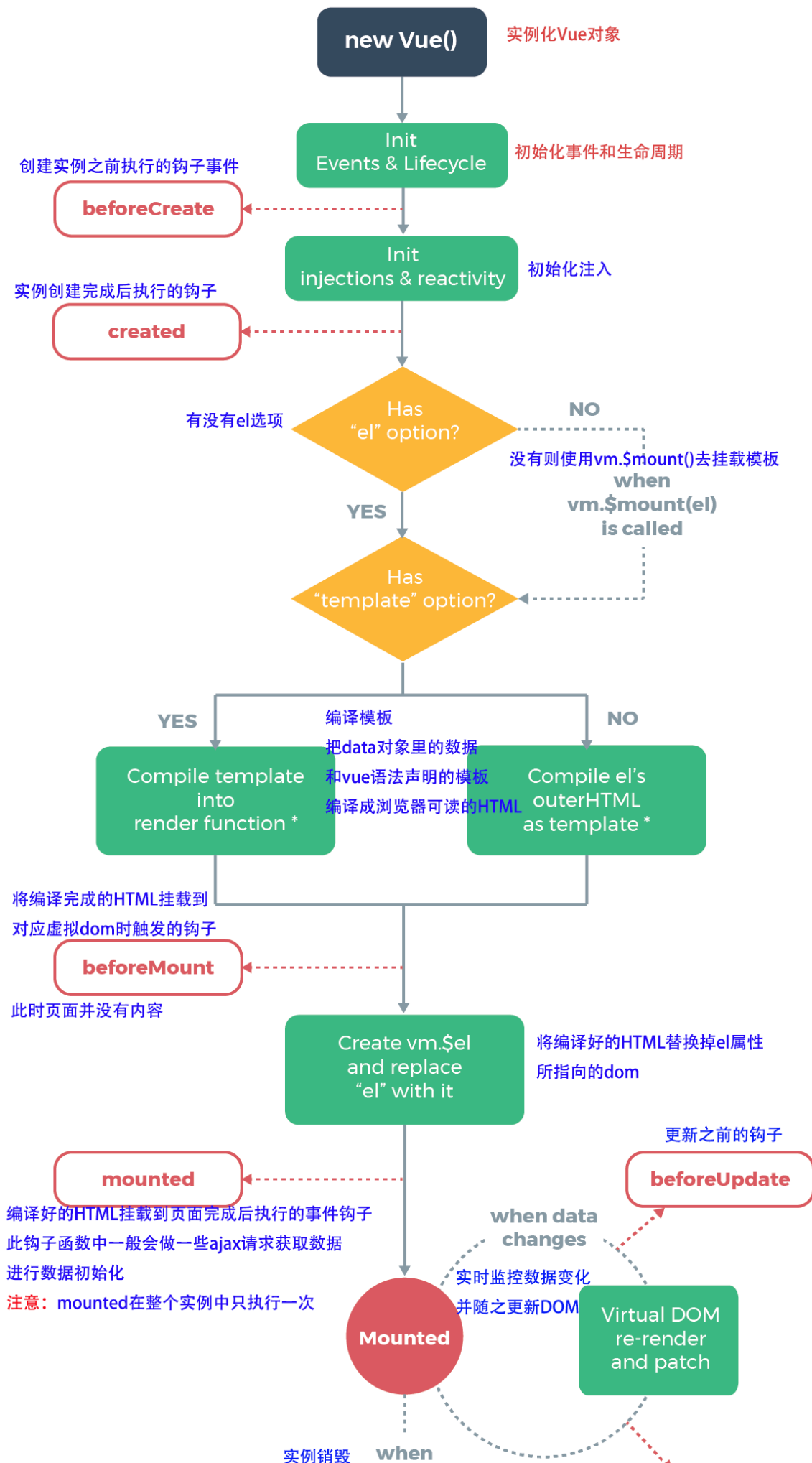
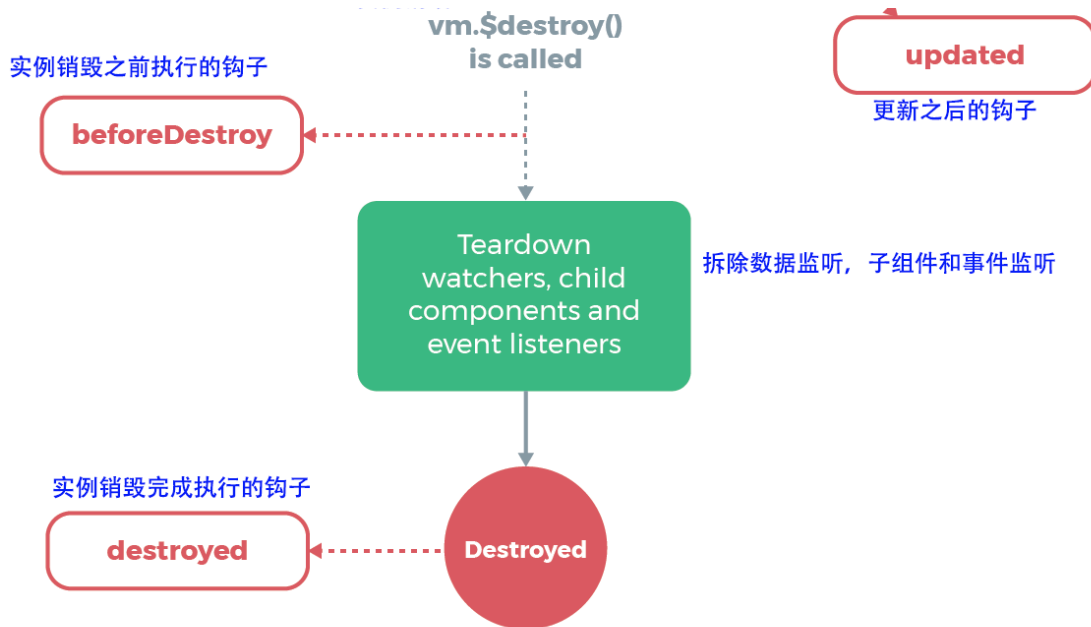


VUE生命周期

`Vue` 实例有一个完整的生命周期，也就是从开始创建、初始化数据、编译模板、挂载 DOM、渲染→更新→渲染、卸载等一系列过程，我们称这是 `Vue` 的生命周期。通俗说就是 `Vue` 实例从创建到销毁的过程，就是生命周期。

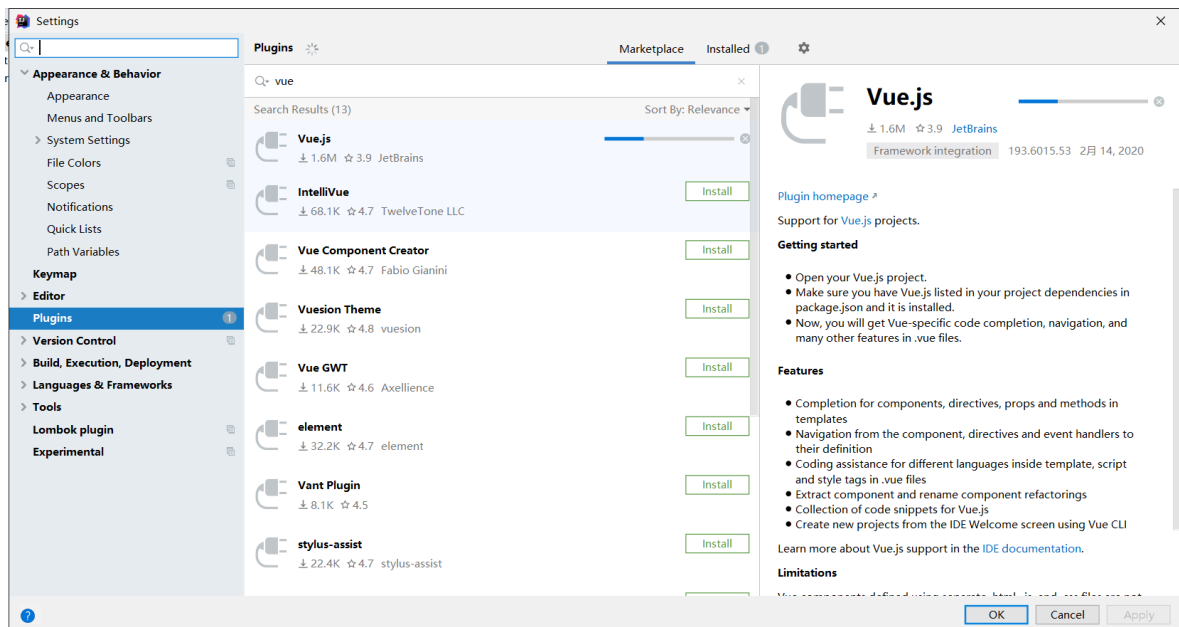




* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

前置环境

1. 在idea中的插件一栏中搜索vue, 安装



2. 导入在线CDN

```
1 <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js"></script>
2 <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>
```

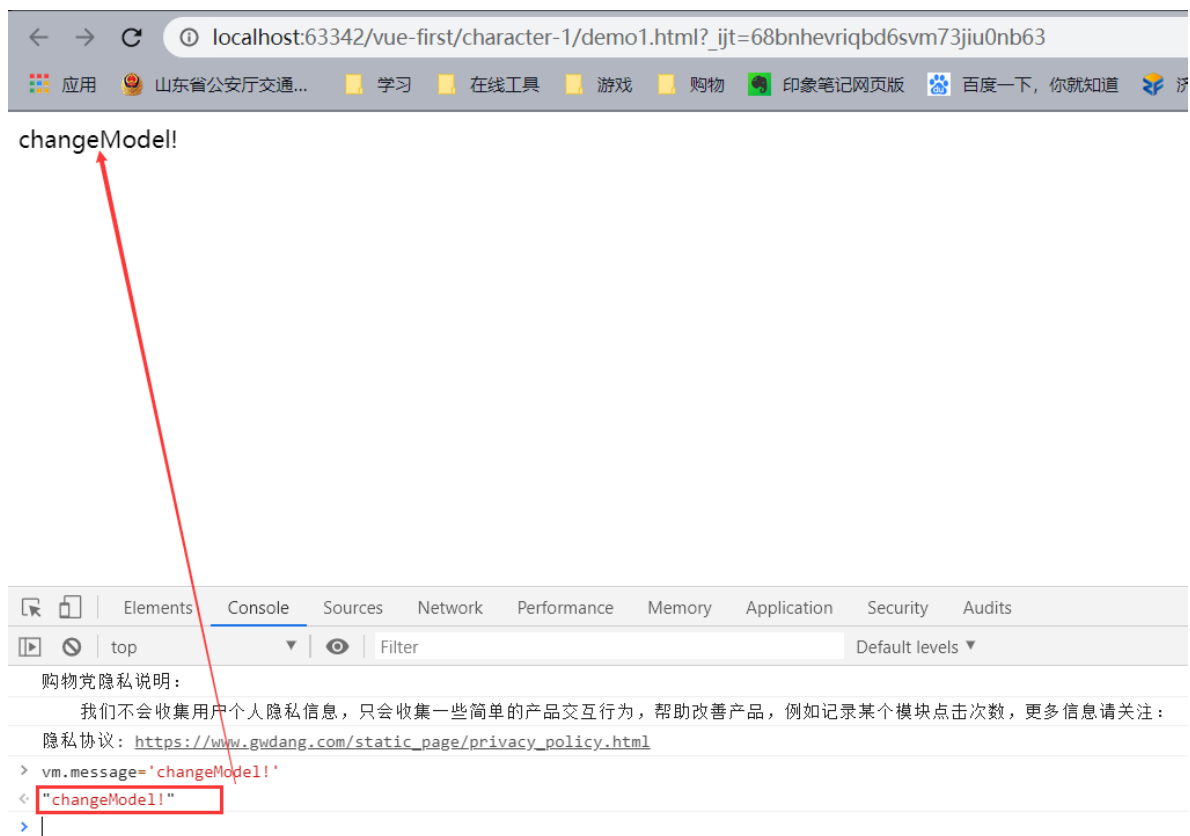
起步

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <!--第一个VUE程序-->
5   <meta charset="UTF-8">
6   <title>Title</title>
7   <!-- 1. 导入在线CDN-->
8   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>
9 </head>
10 <body>
11
12   <!--view层，变成了一个template模版-->
13
14   <div id="app"><!--id: app, 用于绑定数据-->
15     {{message}}<!--取出数据message-->
16   </div>
17
18   <script>
19
20     var vm = new Vue({
21       el: "#app", //el: 元素element, 绑定元素
22       /*Modle: 数据*/
23       data: { //data: 数据, 用于修改数据
24         message: "Hello,Vue!" //message: 具体的数据
25       }
26     });
27
28   </script>
29 </body>
30 </html>
```



Hello,Vue!

这里实现了 **双向绑定**，只要更改数据层，前段的view层也会跟着改变，而不用刷新界面，比如：



这样的效果以前的前段是做不到的，但是现在可以

基本语法

指令介绍

- 指令：`v-xxx` 的形式称之为指令，也就是说以 `v-` 开头的都是指令，这样来表示他们是VUE可以提供的特殊特性

绑定

`v-bind` 是 `vue` 中的绑定指令，作用是绑定到 `Model` 层的数据，其效果类似于 `{{}}`，简写为 `:`，比如

`v-bind:title="xx" === :title="xxx"`

使用：

```
1 <!DOCTYPE html>
2 <html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
```

```

6     <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>
7   </head>
8   <body>
9     <!--
10      1. 根据id绑定元素
11      2. v-bind绑定提示信息
12    -->
13    <div id="app" v-bind:title="message">
14      鼠标悬停几秒钟查看此处动态绑定的提示信息!
15    </div>
16
17    <script>
18
19      var vm = new Vue({
20        el: "#app",
21        data: {
22          message: "Hello,Vue!"
23        }
24      });
25
26    </script>
27  </body>
28 </html>

```

判断

v-if

需求: 假如 `msg` 为 `true` , 则显示 `VUE`

```

1   <body>
2
3     <!--假如msg为真, 则显示VUE-->
4     <div id="app" v-if="msg" v-bind:title="msg">VUE</div>
5
6
7     <script>
8       var vm = new Vue({
9         el: "#app",
10        data: {
11          msg: true
12        }
13      });
14    </script>
15
16
17  </body>

```

v-else

需求: 假如 `msg` 为 `true` , 则显示 `VUE` ; 假如 `msg` 为 `false` , 则显示 `Hello`

```

1   <body>

```

```

2
3     <!--假如msg为true, 则显示VUE-->
4     <!--假如msg为false, 则显示Hello-->
5     <div id="app">
6         <div v-if="msg">VUE</div>
7         <div v-else>Hello</div>
8     </div>
9
10    <script>
11        var vm = new Vue({
12            el: "#app",
13            data: {
14                msg: true
15            }
16        });
17    </script>
18
19
20 </body>

```

v-else-if

