35},
16       {id:4, name: 'Rondo', age:32}
17     ],
18   },
19   getters: {
20     arrayStu(state) {
21       // 返回结果，获取学生年龄大于20的个数
22       return state.students.filter(s => s.age > 20)
23     },
24     arrayStuLength(state, getters) {
25       return getters.arrayStu.length
26     },
27     // 判断年龄
28     arrayStuAge(state) {
29       // return function (age) {
30       //   return state.students.filter(s => s.age > age)
31       // }
32       return age => {
33         return state.students.filter(s => s.age > age)
34       }
35     }
36   }
37 })
38
39 // 导出store
40 export default store
```

HelloVuex.vue

```
1 <template>
2   <div>
3     <h3>{{$store.state.counter}}</h3>
4     <h3>{{$store.getters.arrayStu}}</h3>
5
6   </div>
7 </template>
8
9 <script>
10 export default {
11   name: "HelloVuex"
12 }
```

```
13 </script>
14
15 <style scoped>
16
17 </style>
```

APP.vue

```
1 <template>
2   <div id="app">
3     <h3>++++App组件内容:getters相关信息++++</h3>
4     <h3>{{ $store.getters.arrayStu }}</h3>
5     <h3>{{ $store.getters.arrayStuLength }}</h3>
6     <h3>{{ $store.getters.arrayStuAge(33) }}</h3>
7
8     <h3>++++HelloVuex组件内容:getters相关信息++++++</h3>
9     <Hellovuex/>
10    </div>
11  </template>
12
13 <script>
14   import Hellovuex from "./components/Hellovuex";
15   export default {
16     name: 'App',
17     // 注册组件
18     components: {
19       Hellovuex
20     }
21   }
22 </script>
23
24 <style>
25 </style>
```

执行结果

The screenshot shows the Chrome DevTools interface with the 'Vue' tab selected. At the top, there's a URL bar with 'localhost:8080'. The main area displays the application's state and mutations.

Base State:

```
[ { "id": 2, "name": "kobe", "age": 41 }, { "id": 3, "name": "James", "age": 35 }, { "id": 4, "name": "Rondo", "age": 32 } ]
```

mutations:

```
[ { "id": 2, "name": "kobe", "age": 41 }, { "id": 3, "name": "James", "age": 35 } ]
```

getters:

```
[ { "id": 2, "name": "kobe", "age": 41 }, { "id": 3, "name": "James", "age": 35 }, { "id": 4, "name": "Rondo", "age": 32 } ]
```

The 'Vue' tab has a red box highlighting the 'mutations' and 'getters' sections. The 'mutations' section shows the array of student objects. The 'getters' section shows the same array, along with other getters like 'arrayStu', 'arrayStuLength', and 'arrayStuAge'.

16.4.2 Mutation基本使用

Vuex的store状态的更新唯一方式：提交Mutation。

Mutation主要包括两部分

一个是字符串的事件类型 (type) 。

一个是回调函数 (handler) ,该回调函数的第一个参数就是state。

mutation的定义

```
1 mutations:{  
2     increment(state){  
3         state.counter++  
4     },  
5     decrement(state){  
6         state.counter--  
7     }  
8 }
```

通过mutation更新

```
1 increment: function(){  
2     this.$store.commit('increment')  
3 }
```

Mutation传递参数

在通过mutation更新数据的时候, 有可能希望携带一些额外的参数, 参数被称为是mutation的载荷 (Payload)。

如果参数不是一个, 比如有很多参数需要传递, 通常会以对象的形式传递, 也就是payload是一个对象。这个时候可以再从对象中取出相关的信息。

index.js

```
1 // 导入vue  
2 import Vue from 'vue'  
3 import Vuex from 'vuex'  
4  
5 // 安装插件  
6 Vue.use(Vuex)  
7  
8 // 创建对象  
9 const store = new Vuex.Store({  
10     // 放置状态相关的信息  
11     state:{  
12         counter:10,  
13         students:[  
14             {id:1, name: 'guardwhy', age:18},  
15             {id:2, name: 'kobe', age:41},  
16             {id:3, name: 'James', age:35},  
17             {id:4, name: 'Rondo', age:32}  
18         ],  
19     },  
20     // 方法  
21     mutations:{
```

```

22     increment(state){
23         state.counter++
24     },
25     decrement(state){
26         state.counter--
27     },
28     // 普通提交方式: payload(负载)
29     incrementCount(state, count){
30         state.counter += count
31     },
32     addStudent(state, stu){
33         state.students.push(stu)
34     }
35 },
36 getters:{
37     arrayStu(state){
38         // 返回结果
39         return state.students.filter(s =>s.age > 20)
40     }
41 }
42 })
43
44 // 导出store
45 export default store

```

App.vue

```

1 <template>
2 <div id="app">
3     <h3>====APP组件内容=====</h3>
4     <h3>{{$store.state.counter}}</h3>
5     <button @click="addition">+</button>
6     <button @click="subtraction">-</button>
7     <button @click="addCount(5)">+5</button>
8     <button @click="addCount(10)">+10</button>
9     <button @click="addStudent">添加学生</button>
10
11
12     <h3>++++App组件内容:getters相关信息++++</h3>
13     <h3>{{$store.getters.arrayStu}}</h3>
14     <h3>++++Hellovuex组件内容:getters相关信息++++++++++</h3>
15     <Hellovuex/>
16 </div>
17 </template>
18
19 <script>
20 import Hellovuex from "./components/Hellovuex";
21 export default {
22     name: 'App',
23     // 注册组件
24     components:{
25         Hellovuex
26     },
27     methods:{
28         addition(){
29             this.$store.commit('increment')
30         },

```

