## 什么是Vue

Vue是一个构建用户界面(UI)的渐进式JavaScript框架。

官方网址：<https://cn.vuejs.org/>

Vue作者：尤雨溪/中国无锡

## 库和框架的区别

### 库(Library)

本质上是一些函数的集合。每次调用函数，实现一个特定的功能，接着把控制权交给使用者。

代表：**jQuery**

jQuery这个库的核心：DOM操作，即：封装DOM操作，简化DOM操作。

### 框架(framework)

是一套完整的解决方案，使用框架的时候，需要把你的代码放到框架合适的地方，框架会在合适的时机调用你的代码。

代表：**vue**

1. 框架规定了自己的编程方式，是一套完整的解决方案
2. 使用框架的时候，由框架控制一切，我们只需要按照规则写代码
3. 框架的侵入性很高(从头到尾)

## MVVM介绍

**MVVM**是一种更好的UI模式解决方案，MVVM通过数据双向绑定让数据自动地双向同步。类似MVC，本质上是MVC的一种改进版本。核心概念分为：M / V / VM

1. M：model数据模型。（后端传递的数据，例如：JSON）
2. V：view视图。（数据渲染的页面，例如：HTML元素）
3. VM：ViewModel视图模型（MVVM核心，连接V和M的桥梁）



**什么是双向绑定？（重点）**

1）将M转化成V，即将后端传递的数据转化成所看到的页面。实现的方式是：数据绑定；

2）将V转化成M，即将所看到的页面转化成后端的数据。实现的方式是：DOM 事件监听；

理解：之前的数据交互方式，是通过ajax将json（M）数据绑定到HTML元素（V）上，而现在中间多了一层叫视图模型（VM），将M层和V层结合起来，从而实现数据的自动绑定（双向绑定）。

补充说明：

注1：虽然没有完全遵循MVVM模型，Vue的设计无疑受到了它的启发。另外一个js框架“knockout”完全遵循MVVM模型

注2：学习Vue要转化思想：“不要在想着怎么操作DOM，而是想着如何操作数据！！！”

## Vue

### 创建Vue项目

使用HBuilder创建一个空白的项目，如下图：



### 安装Vue

#### 手动下载

|  |
| --- |
| *<!-- 开发环境版本，包含了有帮助的命令行警告(推荐) -->* <**script type="text/javascript" src="js/vue.min.js"**></**script**> *<!-- 生产环境版本，优化了尺寸和速度 -->* <**script type="text/javascript" src="js/vue.js"**></**script**> |

#### cdn下载(需连接网络)【推荐】

|  |
| --- |
| *<!-- 开发环境版本，包含了有帮助的命令行警告(推荐) -->* <**script type="text/javascript" src="https://cdn.bootcss.com/vue/2.6.10/vue.min.js"**></**script**> *<!-- 生产环境版本，优化了尺寸和速度 -->* <**script type="text/javascript" src="https://cdn.jsdelivr.net/npm/vue"**></**script**> |

**请在这里讲述CDN是什么、有什么优点等！！！**

**请在这里讲述CDN是什么、有什么优点等！！！**

**请在这里讲述CDN是什么、有什么优点等！！！**

#### 什么是CDN

CDN是构建在网络之上的内容分发网络，依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块， 使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。CDN的关键技术主要有内容存储和分发技术。

**CND概况：**

1）CDN的全称是Content Delivery Network，即内容分发网络。

2）CND加速主要是加速静态资源，如网站上面上传的图片、媒体，以及引入的一些Js、css等文件。

3）CND加速需要依靠各个网络节点，例如100台CDN服务器分布在全国范围，从上海访问，会从最近的节点返回资源，这是核心。

4）CND服务器通过缓存或者主动抓取主服务器的内容来实现资源储备

5）CDN基本原理：将源站内容分发至最接近用户的节点，使用户可就近取得所需内容，提高用户访问的响应速度和成功率。

理解：如阿里、腾讯、网页等互联网运行商，在访问它们的数据资源（图片、媒体以及js/css文件）时有没有出现过数据加载延迟情况？答案是没有，为什么？因为CDN，在你访问阿里的官网的时候，阿里官网是部署在杭州总部的服务器上的，在你访问的时候，根据CDN的特点，先到本地CDN服务器中去找到相应资源，没有再到杭州服务器上去找（下载到本地），下次即可直接从本地CDN服务器上获取即可。

**特点：就近匹配原则、减少网络阻塞、提供响应速度和成功率。**

#### 什么是BootCDN

BootCDN是Bootstrap中文网支持并维护的前端开源项目免费CDN服务，致力于为Bootstrap、jQuery、Angular、Vuejs一样优秀的前端开源项目提供稳定、快速的免费 CDN 加速服务。