```

1 <body>
2
3     <!--假如msg为A, 则显示A-->
4     <!--假如msg为B, 则显示B-->
5     <!--假如msg为C, 则显示C-->
6     <!--假如msg不为以上, 则显示D-->
7     <div id="app">
8         <div v-if="msg==='A'">A</div>
9         <div v-else-if="msg==='B'">B</div>
10        <div v-else-if="msg==='C'">C</div>
11        <div v-else>D</div>
12    </div>
13
14    <script>
15        var vm = new Vue({
16            el: "#app",
17            data: {
18                msg: "A"
19            }
20        });
21    </script>
22
23
24 </body>

```

A

循环

v-for 是循环，可以遍历每一项

```
1 <body>
2
3 <!--首先绑定id-->
4 <div id="app">
5   <!--然后绑定items，然后遍历每一项item-->
6   <li v-for="item in items">{{item.message}}</li>
7 </div>
8
9 <script>
10   var vm = new Vue({
11     el: "#app",
12     data: {
13       items: [
14         {message: 'A'},
15         {message: 'B'},
16         {message: 'C'}
17       ]
18     }
19   });
20 </script>
21
22
23 </body>
```

还可以获取到当前的索引

```
1 <body>
2
3 <!--首先绑定id-->
4 <div id="app">
5   <!--然后绑定items，然后遍历每一项item-->
6   <li v-for="(item,index) in items">{{item.message}}----{{index}}</li>
7 </div>
8
9 <script>
10   var vm = new Vue({
11     el: "#app",
12     data: {
13       items: [
14         {message: 'A'},
15         {message: 'B'},
16         {message: 'C'}
17       ]
18     }
19   });
20 </script>
21
22
23 </body>
```


- A---0
- B---1
- C---2

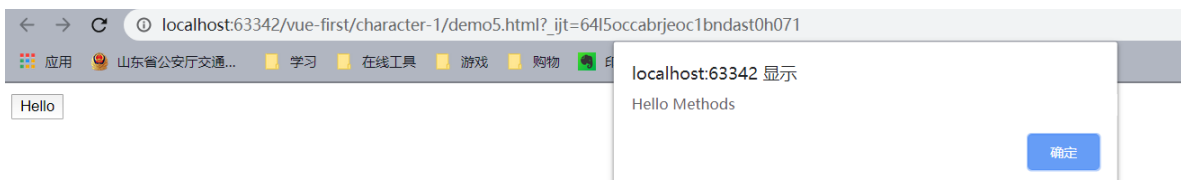
事件

- `v-on` 指令可以监听 `dom` 事件，并触发一些 `js` 代码，简写为 `@`，比如 `v-on:click="hello" === @click="Hello"`

jQuery 事件: https://www.w3school.com.cn/jquery/jquery_ref_events.asp

事件有了，那么肯定就有方法，之前我们学了 `el`，`data`，下面我们学习 `methods`，方法必须定义在 `methods` 中

```
1 <body>
2
3 <!--首先绑定id-->
4 <div id="app">
5   <!--使用v-on绑定点击事件，然后执行hello方法-->
6   <button v-on:click="hello">Hello</button>
7 </div>
8
9 <script>
10   var vm = new Vue({
11     el: "#app",
12     data: {
13       message: "Hello Methods"
14     },
15     methods: {
16       /*方法必须定义在methods里面*/
17       hello: function(){
18         /*定义一个叫做hello的function，this.message表示当前对象为message的数据*/
19         alert(this.message)
20       }
21     }
22   });
23 </script>
24
25
26 </body>
```



双向绑定

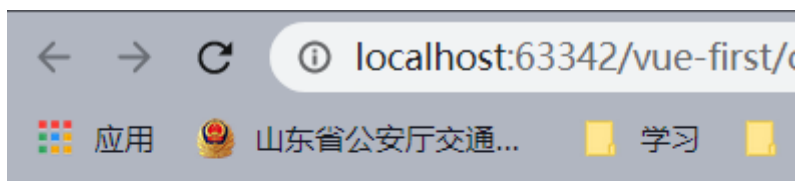
- 当数据变化时，视图随之变化
- 当视图变化时，数据随之变化

这就是VUE的精髓，双向绑定

v-model

可以使用 `v-model` 在表单的 `<input>` `<textarea>` `<select>` 等元素上实现双向绑定

```
1 <body>
2
3 <!--首先绑定id-->
4 <div id="app">
5   <!--实现双向绑定，令视图改变的时候数据也跟着变，这里绑定了message-->
6   <input type="text" v-model="message"> <br>
7   绑定的数据: <br>
8   {{message}}
9 </div>
10
11 <script>
12   var vm = new Vue({
13     el: "#app",
14     data: {
15       message: ""
16     },
17     methods: {}
18   });
19 </script>
20
21
22 </body>
```



在输入框中打出的文字

绑定的数据:

在输入框中打出的文字

```
1 <body>
2
3 <!--首先绑定id-->
4 <div id="app">
5   男: <input type="radio" v-model="message" value="男">
```

```

6      女: <input type="radio" v-model="message" value="女">
7
8      <p>选中的值: {{message}}</p>
9  </div>
10
11  <script>
12      var vm = new Vue({
13          el: "#app",
14          data: {
15              message: ""
16          },
17          methods: {}
18      });
19  </script>
20
21
22  </body>

```

男: ☐ 女: ☒

选中的值: 女

因为数据双向绑定之后，下拉框会出现一个问题，就是默认选中值会变为空，如：

```

1  <select v-model="message">
2    <option>A</option>
3    <option>B</option>
4    <option>C</option>
5  </select>

```



选中的值:

假如令默认值为空的时候，苹果用户还是选择不了（不是显示文字的问题，是 `value=""`）

```

1  <select v-model="message">
2    <option value="">A</option>
3    <option>B</option>
4    <option>C</option>
5  </select>

```



选中的值:

而苹果用户当第一个值变为空的时候就选择不了

为了照顾土豪用户，我们把第一个值设置为空，显示为 **"--请选择--"**

```

1   <body>
2
3   <!-- 首先绑定id-->
4   <div id="app">
5
6       <!--
7           因为数据双向绑定之后，下拉框会出现一个问题，就是第一个值会变为空
8           而苹果用户当第一个值变为空的时候就选择不了
9           为了照顾土豪用户，我们把第一个值设置为空，显示为"--请选择--"
10      -->
11      <select v-model="message">
12          <option value="" disabled>--请选择--</option>
13          <option>A</option>
14          <option>B</option>
15          <option>C</option>
16      </select>
17
18      <p>选中的值: {{message}}</p>
19  </div>
20
21  <script>
22      var vm = new Vue({
23          el: "#app",
24          data: {
25              message: ""
26          },
27          methods: {}
28      });
29  </script>
30
31
32  </body>

```

--请选择-- ▼

选中的值:

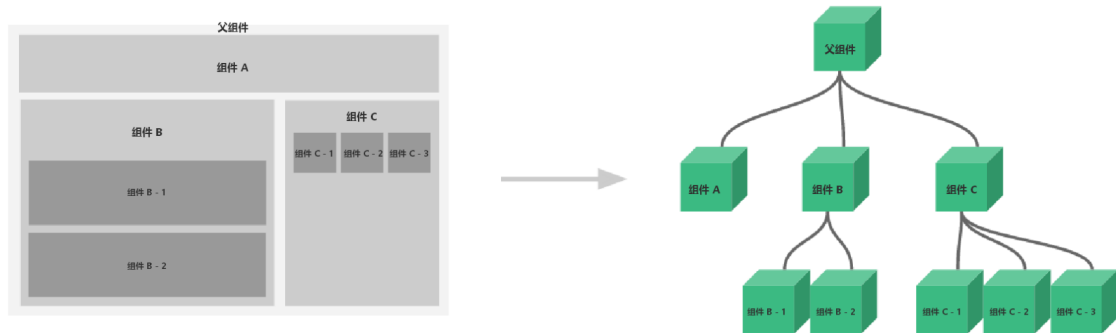
A ▼

选中的值: A

组件

什么是组件:

组件可以认为是多种标签的组合体，也可以认为是 容器+容器内容，可以认为是 div+内容，官网给了一张图:



组件起步

- 使用Vue组件component

```
1 <body>
2
3 <!-- 首先绑定元素 -->
4 <div id="app">
5   <vc></vc>
6 </div>
7
8 <script>
9
10
11   /*用刚才定义的vue创建一个组件vc*/
12   Vue.component("vc", {
13     /*template: 模版, 这里只有一个列表项组件, 叫做Hello*/
14     template: "<li>Hello</li>"
15   });
16
17   /*定义Vue*/
18   var vm = new Vue({
19     el: "#app"
20   });
21 </script>
22
23 </body>
```

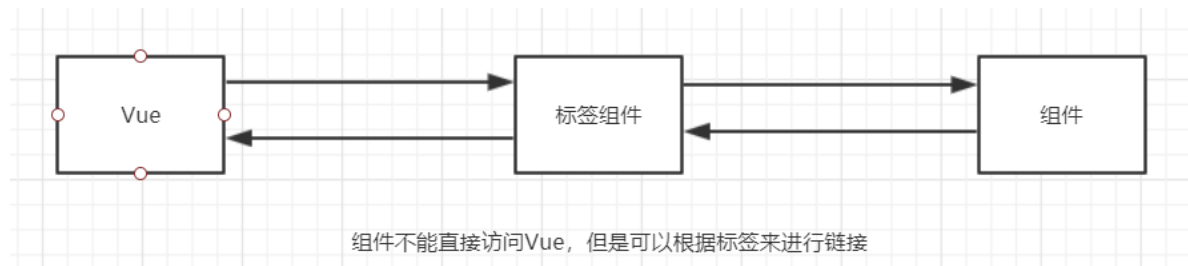
- Hello

数据绑定

像上面这样使用组件不是真正的组件，因为Vue的精髓之一是数据绑定，所以现在我们要实现数据绑定

在使用数据绑定之前我们要明白几件事：

1. 组件 `component` 不能直接访问Vue里面的 `data`
2. 组件标签可以访问 `Vue` 里面的 `data`，也可以访问组件自身
3. 基于以上两点，我们要用组件标签这个中间件来将Vue和组件进行链接



```
1 <body>
2
3 <!--首先绑定元素-->
4 <div id="app">
5   <!--使用v-for链接data，使用v-bind链接component-->
6   <vc v-for="item in items" v-bind:binditem="item"></vc>
7 </div>
8
9 <script>
10
11   Vue.component("vc",{
12     /*使用props接受数据*/
13     props: ['binditem'],
14     template: "<li>{{binditem}}</li>",
15   });
16
17   /*定义Vue*/
18   var vm = new Vue({
19     el: "#app",
20     data: {
21       items: ['A','B','C']
22     }
23   });
24 </script>
25
26 </body>
```

- A
- B
- C

错误类型

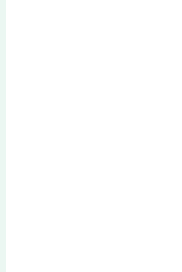
除了上面的之外，在这里绑定数据的时候还有一个小坑，就是当props接受数据的时候，参数的名字只能第一个字母大写，如果其他字母大写，数据绑定就会失效

比如：`Binditem` 可以，`BindItem` 不行，失效实现：



列表符都出现了，但是参数没出现，这个时候就要想想为啥了

还有一种错误：



啥也没有，代表根本就没绑上，肯定是参数错了，比如：

```
<!-- 首先绑定元素-->
<div id="app">
  <!-- 使用v-for链接data，使用v-bind链接component-->
  <vc v-for="item in items" v-bind:bindItem="item"></vc>
</div>

<script>

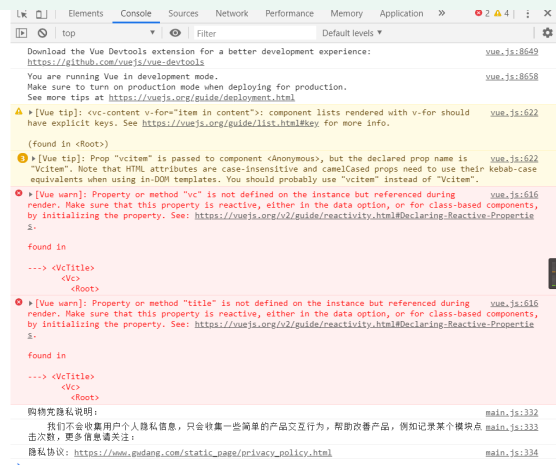
  Vue.component("vc",{
    /*使用props接受数据*/
    props: ['bindItem'],
    template: "<li>{{bindIte}}</li>",
  });
```

还有一种错误是中间加了横杠，就会出现NaN