```

31     subtraction(){
32         this.$store.commit('decrement')
33     },
34     addCount(count){
35         // 1.普通的提交封装
36         this.$store.commit('incrementCount', count)
37     },
38     addStudent(){
39         // 创建学生对象
40         const stu = {id: 5, name: 'curry', age:34}
41         this.$store.commit('addStudent', stu)
42     },
43 }
44 }
45 </script>
46 <style>
47 </style>

```

执行结果

=====APP组件内容=====

25

[+/-](#) [+5](#) [+10](#) [添加学生](#)

+++++App组件内容: getters相关信息+++++

[{ "id": 2, "name": "kobe", "age": 41 }, { "id": 3, "name": "James", "age": 35 }, { "id": 4, "name": "Rondo", "age": 32 }, { "id": 5, "name": "curry", "age": 34 }]

+++++HelloVuex组件内容: getters相关信息++++++

25

[{ "id": 2, "name": "kobe", "age": 41 }, { "id": 3, "name": "James", "age": 35 }, { "id": 4, "name": "Rondo", "age": 32 }, { "id": 5, "name": "curry", "age": 34 }]

16.4.3 Mutation提交风格

Vue还提供了另外一种风格, 它是一个包含type属性的对象。

index.js

```

22 increment(state){
23     state.counter++
24 },
25 decrement(state){
26     state.counter--
27 },
28 // 普通提交方式: payload(负载)
29 /*incrementCount(state, count){
30     state.counter += count
31 },*/
32 // payload提交方式
33 incrementCount(state, payload){
34     state.counter += payload.count
35 },
36 addStudent(state, stu){
37     state.students.push(stu)
38 }

```

App.vue

```

Project ▾
Vuex E:\workspace\Web\06-Vue.js\Vuex
  vuextest
    build
    config
  node_modules library root
    src
      assets
      components
        HelloVuex.vue
      store
        index.js
        App.vue
        main.js
      static
        .babelrc
        .editorconfig
        .gitignore
        .postcssrc.js
        index.html
        package.json
        package-lock.json
        README.md
External Libraries
Scratches and Consoles

```

```

  33 },
  34   },
  35   },
  36   },
  37     addition(){
  38       this.$store.commit( type: 'increment' )
  39     },
  40     subtraction(){
  41       this.$store.commit( type: 'decrement' )
  42     },
  43     addCount(count){
  44       // 1. 普通的提交封装
  45       // this.$store.commit('incrementCount', count)
  46
  47       // 2. 特殊的提交封装
  48       this.$store.commit( payloadWithType: {
  49         type: 'incrementCount',
  50         count
  51       })
  
```

执行结果

====APP组件内容=====

30

+ + + + + App组件内容: getters相关信息 + + + + +

[{ "id": 2, "name": "kobe", "age": 41 }, { "id": 3, "name": "James", "age": 35 }, { "id": 4, "name": "Rondo", "age": 32 }, { "id": 5, "name": "curry", "age": 34 }]

+ + + + + HelloVuex组件内容: getters相关信息 + + + + + + + + +

30

[{ "id": 2, "name": "kobe", "age": 41 }, { "id": 3, "name": "James", "age": 35 }, { "id": 4, "name": "Rondo", "age": 32 }, { "id": 5, "name": "curry", "age": 34 }]

Mutation响应规则

Vuex的store中的state是响应式的，当state中的数据发生改变时，Vue组件会自动更新。

这就要求必须遵守一些Vuex对应的规则，**提前在store中初始化好所需的属性**。

index.js

这些属性都会被加入到响应式系统中，而响应式系统会监听属性的变化，当属性发生变化时，会通知所有界面中的用到该属性的地方，让界面发生刷新。