```
1  Vue.component("vc-title",{
2    props: ['vc-title'],
3    template: '<p>{{vc-title}}</p>'
4  });
```

NaN

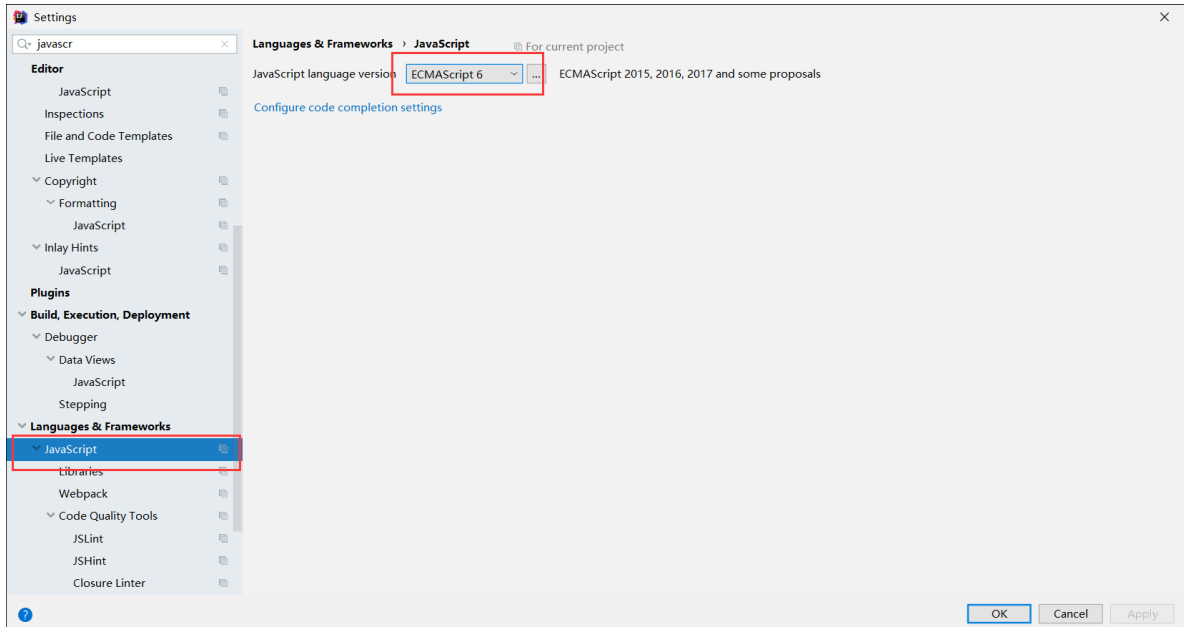
- JAVA
- Linux
- Vue



Axios 异步通信

介绍和环境

首先第一件事就是改为ES6规范，因为ES5根本就写不了这个代码



为什么要学 Axios

因为 vue 是专注于视图层的，而且作者严格遵守关注点分离原则（SOC），所以VUE不包含网络通信的功能

为了解决通信问题，我们需要网络通信的框架，这里有两个

- JQuery
- Axios

那么为什么要选择 Axios？

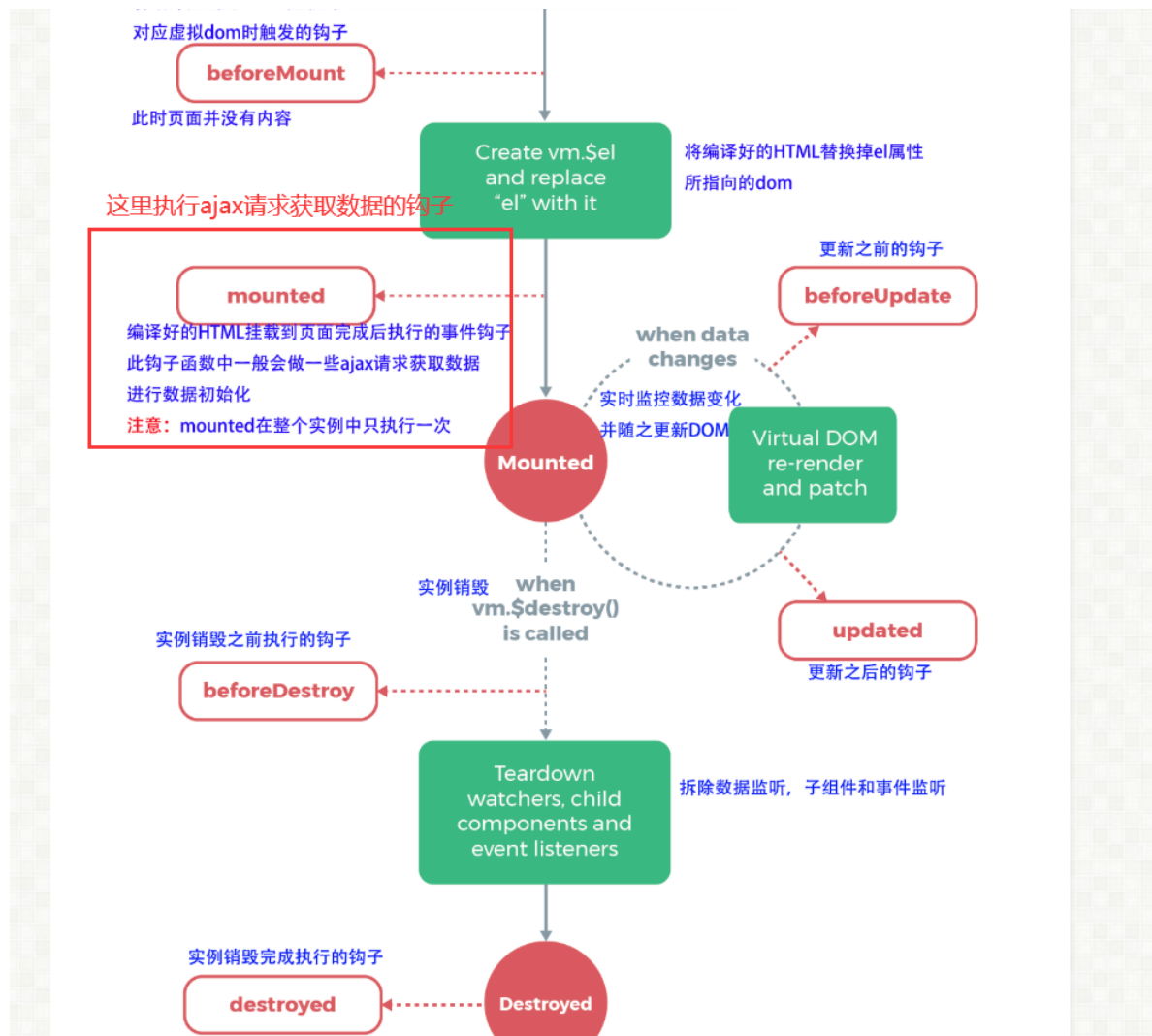
1. VUE作者推荐
2. JQuery 操作 dom 太频繁

在线CDN

```
1 <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```


Axios 起步

我们查看一下 `Vue` 的生命周期，发现了很多钩子事件，这些钩子事件就是可以插入到VUE的生命周期中，在那里执行的，而我们可以看到网络通信的钩子函数是在 `mounted` 那里执行的



因为我们经常使用的是 `JSON` 格式，所以这里来了一段 `JSON` 代码

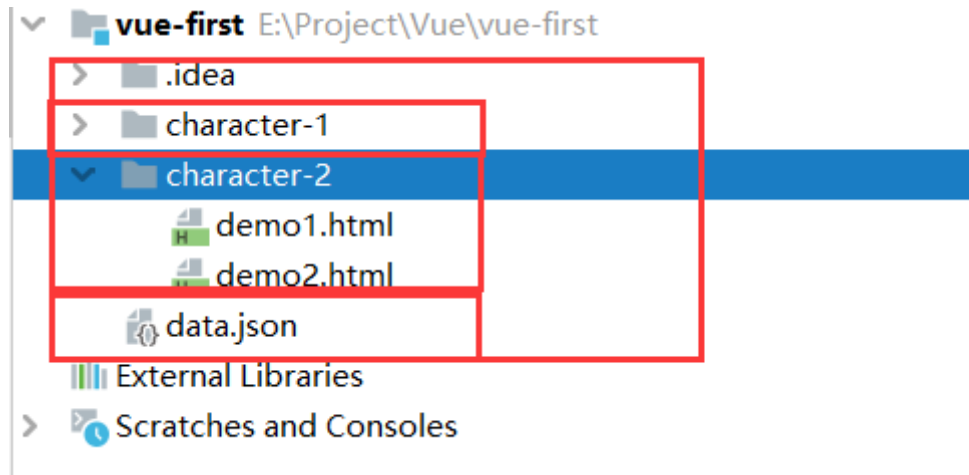
```
1  {
2    "name": "狂神说Java",
3    "url": "https://blog.kuangstudy.com",
4    "page": 1,
5    "isNonProfit": true,
6    "address": {
7      "street": "含光门",
8      "city": "陕西西安",
9      "country": "中国"
10   },
11   "links": [
12     {
13       "name": "bilibili",
```

```

14     "url": "https://space.bilibili.com/95256449"
15   },
16   {
17     "name": "狂伸说java",
18     "url": "https://blog.kuangstudy.com"
19   },
20   {
21     "name": "百度",
22     "url": "https://www.baidu.com/"
23   }
24 ]
25 }

```

来一个目录：

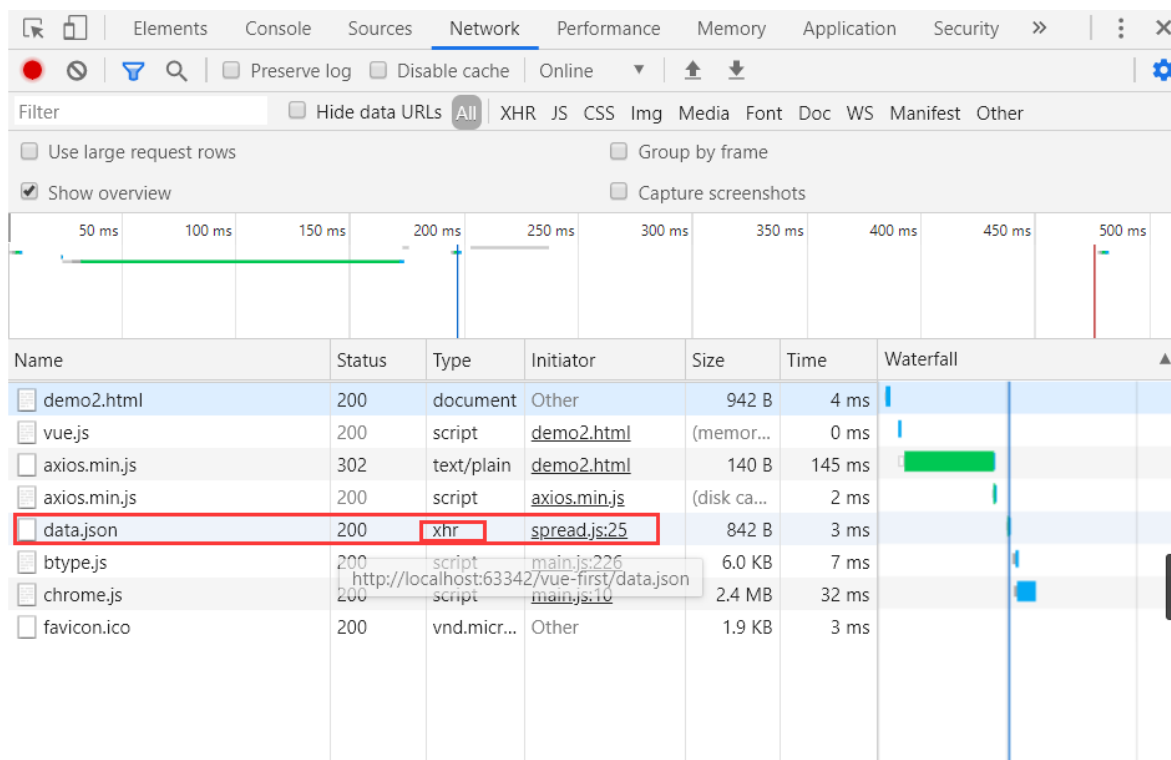
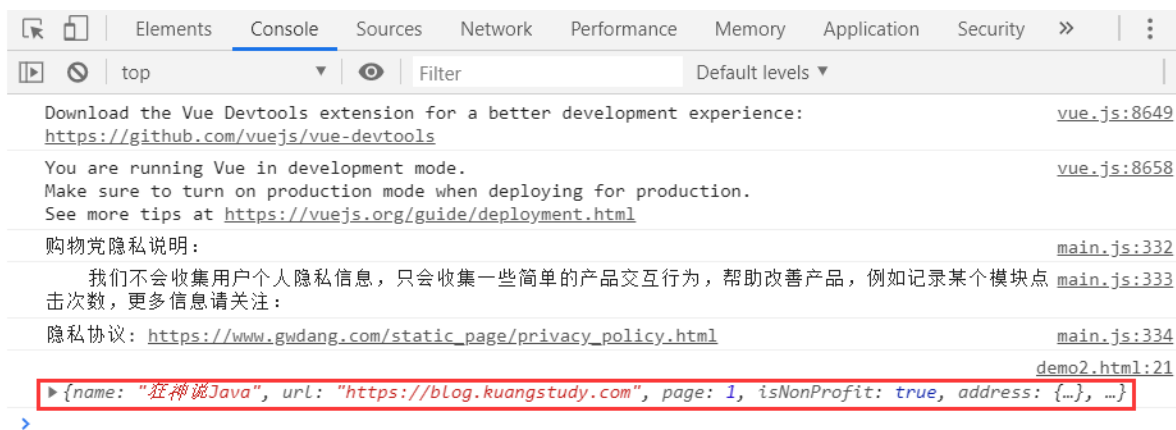


上代码：

```

1  <body>
2
3  <div id="app"></div>
4
5  <script>
6
7
8      var vm = new Vue({
9          el: "#app",
10         data: {},
11         methods: {},
12         mounted() { // 钩子函数，链式编程
13             /*使用get函数就是axios.get(), 其余同理，我们直接访问这个路径，得到response返回值，然后输出
14             一下看看*/
15             axios.get("../data.json").then(response=>(console.log(response.data)));
16         }
17     });
18 </script>
19 </body>

```



注意，这里是ES6的新特性，所以有些ES5的浏览器不支持，以后会学下降版本

data 方法绑定

刚才我们用的是直接输出，但是我们显然不会这么做，我们需要将这些数据放到 `data` 中，毕竟这才是正规操作

但是注意：

1. 这里的这个 `data` 不是属性data，而是方法 `data`
2. 我们需要放的不是数据，而是数据的格式

这样说可能不懂，但是一写代码就懂了

首先是JSON，注意它的格式：

```

1  {
2    "name": "狂神说Java",
3    "url": "https://blog.kuangstudy.com",
4    "page": 1,
5    "isNonProfit": true,
6    "address": {
7      "street": "含光门",
8      "city": "陕西西安",
9      "country": "中国"
10   },
11   "links": [
12     {
13       "name": "bilibili",
14       "url": "https://space.bilibili.com/95256449"
15     },
16     {
17       "name": "狂伸说java",
18       "url": "https://blog.kuangstudy.com"
19     },
20     {
21       "name": "百度",
22       "url": "https://www.baidu.com/"
23     }
24   ]
25 }

```

VUE

```

1  <body>
2
3    <div id="app">
4      {{info.name}} <br>
5      {{info.address.street}} <br>
6      <li v-for="item in info.links">
7        <a v-bind:href="item.url">{{item.url}}</a>
8      </li>
9    </div>
10
11    <script>
12
13
14      var vm = new Vue({
15        el: "#app",
16        data: {},
17        methods: {},
18        /*看好了，这个data是data方法，不是上面的data属性*/
19        data(){
20          return {
21            /*定义info的格式，注意，这个格式必须和返回来的json数据格式要相同
22             * 可以不写，但是一定不能写错
23             * 不写的比如：isNonProfit,page等等我就没写，但是没错
24             * */
25            info: {
26              name: null,
27              address: {
28                street: null,
29                city: null,

```

```

30         },
31         links: [
32             {
33                 name: null,
34                 url: null
35             }
36         ]
37     }
38 }
39 },
40 mounted(){
41     /*这里令info为传过来的response.data*/
42     axios.get("../data.json").then(response=>(this.info=response.data));
43 }
44 });
45 </script>
46
47 </body>

```

狂神说Java

含光门

- <https://space.bilibili.com/95256449>
- <https://blog.kuangstudy.com>
- <https://www.baidu.com/>

闪烁问题解决

我们看生命周期的时候就会发现，数据绑定是在后面的操作，那么在数据绑定之前页面都是el属性的内容，那么这就会产生一个结果：

页面首先出现el属性，然后数据替换el属性，这样就形成了闪烁

要解决闪烁问题好解决，我们只需要让数据替换之后再把页面给用户渲染出来就好了

这个方法不是 VUE 的，只是可以这样去解决

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6     <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js"></script>
7     <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
8
9     <style>

```

```

10      /*通过属性选择器，让它没加载出来之前白屏*/
11      [v-clock]{
12          display: none;
13      }
14  </style>
15  </head>
16  <body>
17
18      <div id="app" v-clock>
19          {{info.name}} <br>
20          {{info.address.street}} <br>
21          <li v-for="item in info.links">{{item.url}}</li>
22      </div>
23
24  </body>
25  </html>

```

计算属性

计算属性是什么

计算属性的重点突出在 **属性** 两个字上（属性是名词）

- 首先它是个 **属性**
- 其次这个属性有 **计算** 的能力（计算是动词），这里的 **计算** 就是个函数；
- 可以想象为缓存！

也就是说，这是个拥有着方法能力的属性，当数据不变的时候不会发生变化，起到了一个缓存的作用

和方法不同，比如 **1+1**，当数据不会改变的时候，方法是再次算一次然后返回结果，而计算属性是直接返回上次的结果

代码：

```

1  <body>
2
3      <div id="app" >
4          <!--方法调用使用()-->
5          <p>{{count1()}}</p>
6          <!--计算属性属于属性，直接调用-->
7          <p>{{count2}}</p>
8      </div>
9
10     <script>

```

```

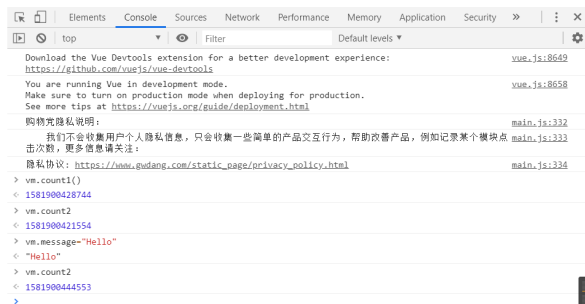
11
12
13     var vm = new Vue({
14         el: "#app",
15         data: {
16             message: "data"
17         },
18         methods: {
19             /*方法*/
20             count1: function () {
21                 return Date.now();
22             }
23         },
24         computed: {
25             /*computed: 计算属性，计算属性写法上和方法没有什么不同，但是有根本的不同*/
26             count2: function () {
27                 this.message;
28                 return Date.now();
29             }
30         }
31     });
32 </script>
33
34 </body>

```

1581900444552

1581900444553

1. 第一次加载页面显示的数据为xxx1554
2. 在控制台调用count1()方法得到xxx8744
3. 在控制台调用count2计算属性得到xxx1554
4. 在控制台改变计算属性中的属性message
5. 在控制台调用count2计算属性得到xxx4553



- `count1()` 方法每次返回的结果必定计算
- `count2` 计算属性只有当其中的数据变化的时候才会再次计算

内容分发

插槽是什么

在一个页面中，使用组件对内容进行动态的更改，这就叫做插槽

比如狂神的页面：只有中间的内容区域改变了，其余都不变，这个用插槽也可以做



当然，不用插槽也可以做，但是使用了插槽之后回变得更加容易

插槽起步

首先来看一段代码：

```
1 <body>
2
3 <p>Bean</p>
4 <ul>
5 <li>JAVA</li>
6 <li>Linux</li>
7 <li>Vue</li>
8 </ul>
9
10 </body>
```

Bean

- JAVA
- Linux
- Vue

无论是Bean，Java，Linux还是Vue，这些数据显然是不能够写死的，应该从数据读出来，下面我们就是用插槽来替换

提示：

- 插槽也是组件

```
1   <body>
2
3   <div id="app" >
4     <vc>
5       <!--注意点是：这里也需要使用slot进行绑定，假如不绑定不会报错，但是不会显示数据-->
6       <vc-title slot="vc-title" v-bind:Vctitle="title"></vc-title>
7       <vc-content slot="vc-content" v-for="item in content" v-bind:Vcitem="item"></vc-
content>
8     </vc>
9   </div>
10
11  <script>
12
13    /*注意点：<slot></slot>不能是根标签，外边要套上一层容器，否则会报错
14    * vc是总组件
15    * <slot name="vc-title"></slot> 使用name绑定组件vc-title
16    * <slot name="vc-content"></slot>使用name绑定组件vc-content
17    * */
18    Vue.component("vc",{
19      template: '<div>\
20                <slot name="vc-title"></slot>\
21                <ul>\
22                  <slot name="vc-content"></slot>\
23                </ul>\
24                </div>'
25    });
26
27    /*插槽也是组件，这个是分组件vc-title
28    * 接受参数Vctitle，注意这个参数除了第一个字母之外不能大写，也不能用 - 链接
29    * */
30    Vue.component("vc-title",{
31      props: ['Vctitle'],
32      template: '<p>{{Vctitle}}</p>'
33    });
34
35    /*插槽也是组件，这个是分组件vc-content
36    * 接受参数Vcitem，注意这个参数除了第一个字母之外不能大写，也不能用 - 链接
37    * */
38    Vue.component("vc-content",{
39      props: ['Vcitem'],
40      template: '<li>{{Vcitem}}</li>'
41    });
42
43
44    /*定义好数据*/
45    var vm = new Vue({
46      el: "#app",
47      data: {
48        title: "Bean",
49        content: ["JAVA","Linux","Vue"]
50      },
51    });
52  </script>
53
```

Bean

- JAVA
- Linux
- Vue

错误类型

1. 报错说 `<slot></slot>` 标签不能为 **root** 标签

在 `component` 组件定义的时候使用容器包裹一下，比如 `<div></div>`，不能用 `<slot></slot>` 作为根标签

2. 数据不显示，但是不报错

在使用插槽标签的时候也要使用 **slot** 属性链接插槽，否则不会报错，但是不显示数据

3. 报错

注意点是组件的数据绑定的时候 `props` 接收的参数除了第一个字母之外不能大写，而且不能有横杠链接

自定义事件

什么是自定义事件

问题描述

内容分发的部分结束之后，我们发现一个问题：如果我们想要删除 `data` 里面的数据是删不掉的，因为插槽也算是一个组件，而数据是存放在 `Vue` 中的 `data` 部分里的

自定义事件

我们在 `Vue` 里面定义删除方法，但是在插槽组件中调用不到（因为插槽组件只能调用自己组件中的方法），所以这个时候：

我们就需要新的内容：**自定义事件** 了

我们都知道，事件分为很多类型，比如点击事件，长按事件，之前讲过事件使用 `v-on` 绑定，然后执行方法

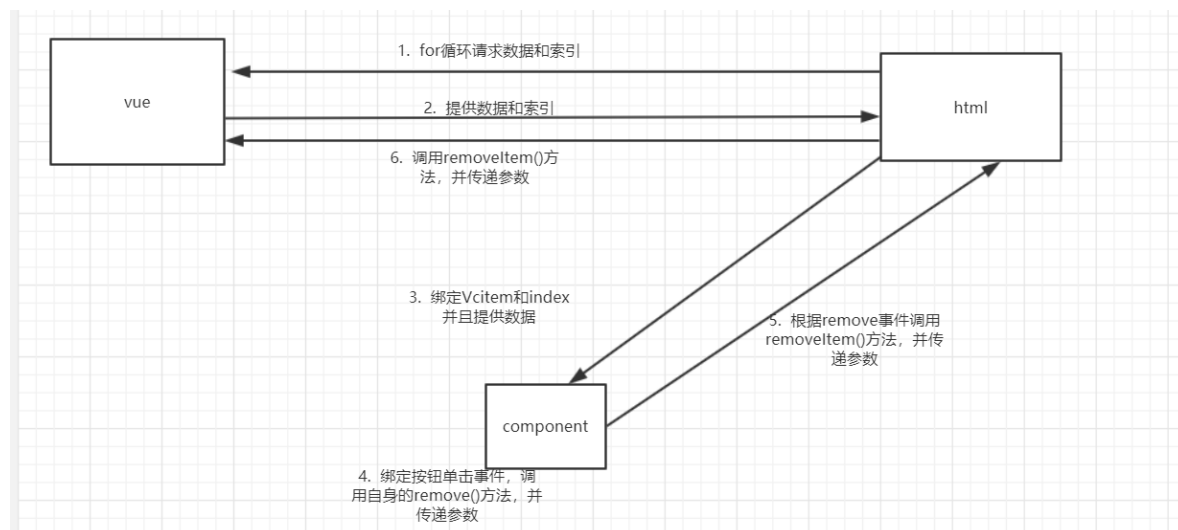
那么自定义事件就是我们自己定义的事件，和点击事件长按事件没有什么区别，都需要 `v-on` 进行绑定（简写 `@`）

自定义事件的功能

使用自定义事件，可以在任何组件上执行对应的方法，以达到在组件中使用其他组件的功能

使用自定义事件

使用： `this.$emit('自定义事件名称', 向执行的方法传递的参数);`



```
1 <body>
2
3 <div id="app" >
4   <vc>
5     <!--v-on绑定事件remove, 执行对应方法removeItem-->
6     <vc-content slot="vc-content" v-for="(item,index) in content" :Vcitem="item"
:index="index" @remove="removeItem(index)">
7       </vc-content>
8     </vc>
9   </div>
10
11 <script>
12
13   Vue.component("vc",{
14     template: '<div>\
15       <ul>\
16         <slot name="vc-content"></slot>\
17       </ul>\
18     </div>'
19   });
```

```

20
21      /*获得参数Vcitem和索引index
22      令Button绑定点击事件执行vc-content中的remove方法
23      * */
24      Vue.component("vc-content",{
25          props: ['Vcitem','index'],
26          template: '<li>{{Vcitem}} <button @click="remove(index)">删除</button></li>',
27          methods: {
28              /*这里的remove方法
29              1.接受参数
30              2.执行自定义事件remove并传递参数
31              */
32              remove: function (index) {
33                  this.$emit('remove',index);
34              }
35          }
36      });
37
38
39      /*定义好数据*/
40      var vm = new Vue({
41          el: "#app",
42          data: {
43              title: "Bean",
44              content: ["JAVA","Linux","Vue"]
45          },
46          methods: {
47              removeItem: function (index) {
48                  /*splice这个js方法是个万能方法，他有多参数：splice(索引，删除几个元素，添加的元
49                  素...)
50
51                  * 所以在Index索引处删除一个，什么元素也不添加是(index,1)
52                  * 注意添加的元素是个可变参数
53                  * */
54                  this.content.splice(index,1);
55              }
56          }
57      });
58      </script>
59
60 </body>

```

第一个VUE程序

环境安装

首先安装环境

1. **nodejs** : <http://nodejs.cn/download/>
2. **cnpm** (淘宝国内镜像源) :

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

3. `vue-cli` : `cnpm install vue-cli -g`

安装完成之后可以使用 `vue list` 命令在命令行中查看有什么模版

```
C:\Users\Bean>vue list

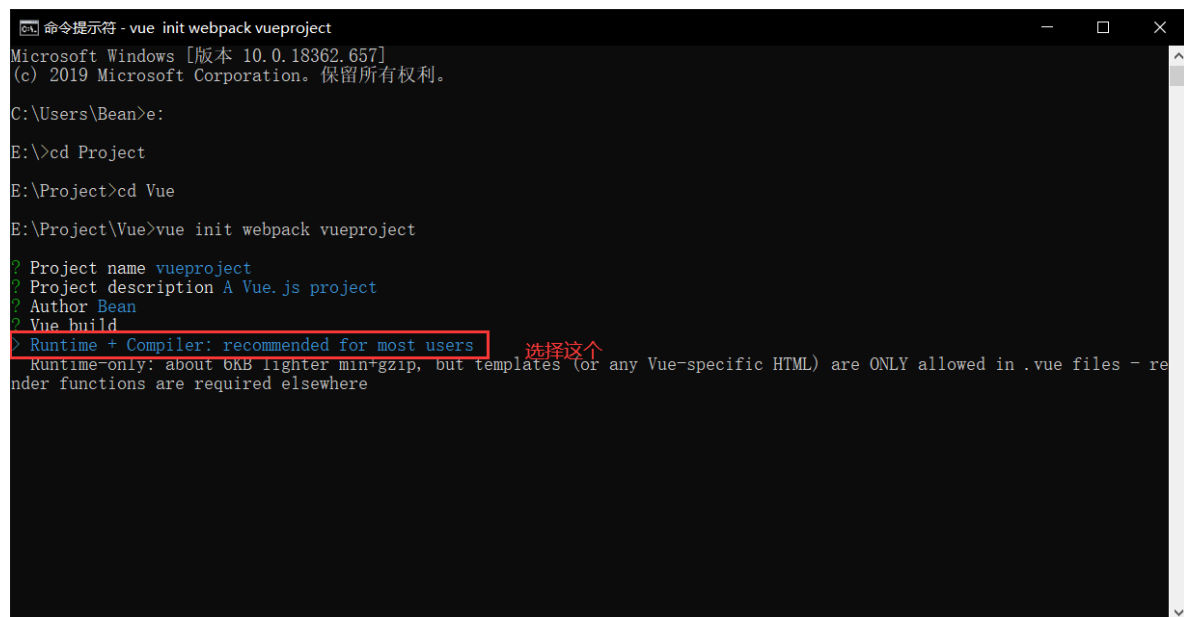
Available official templates:

★ browserify - A full-featured Browserify + vueify setup with hot-reload, linting & unit testing.
★ browserify-simple - A simple Browserify + vueify setup for quick prototyping.
★ pwa - PWA template for vue-cli based on the webpack template
★ simple - The simplest possible Vue setup in a single HTML file
★ webpack - A full-featured Webpack + vue-loader setup with hot reload, linting, testing & css extraction.
★ webpack-simple - A simple Webpack + vue-loader setup for quick prototyping.
```

第一个VUE项目

初始化项目

1. 切到对应的目录
2. 在目录下使用命令 `vue init webpack vueproject` (`vueproject` 是项目名字)



```
命令提示符 - vue init webpack vueproject
Microsoft Windows [版本 10.0.18362.657]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\Bean>e:
E:\>cd Project
E:\Project>cd Vue
E:\Project\Vue>vue init webpack vueproject

? Project name vueproject
? Project description A Vue.js project
? Author Bean
? Vue build
> Runtime + Compiler: recommended for most users 选择这个
  Runtime-only: about 6kB lighter min+gzip, but templates (or any Vue-specific HTML) are ONLY allowed in .vue files - render functions are required elsewhere
```

```
选择命令提示符 - vue init webpack myvue
Microsoft Windows [版本 10.0.18362.657]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\Bean>e:

E:\>cd Project

E:\Project>cd Vue

E:\Project\Vue>vue init webpack myvue

? Project name myvue
? Project description A Vue.js project
? Author Bean
? Vue build standalone
? Install vue-router? No
? Use ESLint to lint your code? No
? 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)
  Yes, use NPM
  Yes, use Yarn
> No, I will handle that myself
```

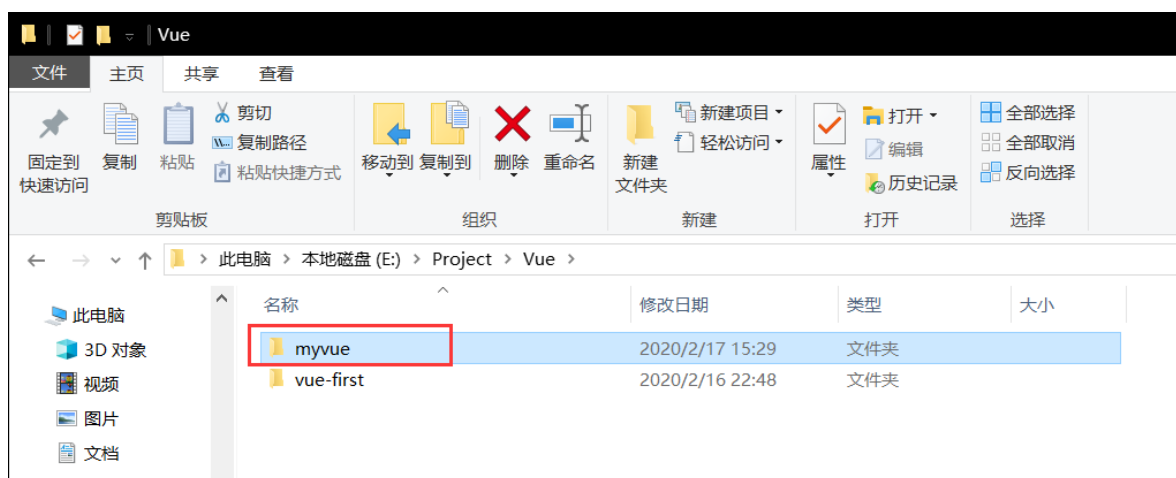
```
命令提示符
E:\Project>cd Vue
E:\Project\Vue>vue init webpack myvue

? Project name myvue 项目名称 (默认为myvue), 可以直接回车跳过
? Project description A Vue.js project 项目描述, 直接回车跳过
? Author Bean 作者, 打一个作者名字
? Vue build standalone
? Install vue-router? No 是否安装vue-router, 打N, 我们要自己安装
? Use ESLint to lint your code? No 是否安装。., 选择N, 不安装
? Set up unit tests No
? Setup e2e tests with Nightwatch? No N
? Should we run `npm install` for you after the project has been created? (recommended) no
  vue-cli · Generated "myvue".
# Project initialization finished!
# =====
To get started:

  cd myvue
  npm install (or if using yarn: yarn)
  npm run dev

Documentation can be found at https://vuejs-templates.github.io/webpack

E:\Project\Vue>
```



安装对应环境

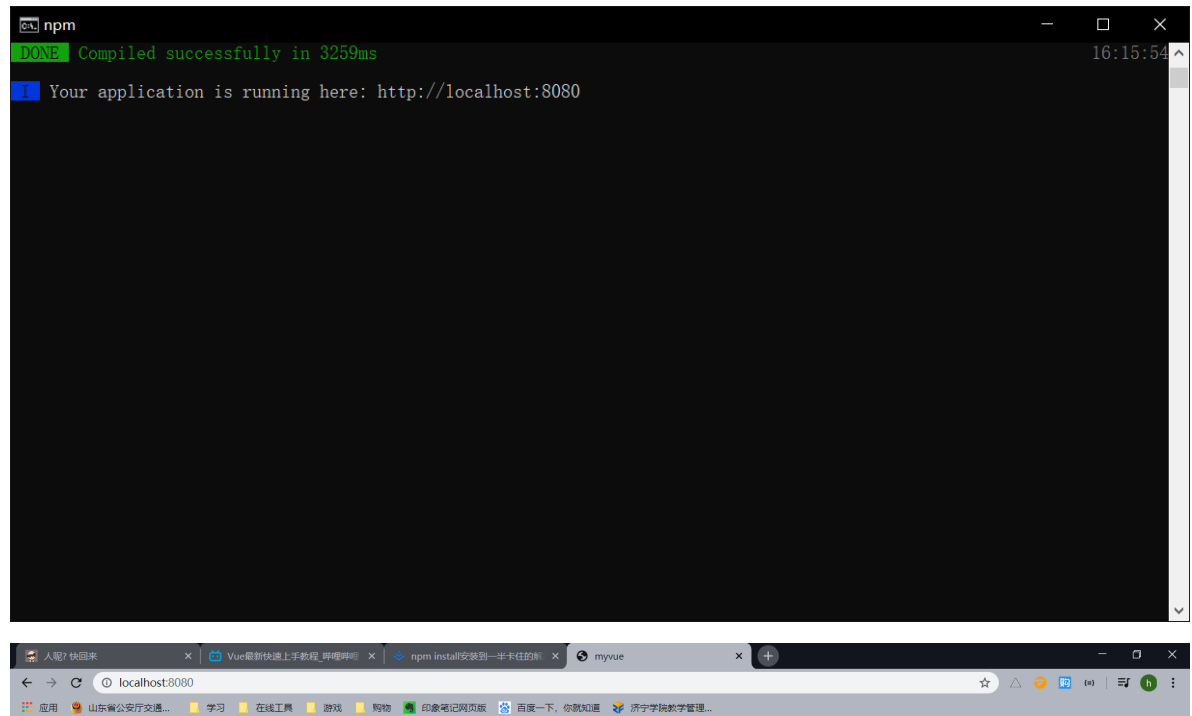
1. 命令行进入到 `myvue` 目录
2. 初始化项目环境: `npm install`, 假如不行就用 `cnpm install`

等待中。。。。

```
E:\Project\Vue\myvue>npm install
npm WARN deprecated extract-text-webpack-plugin@3.0.2: Deprecated. Please use https://github.com/webpack-contrib/mini-css-extract-plugin
npm WARN deprecated bfj-node4@5.3.1: Switch to the `bfj` package for fixes and new features!
npm WARN deprecated core-js@2.6.11: core-js@<3 is no longer maintained and not recommended for usage due to the number of issues. Please, upgrade your dependencies to the actual version of core-js@3.
npm WARN deprecated browserslist@1.7.7: Browserslist 2 could fail on reading Browserslist >3.0 config used in other tools.
[.....] / fetchMetadata: sill pacote version manifest for util@0.10.3 fetched in 331ms
```

运行

初始化完成之后，在命令行中使用 `npm run dev` 打包运行，运行完毕给一个端口，直接进去即可（不是管理员可能不行）



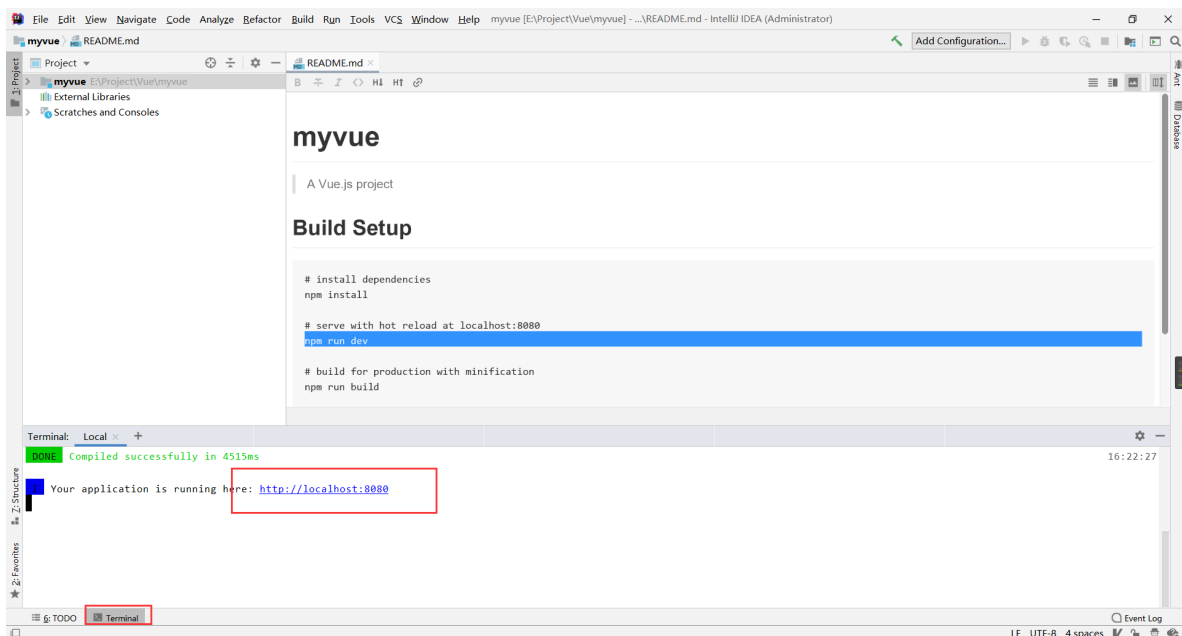
使用 `ctrl+c` 结束运行

使用IDEA

直接 `open` 就可以了，不用 `import`，但是 `IDEA` 中的控制台不是管理员模式，所以设置 `IDEA` 以管理员身份打开



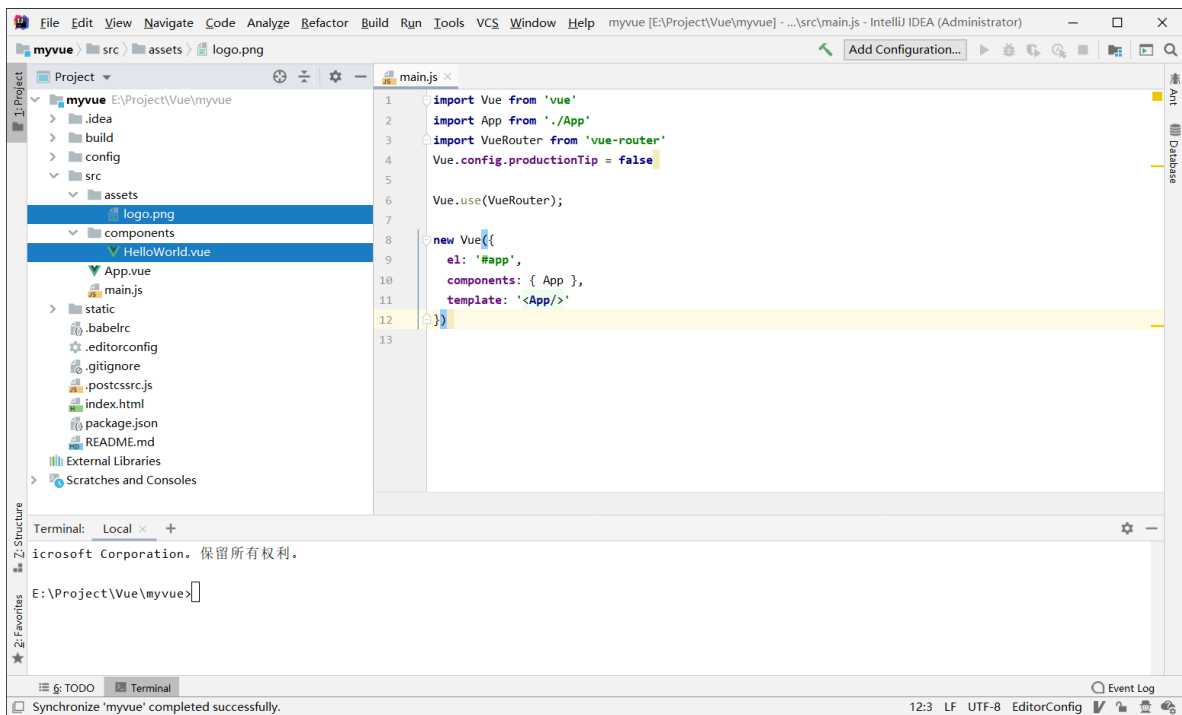
然后就成功了



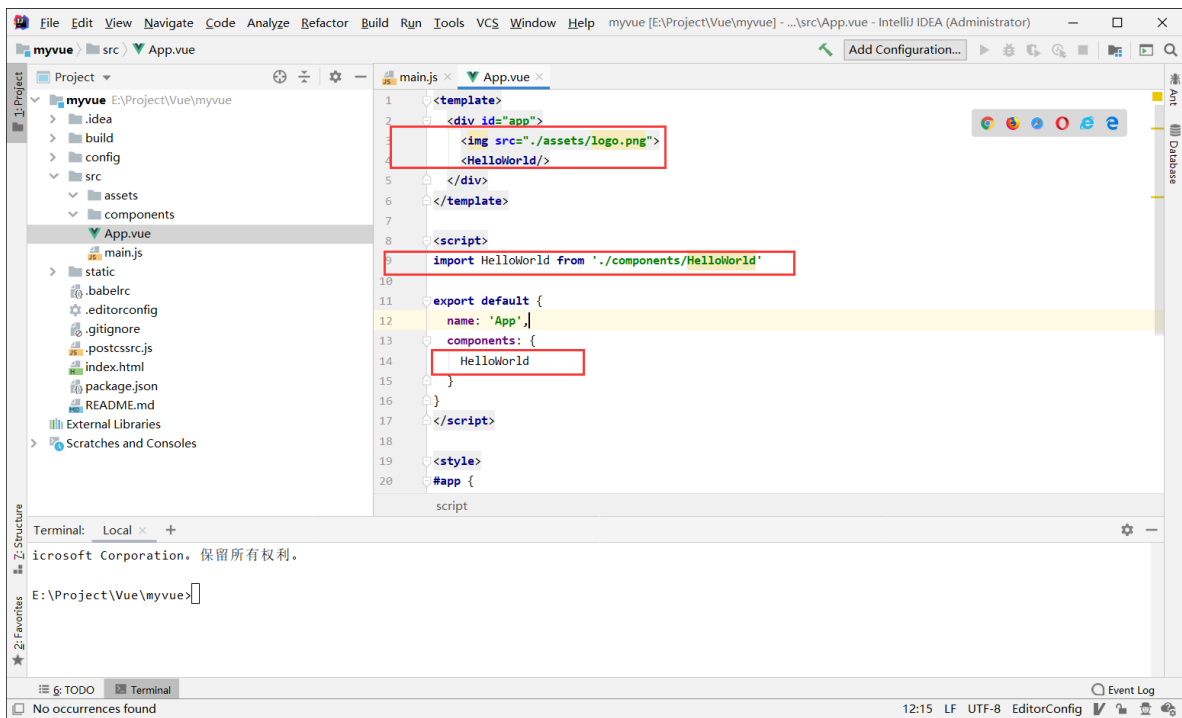
删除多余文件

一个干净的页面应该是什么也没有的，所以：

1. 删除图中标记的一张图，一个Vue组件，这些都是项目创建的时候自动添加的，但是并没有什么用



2. 进入到 `App.vue`，删除标出的部分



这样，一个纯净的项目就出来了

webpack 的使用

ES6

在讲解 `webpack`，之前，首先要了解一下ES6规范

EcmaScript6 标准增加了 JavaScript 语言层面的模块体系定义。

ES6 模块的设计思想，是尽量静态化，使编译时就能确定模块的依赖关系，以及输入和输出的变量。

但是现在的浏览器大多都只支持到ES5规范，所以我们需要一款工具来对其进行管理

什么是 `webpack`

`webpack` 是前端的模块化加载与构建管理工具，类似后端的 `maven`

它能把各种资源，如 JS、JSX、ES6、SASS、LESS、图片等都作为模块来处理和使用

使用 `webpack` 能够将ES6规范打包降级为ES5规范，从而使现在的浏览器支持

安装 `webpack`