```

1 // 导入vue
2 import Vue from 'vue'
3 import Vuex from 'vuex'
4
5 // 安装插件
6 Vue.use(Vuex)
7
8 // 创建对象
9 const store = new Vuex.Store({
10   // 放置状态相关的信息
11   state: {
12     info: {
13       name: 'harden',
14       age: 31,
15     }
16   }
17 }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
7010
7011
7012
7013
7014
7015
7016
7017
7018
7019
7020
7021
7022
7023
7024
7025
7026
7027
7028
7029
70210
70211
70212
70213
70214
70215
70216
70217
70218
70219
70220
70221
70222
70223
70224
70225
70226
70227
70228
70229
70230
70231
70232
70233
70234
70235
70236
70237
70238
70239
70240
70241
70242
70243
70244
70245
70246
70247
70248
70249
70250
70251
70252
70253
70254
70255
70256
70257
70258
70259
702510
702511
702512
702513
702514
702515
702516
702517
702518
702519
702520
702521
702522
702523
702524
702525
702526
702527
702528
702529
702530
702531
702532
702533
702534
702535
702536
702537
702538
702539
702540
702541
702542
702543
702544
702545
702546
702547
702548
702549
702550
702551
702552
702553
702554
702555
702556
702557
702558
702559
702560
702561
702562
702563
702564
702565
702566
702567
702568
702569
702570
702571
702572
702573
702574
702575
702576
702577
702578
702579
702580
702581
702582
702583
702584
702585
702586
702587
702588
702589
7025810
7025811
7025812
7025813
7025814
7025815
7025816
7025817
7025818
7025819
7025820
7025821
7025822
7025823
7025824
7025825
7025826
7025827
7025828
7025829
70258210
70258211
70258212
70258213
70258214
70258215
70258216
70258217
70258218
70258219
70258220
70258221
70258222
70258223
70258224
70258225
70258226
70258227
70258228
70258229
70258230
70258231
70258232
70258233
70258234
70258235
70258236
70258237
70258238
70258239
702582310
702582511
702582512
702582513
702582514
702582515
702582516
702582517
702582518
702582519
702582520
702582521
702582522
702582523
702582524
702582525
702582526
702582527
702582528
702582529
702582530
702582531
702582532
702582533
702582534
702582535
702582536
702582537
702582538
702582539
702582540
702582541
702582542
702582543
702582544
702582545
702582546
702582547
702582548
702582549
702582550
702582551
702582552
702582553
702582554
702582555
702582556
702582557
702582558
702582559
702582560
702582561
702582562
702582563
702582564
702582565
702582566
702582567
702582568
702582569
702582570
702582571
702582572
702582573
702582574
702582575
702582576
702582577
702582578
702582579
702582580
702582581
702582582
702582583
702582584
702582585
702582586
702582587
702582588
702582589
7025825810
7025825811
7025825812
7025825813
7025825814
7025825815
7025825816
7025825817
7025825818
7025825819
7025825820
7025825821
7025825822
7025825823
7025825824
7025825825
7025825826
7025825827
7025825828
7025825829
70258258210
70258258211
70258258212
70258258213
70258258214
70258258215
70258258216
70258258217
70258258218
70258258219
70258258220
70258258221
70258258222
70258258223
70258258224
70258258225
70258258226
70258258227
70258258228
70258258229
70258258230
70258258231
70258258232
70258258233
70258258234
70258258235
70258258236
70258258237
70258258238
70258258239
70258258240
70258258241
70258258242
70258258243
70258258244
70258258245
70258258246
70258258247
70258258248
70258258249
70258258250
70258258251
70258258252
70258258253
70258258254
70258258255
70258258256
70258258257
70258258258
70258258259
702582582510
702582582511
702582582512
702582582513
702582582514
702582582515
702582582516
702582582517
702582582518
702582582519
702582582520
702582582521
702582582522
702582582523
702582582524
702582582525
702582582526
702582582527
702582582528
702582582529
702582582530
702582582531
702582582532
702582582533
702582582534
702582582535
702582582536
702582582537
702582582538
702582582539
702582582540
702582582541
702582582542
702582582543
702582582544
702582582545
702582582546
702582582547
702582582548
702582582549
702582582550
702582582551
702582582552
702582582553
702582582554
702582582555
702582582556
702582582557
702582582558
702582582559
702582582560
702582582561
702582582562
702582582563
702582582564
702582582565
702582582566
702582582567
702582582568
702582582569
702582582570
702582582571
702582582572
702582582573
702582582574
702582582575
702582582576
702582582577
702582582578
702582582579
702582582580
702582582581
702582582582
702582582583
702582582584
702582582585
702582582586
702582582587
702582582588
702582582589
7025825825810
7025825825811
7025825825812
7025825825813
7025825825814
7025825825815
7025825825816
7025825825817
7025825825818
7025825825819
7025825825820
7025825825821
7025825825822
7025825825823
7025825825824
7025825825825
7025825825826
7025825825827
7025825825828
7025825825829
7025825825830
7025825825831
7025825825832
7025825825833
7025825825834
7025825825835
7025825825836
7025825825837
7025825825838
7025825825839
7025825825840
7025825825841
7025825825842
7025825825843
7025825825844
7025825825845
7025825825846
7025825825847
7025825825848
7025825825849
7025825825850
7025825825851
7025825825852
7025825825853
7025825825854
7025825825855
7025825825856
7025825825857
7025825825858
7025825825859
7025825825860
7025825825861
7025825825862
7025825825863
7025825825864
7025825825865
7025825825866
7025825825867
7025825825868
7025825825869
7025825825870
7025825825871
7025825825872
7025825825873
7025825825874
7025825825875
7025825825876
7025825825877
7025825825878
7025825825879
7025825825880
7025825825881
7025825825882
7025825825883
7025825825884
7025825825885
7025825825886
7025825825887
7025825825888
7025825825889
7025825825890
7025825825891
7025825825892
7025825825893
7025825825894
7025825825895
7025825825896
7025825825897
7025825825898
7025825825899
70258258258100
70258258258110
70258258258120
70258258258130
70258258258140
70258258258150
70258258258160
70258258258170
70258258258180
70258258258190
70258258258200
70258258258210
70258258258220
70258258258230
70258258258240
70258258258250
70258258258260
70258258258270
70258258258280
70258258258290
70258258258300
70258258258310
70258258258320
70258258258330
70258258258340
70258258258350
70258258258360
70258258258370
70258258258380
70258258258390
70258258258400
70258258258410
70258258258420
70258258258430
70258258258440
70258258258450
70258258258460
70258258258470
70258258258480
70258258258490
70258258258500
70258258258510
70258258258520
70258258258530
70258258258540
70258258258550
70258258258560
70258258258570
70258258258580
70258258258590
70258258258600
70258258258610
70258258258620
70258258258630
70258258258640
70258258258650
70258258258660
70258258258670
70258258258680
70258258258690
70258258258700
70258258258710
70258258258720
70258258258730
70258258258740
70258258258750
70258258258760
70258258258770
70258258258780
70258258258790
70258258258800
70258258258810
70258258258820
70258258258830
70258258258840
70258258258850
70258258258860
70258258258870
70258258258880
70258258258890
70258258258900
70258258258910
70258258258920
70258258258930
70258258258940
70258258258950
70258258258960
70258258258970
70258258258980
70258258258990
702582582581000
702582582581100
702582582581200
702582582581300
702582582581400
702582582581500
702582582581600
702582582581700
702582582581800
702582582581900
702582582582000
702582582582100
702582582582200
702582582582300
702582582582400
702582582582500
702582582582600
702582582582700
702582582582800
702582582582900
702582582583000
702582582583100
702582582583200
702582582583300
702582582583400
702582582583500
702582582583600
702582582583700
702582582583800
702582582583900
702582582584000
702582582584100
702582582584200
702582582584300
702582582584400
702582582584500
702582582584600
702582582584700
702582582584800
7
```