```
1 npm install webpack -g
2 npm install webpack-cli -g
```

或者使用 `cnpm`

测试是否安装成功：

```
1 webpack -v
2 webpack-cli -v
```

```
C:\Users\Bean>webpack -v
4.41.6

C:\Users\Bean>webpack-cli -v
3.3.11
```

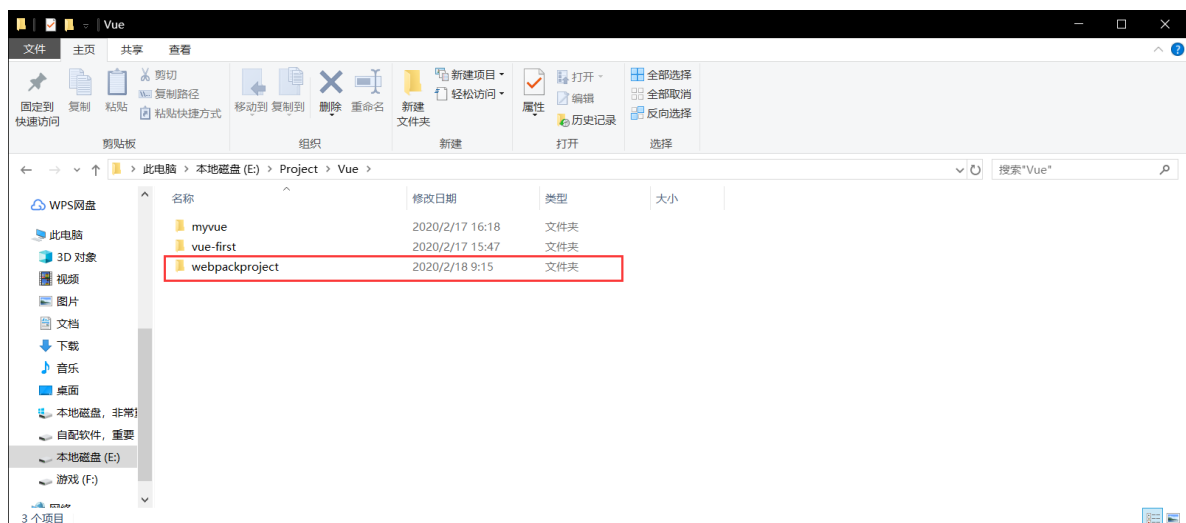
配置

创建 `webpack.config.js` 配置文件

- entry: 入口文件, 指定 `WebPack` 用哪个文件作为项目的入口
- output: 输出, 指定 `WebPack` 把处理完成的文件放置到指定路径
- module: 模块, 用于处理各种类型的文件
- plugins: 插件, 如: 热更新、代码重用等
- resolve: 设置路径指向
- watch: 监听, 用于设置文件改动后直接打包

起步

1. 首先创建一个项目, 比如创建一个文件夹, 然后用IDEA打开



2. 创建目录 `modules`, 用于存放 `js` 文件

3. 在 `modules` 下创建 `js` 文件 `hello.js` 与 `main.js`

`hello.js` 是模块化组件，并设置导出方法

```
1  /*这个exports命令是导出，是暴露接口，只有用了这个命令才能让别的文件导入
2  * 方法名字为sayhi
3  * */
4  exports.sayhi = function () {
5      document.write("<h1>Hello WebPack</h1>")
6  }
```

`main.js` 作为入口函数，并导入对应的方法，运行

```
1  /*注意，这里导入的是hello不是hello.js，相当于java中倒入类但是不导入.java文件一样
2  * 使用require命令导入对应的类(这里是hello.js)，但是导入hello不导入hello.js
3  * 注意引入要加./表示当前目录
4  * */
5  var hello = require("./hello");
6  /*导入之后就可以使用里面的方法了，因为我们只写了一个sayhi()方法，所以调用*/
7  hello.sayhi();
```

4. 设置 `webpack` 打包配置，使用 `webpack.config.js` 进行配置

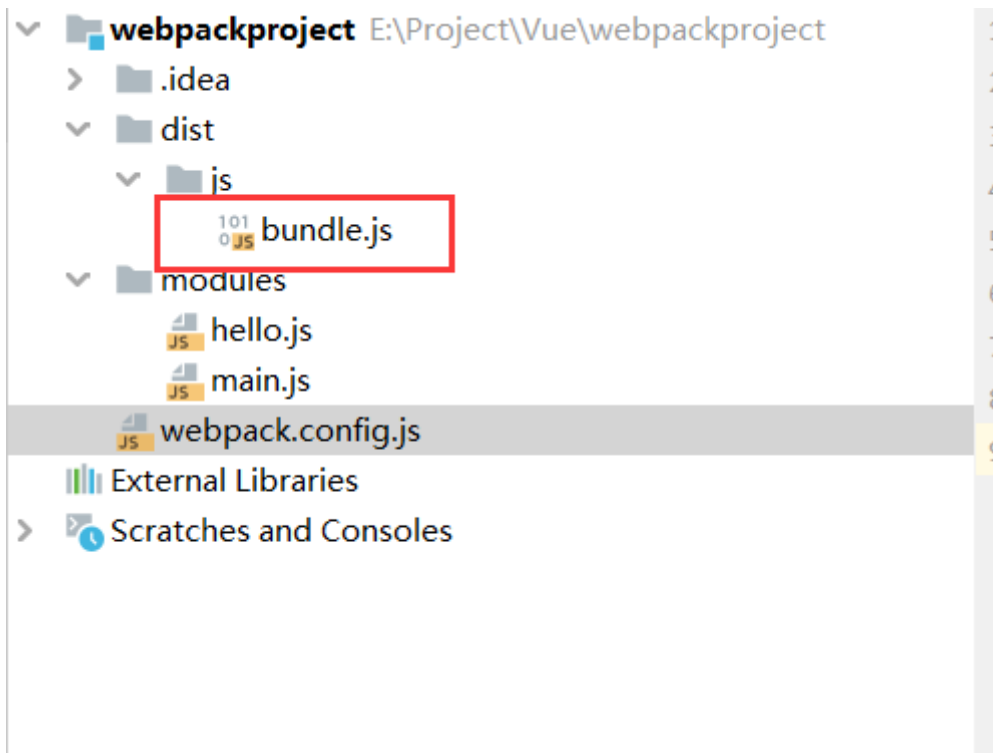
在上面已经有 `webpack.config.js` 的配置了，但是最重要的是输出位置还有入口方法

```
1  module.exports = {
2      /*注意，这里要精确到入口函数，这里写的是.js文件了
3      * 使用./表示当前目录下的xxx*/
4      entry: "./modules/main.js",
5      output: {
6          /*默认的约定是到这里去，这个文件名字*/
7          filename: "./js/bundle.js"
8      }
9  };
```

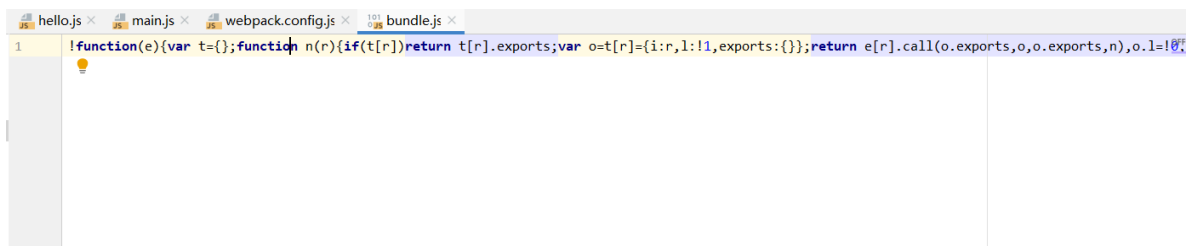
5. 在控制台使用命令 `webpack` 打包

```
E:\Project\Vue\webpackproject>webpack
Hash: 1caeb9420aa313ad655f
Version: webpack 4.41.6
Time: 101ms
Built at: 2020-02-18 9:31:19

    Asset      Size  Chunks             Chunk Names
./js/bundle.js 1020 bytes      0 [emitted]    main
Entrypoint main = ./js/bundle.js
```



打包之后的 `js` 文件



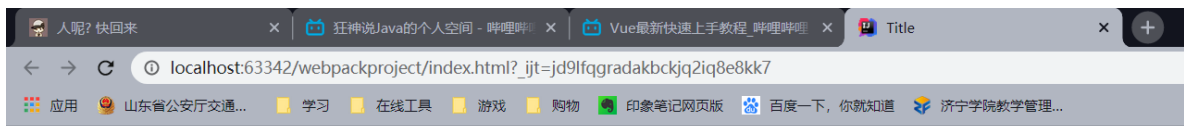
下面进行测试：

在项目目录下新建一个html文件index.html

引入bundle.js

测试





Hello WebPack

Vue 中安装插件

如何安装

懂了 `webpack` 之后，我们就可以使用它来安装插件了

插件有很多中，比如 `axios` 也是个插件，`vue-router` 也是个插件

现在我们以 `vue-router` 作为一个例子，讲解一下 `vue` 中如何安装插件

1. 使用 `npm` 将插件下载下来
2. 在项目中导入
3. 在项目中声明使用

vue-router 路由跳转

导入环境

1. 使用 `npm` 下载 `vue-router`：`npm install vue-router --save-dev`，这个是下载 `vue-router` 并加入到配置中

```
Terminal: Local x +
Microsoft Windows [版本 10.0.18362.657]
(c) 2019 Microsoft Corporation. 保留所有权利。

E:\Project\Vue\myvue>npm install vue-router --save-dev
√ Installed 1 packages
√ Linked 0 latest versions
√ Run 0 scripts
√ All packages installed (1 packages installed from npm registry, used 852ms(network 802ms), speed 156.76kB/s, json 1(8.44kB), tarball 117.28kB)

E:\Project\Vue\myvue>
```

2. 在 `main.js` 中导入 `vue-router`：`import VueRouter from 'vue-router'`

```
main.js x
1 import Vue from 'vue'
2 import App from './App'
3 import VueRouter from 'vue-router'
4 Vue.config.productionTip = false
5
6 new Vue({
7   el: '#app',
8   components: { App },
9   template: '<App/>'
10 })
11
```

这个 `VueRouter` 是导入之后在这个项目中的名字，from xxx才是路径

3. 声明 `vue-router`

```
main.js x
1 import Vue from 'vue'
2 import App from './App'
3 import VueRouter from 'vue-router'
4 Vue.config.productionTip = false
5
6 Vue.use(VueRouter);
7
8 new Vue({
9   el: '#app',
10   components: { App },
11   template: '<App/>'
12 })
13
```

至此，导入完成

使用 `vue-router`

首先我们要准备好几个页面，作为路由跳转，放到 `components` 文件夹下面

- `Content.vue`

```
1 <template>
2
3   <h1>Router-Content</h1>
4
5 </template>
```

```

6
7   <script>
8     export default {
9       name: "content"
10    }
11  </script>
12
13  <style scoped>
14
15  </style>

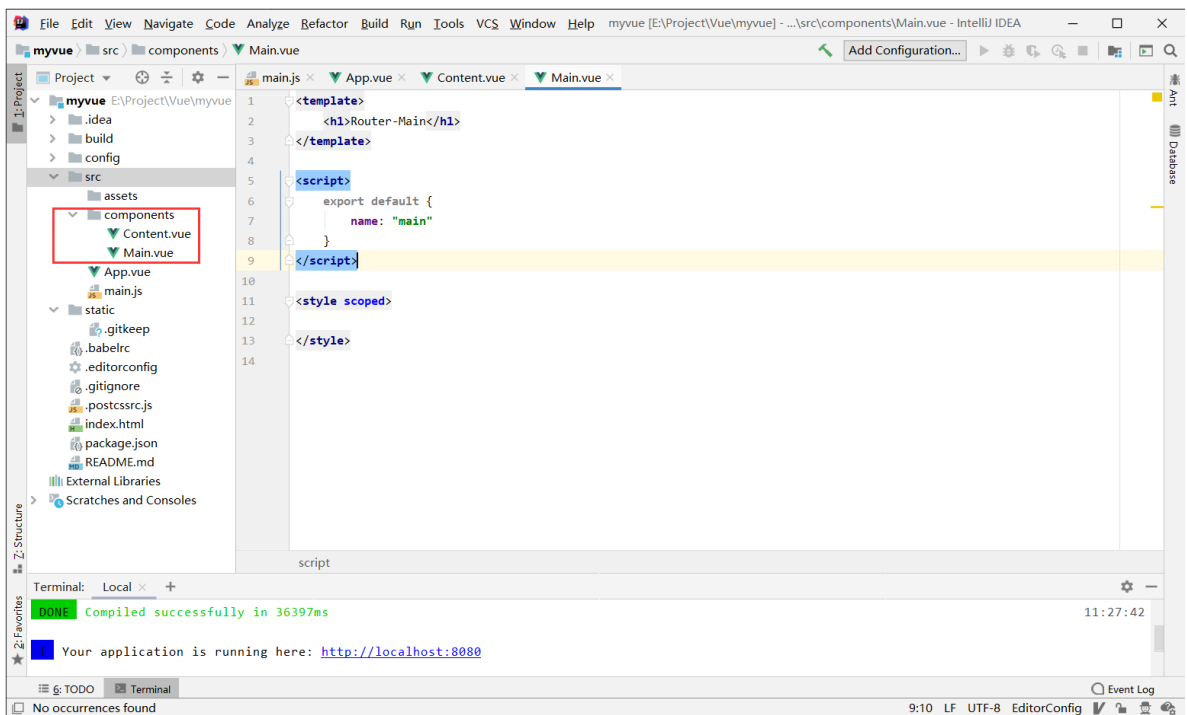
```

- Main.vue

```

1   <template>
2     <h1>Router-Main</h1>
3   </template>
4
5   <script>
6     export default {
7       name: "main"
8     }
9   </script>
10
11  <style scoped>
12
13  </style>

```



要使用 `vue-router`，只要三步：



1. 建立 `router` 文件夹，在文件夹下面创建 `index.js` 作为配置（`Vue` 官方默认的配置文件名就是 `index`）

- `router/index.js`

```
1 //首先导入vue组件，没了vue什么也干不了
2 import Vue from 'vue';
3 //然后导入vue-router
4 import Router from 'vue-router';
5 //导入要跳转的组件content
6 import Content from "../components/Content";
7 //导入要跳转的组件Main
8 import Main from "../components/Main";
9
10 //使用Router
11 Vue.use(Router);
12
13 /*导出，让main.js可以使用
14 * new Router，生成一个新的Router，这个Router就是前面哪个导入的vue-router，在这里叫Router
15 * */
16 export default new Router({
17   /*配置路由*/
18   routes: [
19     {
20       //请求路径，到时候就是localhost:8080/content
21       path: '/content',
22       //路由名称
23       name: 'Content',
24       //跳转的组件
25       component: Content
26     },
27     {
28       path: '/main',
29       name: 'Main',
30       component: Main
31     }
32   ]
33 });
```

2. 在 `main.js` 中配置

- `main.js`

```
1 import Vue from 'vue'
2 import App from './App'
3
4 //注意这里导入了router，但是没有精确到哪一个文件，这是因为只要名字是默认的index.js就会自动导入，不必精确到具体的文件
5 import router from './router'
6
7 Vue.config.productionTip = false
8
9 Vue.use(router);
10
11 new Vue({
12   el: '#app',
13   //这里直接配置路由即可，但是假如导入的名字不是router而是其他的，比如Router
14   // 就要这么配置： router: Router
```

```
15     router,  
16     components: { App },  
17     template: '<App/>  
18   })
```

3. 在 `App.vue` 中设置

```
1  <template>  
2    <div id="app">  
3      <h1>Vue-Router</h1>  
4      <!--  
5        router-link: 跳转的链接  
6        to: 跳转的路径  
7        router-view: 跳转的组件显示  
8      -->  
9      <router-link to="/content">内容</router-link>  
10     <router-link to="/main">首页</router-link>  
11     <router-view></router-view>  
12  
13   </div>  
14 </template>  
15 <script>  
16  
17 export default {  
18   name: 'App'  
19 }  
20 </script>  
21  
22 <style>  
23 #app {  
24   font-family: 'Avenir', Helvetica, Arial, sans-serif;  
25   -webkit-font-smoothing: antialiased;  
26   -moz-osx-font-smoothing: grayscale;  
27   text-align: center;  
28   color: #2c3e50;  
29   margin-top: 60px;  
30 }  
31 </style>
```

Vue 实战快速上手

环境搭建

1. 创建工程 `hello-vue` : `vue init webpack hello-vue`
2. 安装依赖: `vue-router` , `element-ui` , `sass-loader` , `node-sass`

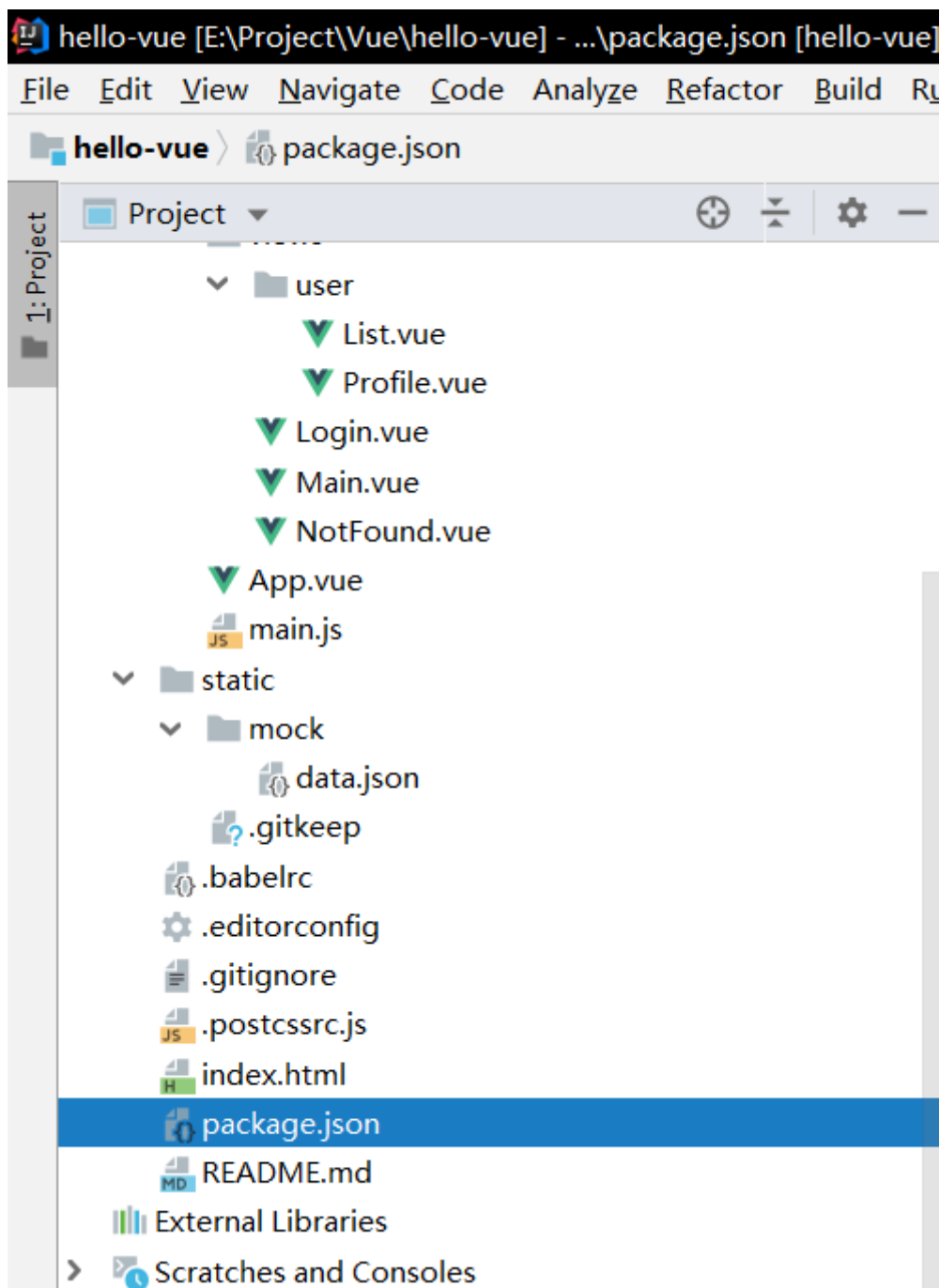
```
1 # 进入工程目录
2 cd hello-vue
3 # 安装 vue-router
4 cnpm install vue-router --save-dev
5 # 安装 element-ui
6 cnpm i element-ui -S
7 # 安装依赖
8 cnpm install
9 # 安装 SASS 加载器, 这个用于解析css的
10 cnpm install sass-loader node-sass --save-dev
11 # 启动测试
12 npm run dev
```

3. npm 命令解释:

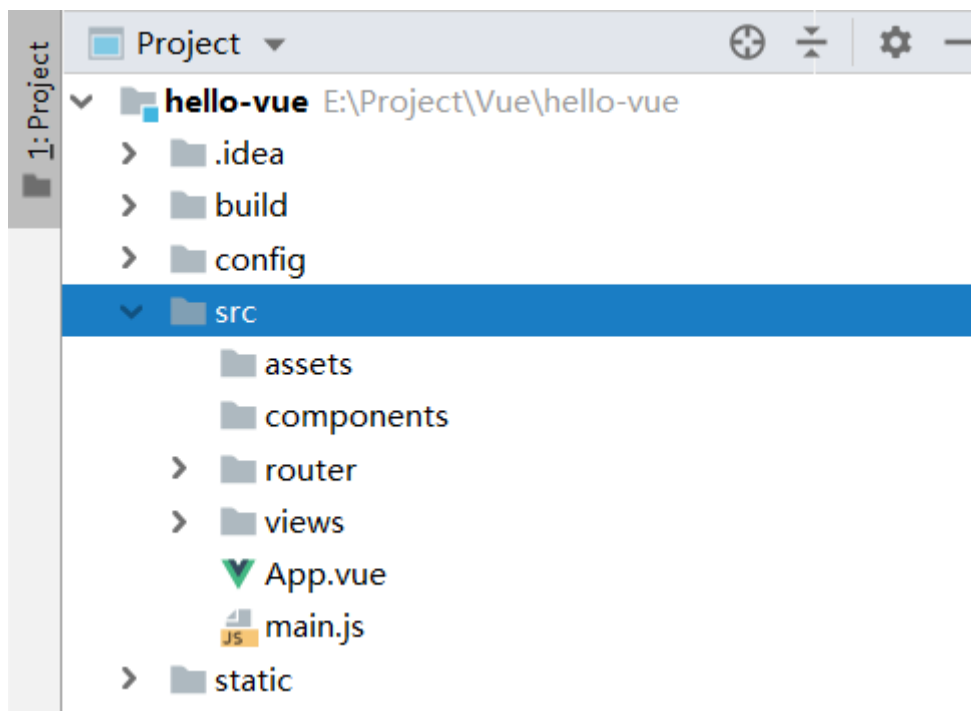
- `npm install moduleName` : 安装模块到项目目录下
- `npm install -g moduleName` : -g 的意思是将模块安装到全局, 具体安装到磁盘哪个位置, 要看 `npm config prefix` 的位置
- `npm install -save moduleName` : --save 的意思是将模块安装到项目目录下, 并在 `package` 文件的 `dependencies` 节点写入依赖, -S 为该命令的缩写
- `npm install -save-dev moduleName` : --save-dev 的意思是将模块安装到项目目录下, 并在 `package` 文件的 `devDependencies` 节点写入依赖, -D 为该命令的缩写

这里可能会有报错, 这可能是因为 `sass` 版本过高导致的, 降级为: `"sass-loader": "^7.3.1",`, 然后 `cnpm install`

文件在 `package.json`



起步



- static: 静态资源
- views: 视图组件
- router: 路由跳转
- components: 组件
- assets: 资源

- views/main.vue

```
1 <template>
2   <h1>首页</h1>
3 </template>
4
5 <script>
6   export default {
7     name: "main"
8   }
9 </script>
10
11 <style scoped>
12
13 </style>
```

- views/Login.vue : 这个来自 ElementUI , 我一个字没写

```
1 <template>
2   <div>
3     <el-form ref="loginForm" :model="form" :rules="rules" label-width="80px" class="login-
4       box">
5       <h3 class="login-title">欢迎登录</h3>
6       <el-form-item label="账号" prop="username">
7         <el-input type="text" placeholder="请输入账号" v-model="form.username"/>
8       </el-form-item>
9       <el-form-item label="密码" prop="password">
10        <el-input type="password" placeholder="请输入密码" v-model="form.password"/>
11      </el-form-item>
```

```
12     <el-button type="primary" v-on:click="onSubmit('loginForm')">登录</el-button>
13   </el-form-item>
14 </el-form>
15
16 <el-dialog
17   title="温馨提示"
18   :visible.sync="dialogVisible"
19   width="30%"
20   :before-close="handleClose">
21   <span>请输入账号和密码</span>
22   <span slot="footer" class="dialog-footer">
23     <el-button type="primary" @click="dialogVisible = false">确 定</el-button>
24   </span>
25 </el-dialog>
26 </div>
27 </template>
28
29 <script>
30   export default {
31     name: "Login",
32     data() {
33       return {
34         form: {
35           username: '',
36           password: ''
37         },
38
39         // 表单验证, 需要在 el-form-item 元素中增加 prop 属性
40         rules: {
41           username: [
42             {required: true, message: '账号不可为空', trigger: 'blur'}
43           ],
44           password: [
45             {required: true, message: '密码不可为空', trigger: 'blur'}
46           ]
47         },
48
49         // 对话框显示和隐藏
50         dialogVisible: false
51       }
52     },
53     methods: {
54       onSubmit(formName) {
55         // 为表单绑定验证功能
56         this.$refs[formName].validate((valid) => {
57           if (valid) {
58             // 使用 vue-router 路由到指定页面, 该方式称之为编程式导航
59             this.$router.push("/main");
60           } else {
61             this.dialogVisible = true;
62             return false;
63           }
64         });
65       }
66     }
67   }
68 </script>
69
```

```

70 <style lang="scss" scoped>
71   .login-box {
72     border: 1px solid #DCDFE6;
73     width: 350px;
74     margin: 180px auto;
75     padding: 35px 35px 15px 35px;
76     border-radius: 5px;
77     -webkit-border-radius: 5px;
78     -moz-border-radius: 5px;
79     box-shadow: 0 0 25px #909399;
80   }
81
82   .login-title {
83     text-align: center;
84     margin: 0 auto 40px auto;
85     color: #303133;
86   }
87 </style>

```

- router/index.js

```

1   import Vue from 'vue'           //引入VUE
2   import Router from 'vue-router' //引入vue-router
3
4   import Main from '../views/main' //引入main页面
5   import Login from '../views/Login' //引入Login页面
6
7   Vue.use(Router);                //使用router
8
9
10  //导出路由组件
11  export default new Router({
12    routes: [
13      {
14        path: '/main',
15        component: Main
16      }, {
17        path: '/login',
18        component: Login
19      }
20    ]
21  });

```

- main.js

```

1
2   import Vue from 'vue'
3   import App from './App'
4
5   //引入路由
6   import router from './router'
7
8
9   //elementUI导入，从官网的快速起步看出
10  import ElementUI from 'element-ui';

```

```

11 import 'element-ui/lib/theme-chalk/index.css';
12
13 //使用路由
14 Vue.use(router);
15
16 //使用element-ui, 从官网的快速起步看出
17 Vue.use(ElementUI);
18
19
20 new Vue({
21   el: '#app',
22   router,
23   render: h=>h(App) //ElementUI, 官网的快速起步
24 })

```

- App.vue

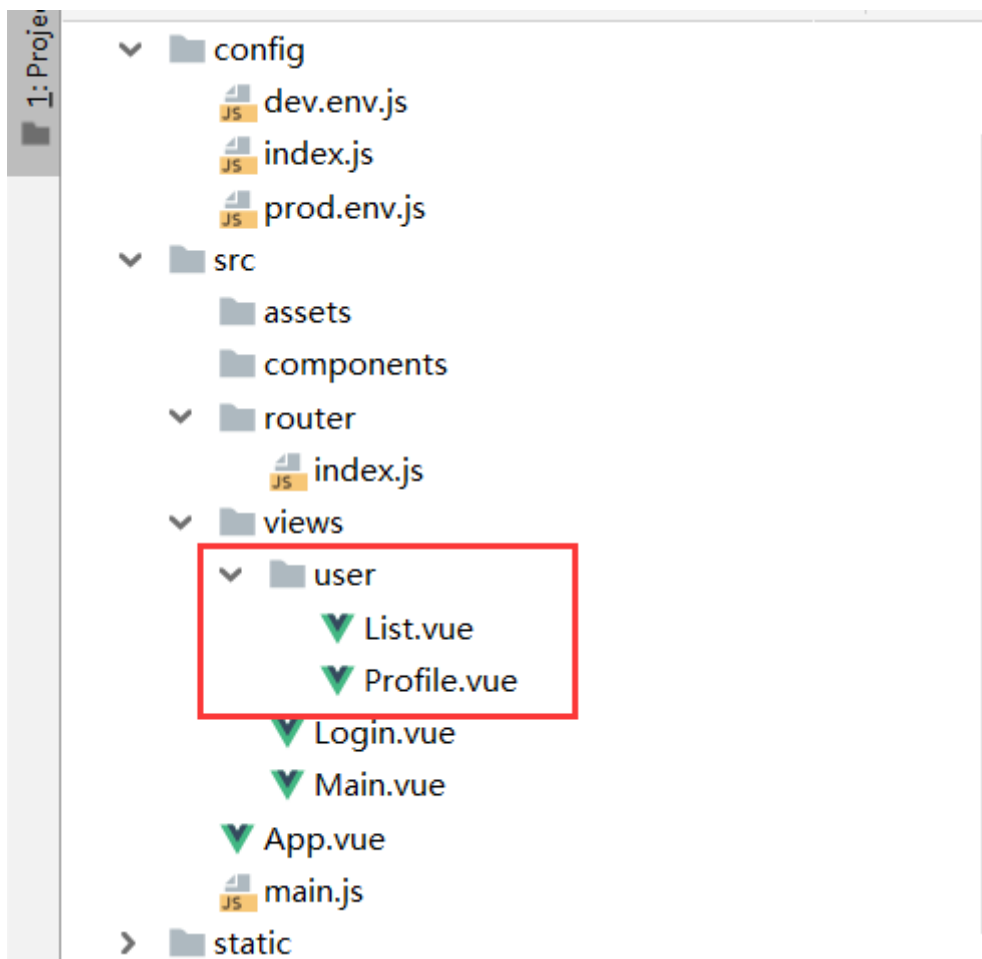
```