```

15     height: 1.93
16   }
17 },
18 // 方法
19 mutations:{
20   updateInfo(state){
21     // state.info.name = 'guardwhy'
22     /*
23      * 这方式是无法做到响应式的
24      * state.info['address'] = '广州'
25      * delete state.info.age
26      */
27     Vue.set(state.info, 'address', '广州')
28     // 删除age
29     Vue.delete(state.info, 'age')
30   }
31 }
32 })
33
34 // 导出store
35 export default store

```

App.vue

```

1 <template>
2   <div id="app">
3     <h3>-----App内容:Info对象是否是响应式-----</h3>
4     <h3>{{store.state.info}}</h3>
5     <button @click="updateInfo">修改信息</button>
6   </div>
7 </template>
8
9 <script>
10 import Hellovuex from "./components/Hellovuex";
11 export default {
12   name: 'App',
13   // 注册组件
14   components: {
15     Hellovuex
16   },
17   methods: {
18     updateInfo(){
19       this.$store.commit('updateInfo')
20     }
21   }
22 }
23 </script>
24 <style>
25 </style>

```

修改信息前

-----App内容:Info对象是否是响应式-----

```
{ "name": "harden", "age": 31, "height": 1.93 }
```

```
Base State 23:07:13
state
  counter: 10
  ▼ info: Object
    age: 31
    height: 1.93
    name: "harden"
  ▼ students: Array[4]
    ▶ 0: Object
    ▶ 1: Object
    ▶ 2: Object
    ▶ 3: Object
```

修改信息响应后

-----App内容:Info对象是否是响应式-----

```
{ "name": "harden", "height": 1.93, "address": "广州" }
```

```
Base State 23:07:13
mutation
  updateInfo 23:08:21
state
  counter: 10
  ▼ info: Object
    address: "广州"
    height: 1.93
    name: "harden"
  ▼ students: Array[4]
```

16.4.4 Action的基本使用

不要再Mutation中进行异步操作，Action类似于Mutation,但是是用来代替Mutation进行异步操作的。

context是什么？

context是和store对象具有相同方法和属性的对象。

也就是说,可以通过context去进行commit相关的操作,也可以获取context.state等。

index.js

```

1 // 导入vue
2 import Vue from 'vue'
3 import Vuex from 'vuex'
4
5 // 安装插件
6 Vue.use(Vuex)
7
8 // 创建对象
9 const store = new Vuex.Store({
10   // 放置状态相关的信息
11   state: {
12     info: {
13       name: 'harden',
14       age: 31,
15       height: 1.93
16     }
17   },
18   // 方法

```

```

19  mutations:{
20    updateInfo(state){
21      Vue.set(state.info, 'address', '广州')
22      // 删除age
23      Vue.delete(state.info, 'age')
24    }
25  },
26  getters:{
27    arraystu(state){
28      // 返回结果
29      return state.students.filter(s =>s.age > 20)
30    }
31  },
32  actions:{
33    // 方式1
34    /*
35     aupdateInfo(context, payload){
36       setTimeout(()=>{
37         context.commit('updateInfo')
38         console.log(payload.message);
39         payload.success()
40       },1000)
41     }
42   */
43
44    // 方式二
45    aupdateInfo(context, payload){
46      return new Promise((resolve, reject) =>{
47        setTimeout(()=>{
48          context.commit('updateInfo');
49          // 打印结果
50          console.log(payload)
51          resolve('测试成功! ! ! ')
52        },1000)
53      })
54    }
55  }
56})
57
58 // 导出store
59 export default store

```

App.vue

```

1 <template>
2   <div id="app">
3     <h3>-----App内容:Info对象是否是响应式-----</h3>
4     <h3>{{$store.state.info}}</h3>
5     <button @click="updateInfo">修改信息</button>
6   </div>
7 </template>
8
9 <script>
10 import Hellovuex from "./components/Hellovuex";
11 export default {
12   name: 'App',
13   // 注册组件

```

```

14  components:{ 
15    Hellovuex
16  },
17  methods:{ 
18    updateInfo(){
19      // this.$store.commit('updateInfo')
20
21      // 方式1: 异步操作
22      /*
23        this.$store.commit('updateInfo')
24        this.$store.dispatch('aUpdateInfo',{
25          message: '携带的信息！！',
26          success:() =>{
27            console.log('里面已经完成了');
28          }
29        })
29      */
30      // 方式二: 异步操作
31      this.$store
32        .dispatch('aUpdateInfo', '携带的信息')
33        .then(res =>{
34          console.log('里面完成了提交操作')
35          console.log(res)
36        })
37      })
38    }
39  }
40}
41</script>
42
43<style>
44</style>

```

执行结果

The screenshot illustrates the execution results of a Vue.js application. At the top, a browser window titled "vuextest" shows the URL "localhost:8080". The page content includes the text "Hellovuex" and a JSON object: { "name": "harden", "height": 1.93, "address": "广州" }. Below this is a button labeled "修改信息". To the right, the Vue DevTools interface is open, showing the "Console" tab with the following logs:

- [HMR] Waiting for update signal from WDS...
- vue-devtools Detected Vue v2.6.12
- 携带的信息
- 里面完成了提交操作
- 测试成功！！！

The "Console" tab also shows the mutation log:

Mutation	Time
Base State	00:44:49
updateInfo	00:45:34
updateInfo	00:45:35

Below the mutations, the state is displayed:

Path	Value
counter	10
info	Object { address: "广州", height: 1.93, name: "harden" }

16.4.5 认识Module

为什么在Vuex中我们要使用模块呢？

- Vue使用单一状态树，那么也意味着很多状态都会交给Vuex来管理。
- 当应用变得非常复杂时，store对象就有可能变得相当臃肿。
- 为了解决这个问题，Vuex允许我们将store分割成模块(Module)，而每个模块拥有自己的states、mutations、actions、getters等。

代码示例

index.js