1 <template>
2   <div id="app">
3     <router-link to="/login">login</router-link>
4     <router-view></router-view>
5   </div>
6 </template>
7
8 <script>
9
10 export default {
11   name: 'App',
12 }
13 </script>
14
15 <style>
16 #app {
17   font-family: 'Avenir', Helvetica, Arial, sans-serif;
18   -webkit-font-smoothing: antialiased;
19   -moz-osx-font-smoothing: grayscale;
20   text-align: center;
21   color: #2c3e50;
22   margin-top: 60px;
23 }
24 </style>

```

路由嵌套

文件更新:



- `views/user/List.vue`

```
1 <template>
2   <h1>用户列表</h1>
3 </template>
4
5 <script>
6   export default {
7     name: "List"
8   }
9 </script>
10
11 <style scoped>
12
13 </style>
```

- `views/user/Profile.vue`

```

1   <template>
2     <h1>个人信息</h1>
3   </template>
4
5   <script>
6     export default {
7       name: "Profile"
8     }
9   </script>
10
11  <style scoped>
12
13  </style>

```

- routers/index.js

```

1   import Vue from 'vue'           //引入VUE
2   import Router from 'vue-router' //引入vue-router
3
4   import Main from '../views/Main' //引入main页面
5   import Login from '../views/Login' //引入Login页面
6
7   import UserList from '../views/user/List'
8   import UserProfile from '../views/user/Profile'
9
10  Vue.use(Router);                //使用router
11
12
13  //导出路由组件
14  export default new Router({
15    routes: [
16      {
17        path: '/main',
18        component: Main,
19        //嵌套路由, Main下还能跳转
20        children: [
21          {
22            path: '/user/profile',
23            component: UserProfile
24          }, {
25            path: '/user/list',
26            component: UserList
27          }
28        ]
29      }, {
30        path: '/login',
31        component: Login,
32      }
33    ]
34  });

```

重点来了：路由嵌套，在main路由下嵌套另一个路由

- Main.vue：只显示部分

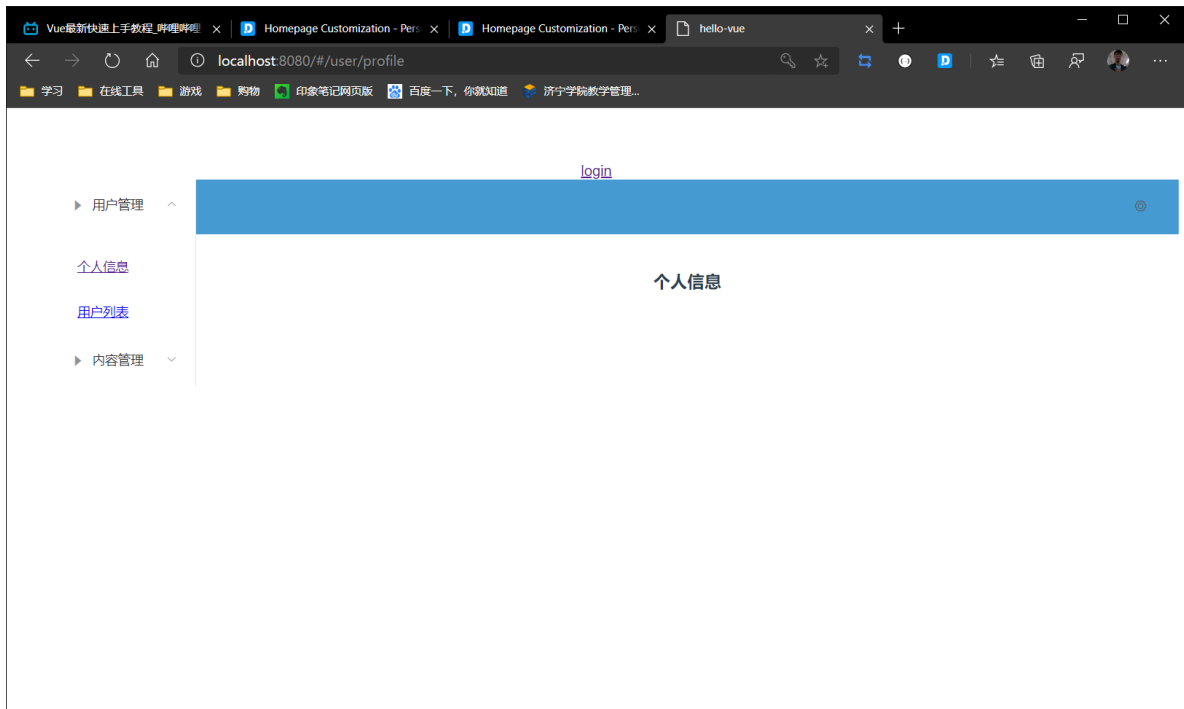
```

1   <template>
2
3   <router-link to="/user/profile">个人信息</router-link>
4   <router-link to="/user/list">用户列表</router-link>
5   <router-view />
6
7   </template>

```

Main.vue 更改之后的效果

最终效果



参数传递和重定向

方法一：

- Main.vue

```

1   <router-link :to="{name: '/user/profile', params: {id:1}}">个人信息</router-link>

```

注意这里使用to来发送参数

- name : 传递地址
- params : 传递参数

注意：因为 vue 是双向绑定的，所以使用 v-bind:to="xxx" 绑定

- router/index.js

```

1   import Vue from 'vue'           //引入VUE
2   import Router from 'vue-router' //引入vue-router
3
4   import Main from '../views/Main' //引入main页面

```

```

5   import Login from '../views/Login'           //引入Login页面
6
7   import UserList from '../views/user/List'
8   import UserProfile from '../views/user/Profile'
9
10  Vue.use(Router);                             //使用router
11
12
13  //导出路由组件
14  export default new Router({
15    routes: [
16      {
17        path: '/main',
18        component: Main,
19        //嵌套路由, Main下还能跳转
20        children: [
21          {
22            path: '/user/profile/:id',
23            component: UserProfile
24          }
25        ]
26      }
27    ]
28  });

```

注意这里要接受一个id, 使用 `/:id` 传递参数

- Profile.vue

```

1   <template>
2     <div>
3       <h1>个人信息</h1>
4       <!-- 接受参数, 注意这里是route, 不是router-->
5       {{ $route.params.id }}
6     </div>
7   </template>
8
9   <script>
10    export default {
11      name: "Profile"
12    }
13  </script>
14
15  <style scoped>
16
17  </style>

```

注意, 所有东西都在组件下, 比如 `div`

方法二:

- Main.vue

```

1   <router-link :to="{name: '/user/profile', params: {id:1}}">个人信息</router-link>

```

- index.js

```

1  import Vue from 'vue'           //引入VUE
2  import Router from 'vue-router' //引入vue-router
3
4  import Main from '../views/Main' //引入main页面
5  import Login from '../views/Login' //引入Login页面
6
7  import UserList from '../views/user/List'
8  import UserProfile from '../views/user/Profile'
9
10 Vue.use(Router);                //使用router
11
12
13 //导出路由组件
14 export default new Router({
15   routes: [
16     {
17       path: '/main',
18       component: Main,
19       //嵌套路由, Main下还能跳转
20       children: [
21         {
22           path: '/user/profile/:id',
23           component: UserProfile,
24           props: true
25         }
26       ]
27     }
28   ]
29 });

```

这里令 `props` 为true, 但是在 `path` 还是要接受参数

如果要接受多个, 比如:

```
'/user/profile/:id/:name/:age'
```

- `Profiles.vue`

```

1  <template>
2    <div>
3      <h1>个人信息</h1>
4      <!-- 接受参数, 注意这里是route, 不是router-->
5      {{id}}
6    </div>
7  </template>
8
9  <script>
10    export default {
11      name: "Profile",
12      props: ['id']
13    }
14  </script>
15
16  <style scoped>
17
18  </style>

```

这样的方式我们更加容易接受

解决两个方法出现的问题：

这两个方法都会出现一个问题：

直接点击链接的时候画面就没了，但是直接上链接：

`http://localhost:8080/#/user/profile/1` 可以显示

这样的结果显然不是我们想要的，所以更改一下：

```
1  import Vue from 'vue'           //引入VUE
2  import Router from 'vue-router' //引入vue-router
3
4  import Main from '../views/Main' //引入main页面
5  import Login from '../views/Login' //引入Login页面
6
7  import UserList from '../views/user/List'
8  import UserProfile from '../views/user/Profile'
9
10 Vue.use(Router);                //使用router
11
12
13 //导出路由组件
14 export default new Router({
15   routes: [
16     {
17       path: '/main',
18       component: Main,
19       //嵌套路由，Main下还能跳转
20       children: [
21         {
22           path: '/user/profile/:id',
23           name: UserProfile,
24           component: UserProfile,
25         }, {
26           path: '/user/list',
27           name: UserList,
28           component: UserList
29         }
30       ]
31     }, {
32       path: '/login',
33       component: Login,
34     }
35   ]
36 });
```

可以很清楚的看到这里增加了一个 `name` 属性

- `Main.vue`

```
1 <router-link :to="{name: UserProfile,params: {id:1}}">个人信息</router-link>
```

使用了 `name` 属性之后就可以了

重定向

```
1  import Vue from 'vue'           //引入VUE
2  import Router from 'vue-router' //引入vue-router
3
4  import Main from '../views/Main' //引入main页面
5  import Login from '../views/Login' //引入Login页面
6
7  import UserList from '../views/user/List'
8  import UserProfile from '../views/user/Profile'
9
10 Vue.use(Router);                 //使用router
11
12
13 //导出路由组件
14 export default new Router({
15   routes: [
16     {
17       path: '/main',
18       component: Main,
19     }, {
20       path: '/login',
21       component: Login,
22     }, {
23       path: '/gohome',
24       redirect: '/login'
25     }
26   ]
27 });
```

- Main.vue

```
1      <el-submenu index="1">
2        <template slot="title"><i class="el-icon-caret-right"></i>用户管理</template>
3
4      <el-menu-item-group>
5
6        <el-menu-item index="1-1">
7          <router-link :to="{name: UserProfile,params: {id:1}}">个人信息</router-link>
8        </el-menu-item>
9
10       <el-menu-item index="1-2">
11         <router-link to="/user/list">用户列表</router-link>
12       </el-menu-item>
13
14       <el-menu-item index="1-3">
15         <router-link to="/gohome">回到登录界面</router-link>
16       </el-menu-item>
17
18     </el-menu-item-group>
```

404和路由钩子

路由模式

路由模式有两种：

- hash：路径带 # 号
- history：路径不带 # 号

修改路由配置：

```
1 export default new Router({
2   mode: 'history',
3   routes: []
```

注意传递字符串

404

- views/NotFound.vue

```
1 <template>
2   <div>
3     <h1>你的页面走丢了</h1>
4   </div>
5 </template>
6
7 <script>
8   export default {
9     name: "NotFound"
10  }
11 </script>
12
13 <style scoped>
14
15 </style>
```

- index.js


```

1 //导出路由组件
2 export default new Router({
3   mode: 'history',
4   routes: [
5     {
6       path: '*',
7       component: NotFound
8     }
9   ]
10 });

```

因为优先级最低，所以除了其他配置的页面，就进入这个页面

钩子与异步请求

钩子

```

1 <template>
2   <div>
3     <h1>个人信息</h1>
4     <!-- 接受参数，注意这里是route，不是router-->
5     {{id}}
6   </div>
7 </template>
8
9 <script>
10   export default {
11     name: "Profile",
12
13     /*
14     这个就和过滤器一样，参数是固定的
15     to:路由将要跳转的路径信息
16     from:路径跳转前的路径信息
17     next: 路由的控制参数
18     next(): 跳转到下一个界面
19     next('/path'): 改变路由的跳转方向，令其跳到另外的路由
20     next(false): 返回原来的页面
21     next((vm)=>{}): 仅在beforeRouteEnter可用，vm是组件实例
22     */
23     beforeRouteEnter: (to, from, next) => { //进入路由之前进行
24       console.log("进入路由之前执行");
25       next(); //要使用这个next，否则会卡住
26
27     },
28     beforeRouteLeave: (to, from, next) => { //进入路由之后进行
29       console.log("进入路由之后执行");
30       next();
31     }
32   }
33 </script>
34
35 <style scoped>
36
37
38 </style>

```

异步请求

首先安装 `axios`

```
cnpm install axios -s
```

然后在 `main.js` 里面引入

```
1 //引入axios
2 import axios from 'axios'
3 import VueAxios from 'vue-axios'
4 //使用Axios
5 Vue.use(VueAxios, axios);
```

- `UserProfile.vue`

```
1 <template>
2   <div>
3     <h1>个人信息</h1>
4     <!-- 接受参数，注意这里是route，不是router-->
5   </div>
6 </template>
7
8 <script>
9   export default {
10     name: "Profile",
11
12     /*
13      这个就和过滤器一样，参数是固定的
14      to:路由将要跳转的路径信息
15      from:路径跳转前的路径信息
16      next: 路由的控制参数
17      next(): 跳转到下一个界面
18      next('/path'): 改变路由的跳转方向，令其跳到另外的路由
19      next(false): 返回原来的页面
20      next((vm)=>{}): 仅在beforeRouteEnter可用，vm是组件实例
21    */
22     beforeRouteEnter: (to, from, next) => { //进入路由之前进行
23       //进入路由之前加载数据，执行这个方法
24       next(vm => {
25         vm.getData()
26       }); //要使用这个next，否则会卡住
27
28     },
29     beforeRouteLeave: (to, from, next) => { //进入路由之后进行
30       next();
31     },
32     methods: {
33       getData: function () {
34         this.axios({
35           method: 'get',
36           url: 'http://localhost:8080/static/mock/data.json'
37         }).then(function (response) {
```

```
38         console.log(response)
39     })
40 }
41 }
42
43 }
44 </script>
45
46 <style scoped>
47
48 </style>
```