```
1 // 导入vue
2 import Vue from 'vue'
3 import Vuex from 'vuex'
4
5 // 安装插件
6 Vue.use(Vuex)
7
8 // 创建module对象
9 const moduleA = {
10   state: {
11     name: 'Duncan'
12   },
13   mutations: {
14     updateName(state, payload){
15       state.name = payload
16     }
17   },
18   getters: {
19     onename(state) {
20       return state.name + ' good'
21     },
22     secondname(state, getters) {
23       return getters.onename + ', study'
24     },
25     threename(state, getters, rootState) {
26       return getters.secondname + rootState.counter
27     }
28   },
29   actions: {
30     aupdateName(context){
31       // 打印上下文
32       console.log(context);
33       setTimeout(() => {
34         context.commit('updateName', 'Durant')
35       }, 1000)
36     }
37   }
38 }
39
40 // 创建对象
41 const store = new Vuex.Store({
42   // 放置状态相关的信息
43   state: {
44     counter: 10
45   },
46
47 }
```

```
46     modules: {
47       a: moduleA
48     }
49   })
50
51 // 导出store
52 export default store
```

App.vue

```
1 <template>
2   <div id="app">
3     <h3>-----App内容: modules中的内容-----</h3>
4     <h3>{{$store.state.a.name}}</h3>
5     <button @click="updateName">修改名字</button>
6     <h3>{{$store.getters.onename}}</h3>
7     <h3>{{$store.getters.secondname}}</h3>
8     <h3>{{$store.getters.threename}}</h3>
9     <button @click="asyncUpdateName">异步修改名字</button>
10    </div>
11  </template>
12
13 <script>
14 import Hellovuex from "./components/Hellovuex";
15 export default {
16   name: 'App',
17   // 注册组件
18   components: {
19     Hellovuex
20   },
21   methods: {
22     updateName(){
23       this.$store.commit('updateName', 'jordan')
24     },
25     asyncUpdateName(){
26       this.$store.dispatch('aupdateName')
27     }
28   }
29 }
30 </script>
31 <style>
32 </style>
```

执行结果

The screenshot shows the Vue DevTools Dev tab with the title "App内容: modules中的内容". On the left, there's a list of names: "jordan", "jordangood", "jordangood, study", and "jordangood, study10". A red box highlights the "修改名字" button next to "jordan". On the right, the mutation log shows a single entry: "updateName" at 02:05:43, with payload "jordan" and type "updateName".

This screenshot is similar to the one above, but the list of names has changed: "Durant,", "Durant, good", "Durant, good, study", and "Durant, good, study10". A red box highlights the "修改名字" button next to "Durant,". The mutation log shows the same "updateName" entry at 02:05:44, with payload "Durant," and type "updateName".

17 - 网络模块封装

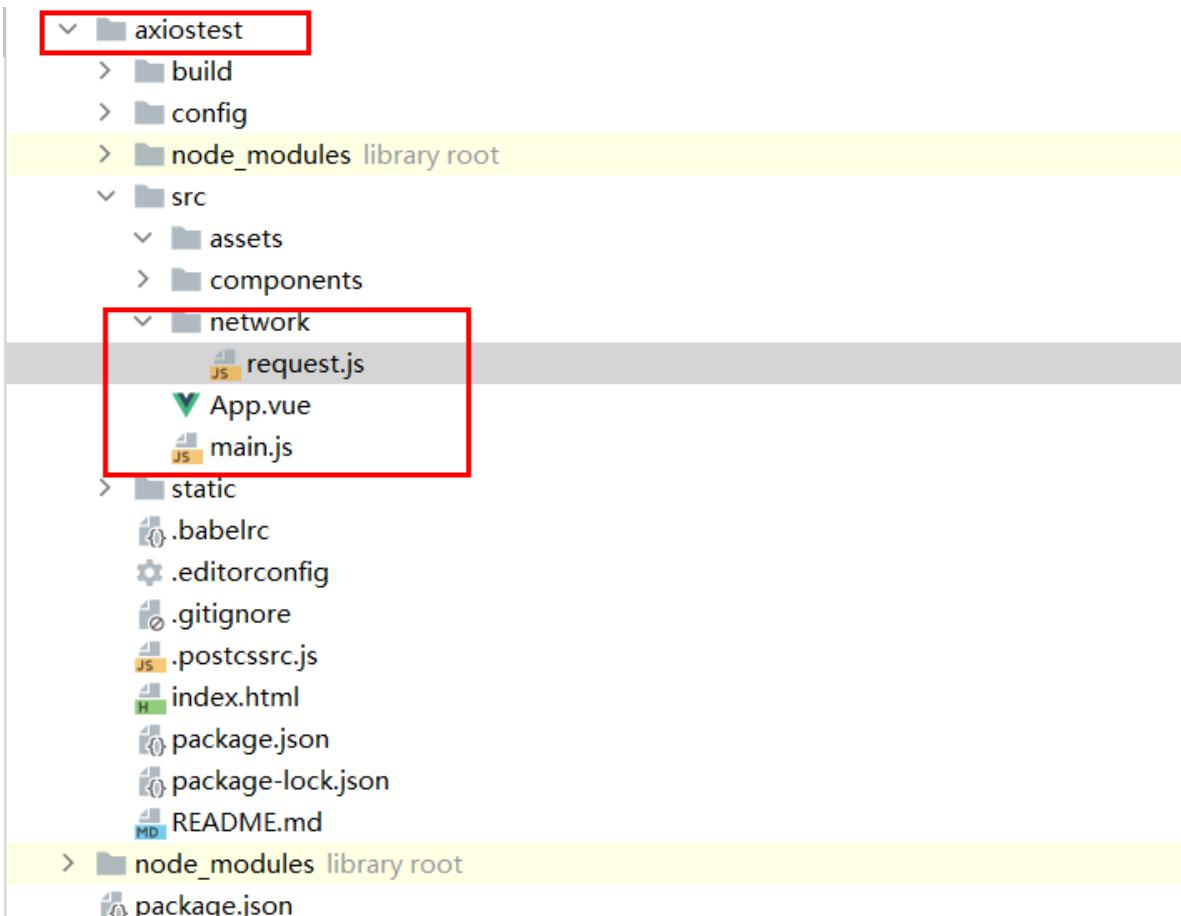
17.1 为什么选择axios?

axios 是一个基于 promise 的 HTTP 库网络请求插件。

基本特点

- 可以用在浏览器(测试网站: httpbin.org/)和 node.js 中
- 支持 Promise API。
- 自动转换 JSON 数据。
- 客户端支持防御 XSRF。

创建项目: `vue init webpack axiostest`, 终端安装插件: `cnpm install axios --save`



17.2 axios基本使用

App.vue

```
1 <template>
2   <div id="app">
3     </div>
4   </template>
5
6 <script>
7   export default {
8     name: 'App'
9   }
10 </script>
11
12 <style>
13   </style>
```

main.js

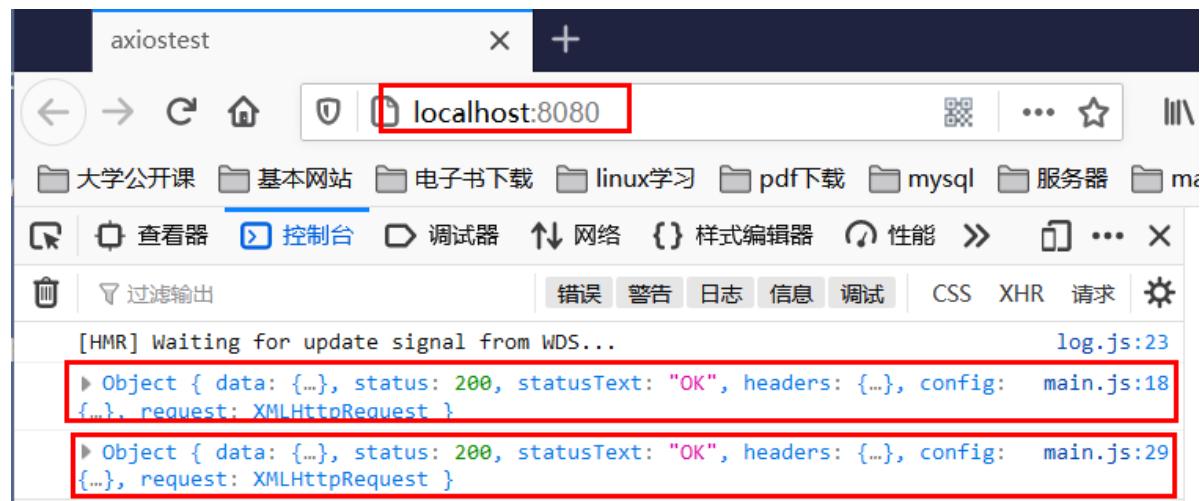
```
1 import vue from 'vue'
2 import App from './App'
3 // 导入axios
4 import axios from 'axios'
5
6 vue.config.productionTip = false
7
8 /* eslint-disable no-new */
9 new vue({
```

```

10    el: '#app',
11    render: h => h(App)
12  })
13 // 1. 基本使用
14 axios({
15   url: 'http://123.207.32.32:8000/home/multidata',
16   method: 'get'
17 }).then(res => {
18   console.log(res)
19 })
20
21 axios({
22   url: 'http://123.207.32.32:8000/home/data',
23   // 专门用来对get请求参数的拼接
24   params: {
25     type: 'pop',
26     page: 1
27   }
28 }).then(res => {
29   console.log(res);
30 })

```

执行结果



17.3 axios发送并发请求

使用axios.all可以放入多个请求的数组，axios.all([]) 返回的结果是一个数组。

main.js

```

1 import Vue from 'vue'
2 import App from './App'
3 // 导入axios
4 import axios from 'axios'
5
6 Vue.config.productionTip = false
7
8 /* eslint-disable no-new */
9 new Vue({
10   el: '#app',
11   render: h => h(App)
12 })
13 // 1. axios发送并发请求

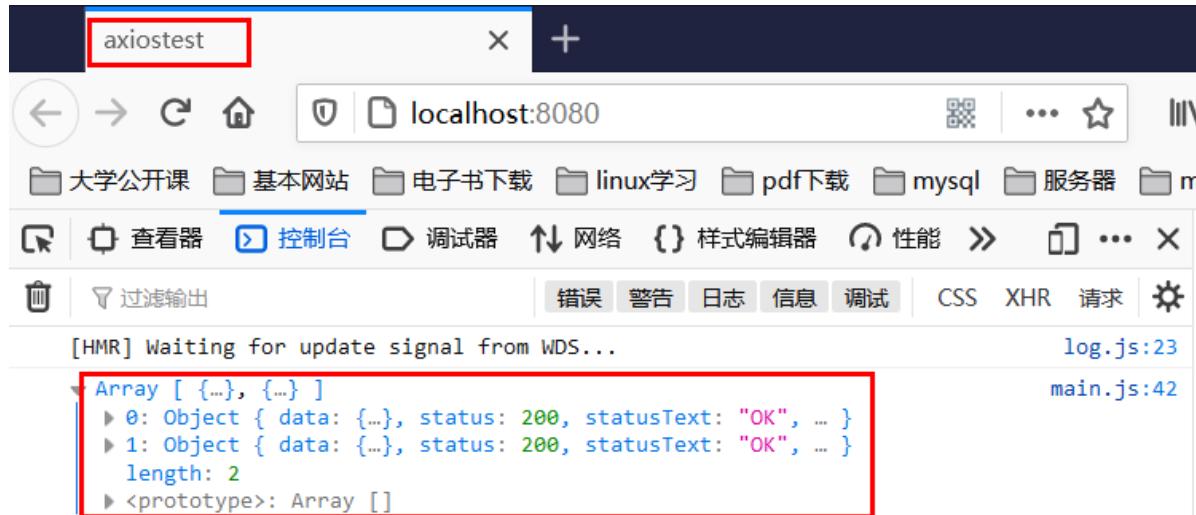
```

```

14 axios.all([axios({
15   url: 'http://123.207.32.32:8000/home/multidata'
16 }), axios({
17   url: 'http://123.207.32.32:8000/home/data',
18   params: {
19     type: 'sell',
20     page: 5
21   }
22 })]).then(results => {
23   console.log(results);
24 })

```

执行结果



使用 `axios.spread` 可将数组 [res1,res2] 展开为 res1, res2。

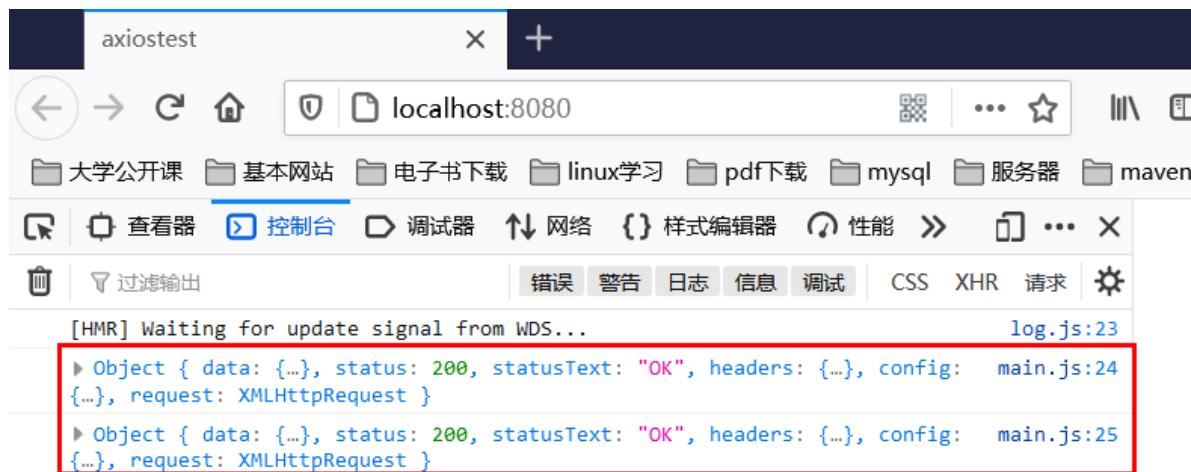
main.js

```

1 import Vue from 'vue'
2 import App from './App'
3 // 导入axios
4 import axios from 'axios'
5
6 Vue.config.productionTip = false
7
8 /* eslint-disable no-new */
9 new Vue({
10   el: '#app',
11   render: h => h(App)
12 })
13
14 // 2.axios发送并发请求
15 axios.all([axios({
16   url: 'http://123.207.32.32:8000/home/multidata'
17 }), axios({
18   url: 'http://123.207.32.32:8000/home/data',
19   params: {
20     type: 'sell',
21     page: 5
22   }
23 })]).then(axios.spread((res1, res2) =>{
24   console.log(res1);
25   console.log(res2);

```

执行结果



17.4 axios的实例

为什么要创建axios的实例呢？

- 当从axios模块中导入对象时，使用的实例是默认的实例。
- 当给该实例设置一些默认配置时，这些配置就被固定下来了。
- 但是后续开发中，某些配置可能会不太一样，比如某些请求需要使用特定的baseURL或者timeout或者content-Type等。
- 这个时候，我们就可以创建新的实例，并且传入属于该实例的配置信息。

```

1 import Vue from 'vue'
2 import App from './App'
3 // 导入axios
4 import axios from 'axios'
5
6 Vue.config.productionTip = false
7
8 /* eslint-disable no-new */
9 new Vue({
10   el: '#app',
11   render: h => h(App)
12 })
13
14 // 4. 创建对应的axios实例
15 const instance1 = axios.create({
16   baseURL: 'http://123.207.32.32:8000',
17   timeout: 5000
18 })
19 instance1({
20   url: '/home/multidata'
21 }).then(res => {
22   console.log(res);
23 })
24 instance1({
25   url: '/home/data',
26   params: {
27     type: 'pop',
28     page: 1
29   }
30 })

```

```
30 }) .then(res => {
31   console.log(res);
32 })
33
34 const instance2 = axios.create({
35   baseURL: 'http://222.111.33.33:8000',
36   timeout: 10000,
37   headers: {}
38 })
```

17.5 axios封装

方式一(回调函数)

request.js

```
1 // 导入axios
2 import axios from 'axios'
3
4 export function request(config, success, failure){
5   // 1. 创建axios的实例
6   const instance = axios.create({
7     baseURL: 'http://123.207.32.32:8000',
8     timeout: 5000
9   })
10
11   // 发送真正的网络请求
12   instance(config)
13     .then(res =>{
14       success(res)
15     })
16     .catch(err =>{
17       failure(err)
18     })
19 }
```

main.js

```
1 import Vue from 'vue'
2 import App from './App'
3
4 Vue.config.productionTip = false
5
6 /* eslint-disable no-new */
7 new Vue({
8   el: '#app',
9   render: h => h(App)
10 })
11 // 1. 封装request模块
12 import { request } from "./network/request";
13
14 request({
15   url: '/home/multidata'
16 }, res =>{
17   console.log(res);
18 }, err =>{
```

```
19     console.log(err);
20 })
```

执行结果

A screenshot of a browser developer tools window titled "axiostest". The address bar shows "localhost:8080". The Network tab is selected. A single network request is listed:

```
[HMR] Waiting for update signal from WDS... log.js:23
▶ Object { data: {...}, status: 200, statusText: "OK", headers: {...}, config: main.js:34
..., request: XMLHttpRequest }
```

The status is 200 OK.

方式二(Promise函数)

request.js

```
1 // 导入axios
2 import axios from 'axios'
3
4 export function request(config) {
5     return new Promise((resolve, reject) => {
6         // 1. 创建axios的实例
7         const instance = axios.create({
8             baseURL: 'http://123.207.32.32:8000',
9             timeout: 5000
10        })
11        // 发送真正的网络请求
12        instance(config)
13            .then(res => {
14                resolve(res)
15            })
16            .catch(err => {
17                reject(err)
18            })
19    })
20 }
```

main.js

```
1 import Vue from 'vue'
2 import App from './App'
3
4 Vue.config.productionTip = false
5
6 /* eslint-disable no-new */
7 new Vue({
8     el: '#app',
9     render: h => h(App)
10})
11
12 // 5. 封装request模块
```

```
13 import {request} from "./network/request";
14
15 request({
16   url: '/home/multidata'
17 }).then(res => {
18   console.log(res);
19 }).catch(err => {
20   console.log(err);
21 })
```

最终方案

request.js

```
1 // 导入axios
2 import axios from 'axios'
3
4 export function request(config) {
5   // 1. 创建axios实例
6   const instance = axios.create({
7     baseURL: 'http://123.207.32.32:8000',
8     timeout: 5000
9   })
10  // 2. 发送真正的网络请求
11  return instance(config)
12 }
```

17.6 如何使用拦截器?

axios提供了拦截器，用于我们在发送每次请求或者得到相应后，进行对应的处理。

响应的成功拦截中，主要是对数据进行过滤。

代码示例

request.js

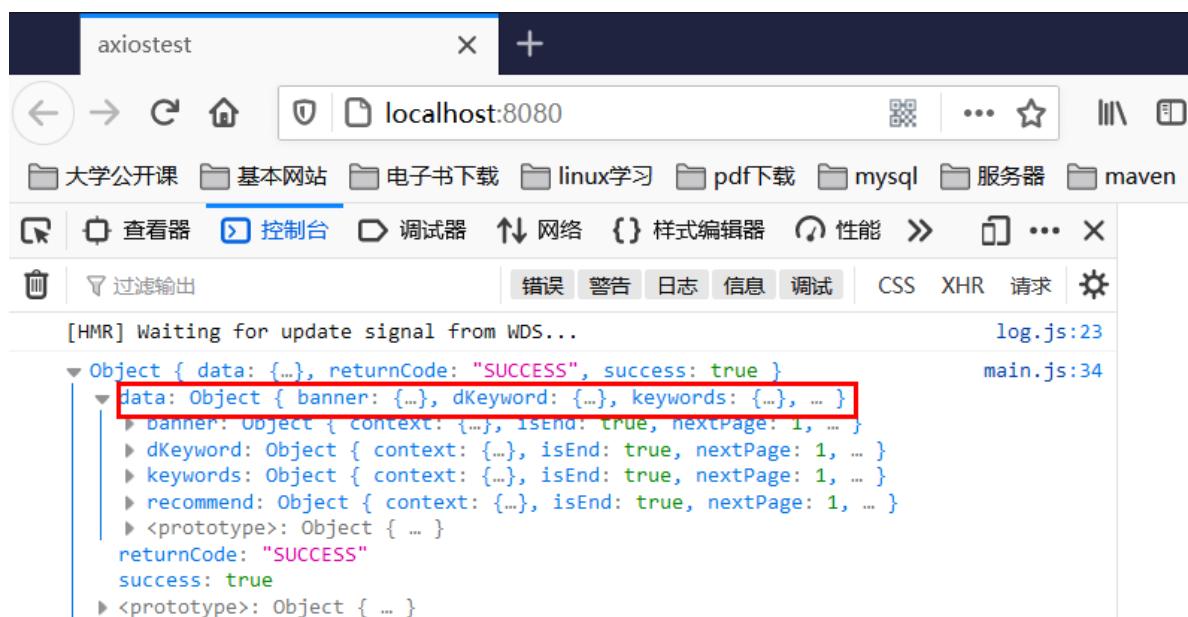
```
1 // 导入axios
2 import axios from 'axios'
3
4 export function request(config) {
5   // 1. 创建axios实例
6   const instance = axios.create({
7     baseURL: 'http://123.207.32.32:8000',
8     timeout: 5000
9   })
10
11  // 2. axios拦截器(请求拦截)
12  instance.interceptors.request.use(config => {
13    /*
14      1. 比如config中的一些信息不符合服务器的要求
15      2. 比如每次发送网络请求时，都希望在界面中显示一个请求的图标
16      3. 某些网络请求(比如登录(token))，必须携带一些特殊的信息
17    */
18    return config
19  }, err => {
20    console.log(err);
21  })
22}
```

```
21 })
22
23 // 2.1 响应拦截
24 instance.interceptors.response.use(res => {
25     return res.data
26 }, err => {
27     console.log(err);
28 })
29
30 // 3.发送真正的网络请求
31 return instance(config)
32 }
```

main.js

```
1 import Vue from 'vue'
2 import App from './App'
3
4 Vue.config.productionTip = false
5
6 /* eslint-disable no-new */
7 new Vue({
8     el: '#app',
9     render: h => h(App)
10 })
11
12 // 封装request模块
13 import {request} from "./network/request";
14
15 request({
16     url: '/home/multidata'
17 }).then(res => {
18     console.log(res);
19 }).catch(err => {
20     console.log(err);
21 })
```

执行结果



18 - Vue生命周期