官网：<https://www.bootcdn.cn/>

* **使用cdn的好处：**

1. 不用担心自己网站访客，在任何时间，任何地点，任何网络运营商，都能快速打开网站，说白了就是把你的资源交给第三方托管，它快你的网站访问就快，并且不需要消耗你的网站流量，提升访问速度。
2. 各种服务器虚拟主机带宽等采购成本，包括后期运维成本都会大大减少。
3. 给网站直接带来的好处就是：流量，咨询量，客户量，成单量，都会得到大幅度提升。

想想看，一个打开速度很快的网站和一个打开速度很慢的网站，用户的体验是完全不一样的，那么哪些群体使用更广泛呢？

* **使用cdn的坏处：**

说完好处也要说说坏处了，对于不使用cdn的网站来说，它崩任他崩，我自坐如松。对于少数资源依赖cdn的网站来说，可能仅影响体验，比如页面显示、js事件等，但对于严重依赖cdn的网站来说，只能说同归于尽了。。。

注1：类似的网站还有国外公共静态资源库：unpkg

**理解：这个BootCDN类似于Maven中的国内中央仓库地址，连接速度较快。**

### Vue实例

#### 语法

|  |
| --- |
| **var vm = new Vue({**  **el：'#ID' // DOM 元素，挂载视图模型，**  **data：{}, // 定义属性，并设置初始值**  **methods：{} // 定义方法，用于事件交互时使用的函数**  **});** |

**补充说明（非常重要）：**

1. 每个Vue应用都是通过用Vue构造器创建一个新的Vue实例开始的；
2. 通过Vue实例的el属性指定vue管理内容区域，通常我们也把它叫做边界，这意味着我们接下来的改动全部在指定的div内，div外部不受影响；
3. 定义Vue实例的常用参数：el(展示数据的DOM元素)、data(挂载属性)、methods(事件方法)；
4. {{ }} 用于输出对象属性和函数返回值；

#### 第一个Vue实例

|  |
| --- |
| <**div id="app"**>  <**h1**>msg={{msg}},ts={{ts}}</**h1**>  <**button v-on:click="aaa"**>确定</**button**>  </**div**> <**div id="dd"**>  <**h1**>msg={{msg}},ts={{ts}}</**h1**> </**div**>  -------------------------------------------------------------------------------------------  <**script**>  **var *vue***=**new** Vue({  **el**:**'#app'**,  ***//data方式一：简单写法***  */\* data:{  msg:'hello Vue!',  ts:new Date().getTime()  }, \*/*  ***//data方式二：在组件开发中的写法必须是一个函数(效果与方式一相同)*** data:**function**(){  **return** {  **msg**:**'hello Vue!'**,  **ts**:**new *Date***().getTime()  };  },  *//定义方法事件* **methods**:{  *//指定方法名,随意!!!  //在HTML元素中使用v-指定的方式,访问定义事件* aaa:**function**(){  *alert*(111);  }  }  }); </**script**> |

注意：

1. data属性在组件开发中的写法必须是一个函数；
2. Vue指令指的是带有“v-”前缀的特殊属性；
3. 用v-on:click指令绑定单机事件；

### 双向数据绑定

#### 示例一

**案例：使用v-model和v-on:click实现数据双向绑定**

|  |
| --- |
| <**div id="app"**>  <**h1**>msg={{msg}},ts={{ts}}</**h1**>  *<!--1.示例一：使用v-model和doclick实现Vue的数据双向绑定-->* <**input id="count" type="text" v-model="count"** />  <**button v-on:click="doClick"**>确定</**button**>  </**div**>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#app'**,  **data**:{  **count**:10,  **msg**:**'hello Vue!'**,  **ts**:**new *Date***().getTime()  },  **methods**:{  doClick:**function**(){  ***console***.log(111);  }  }  });</**script**> |

#### 示例二

**案例：只有当实例被创建时data中存在的属性才是响应式的**

|  |
| --- |
| <**div id="app"**>  <**h1**>msg={{msg}},ts={{ts}}</**h1**>  *<!--2.示例二：只有当实例被创建时data中存在的属性才是响应式的-->  <!-- <div>{{name}}</div> -->* </**div**>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#app'**,  **data**:{  **count**:10,  **msg**:**'hello Vue!'**,  **ts**:**new *Date***().getTime()  },  **methods**:{  doClick:**function**(){  ***console***.log(111);  }  }  });  ***vm***.**name**=**"zs"**;</**script**> |

#### 示例三

**案例：**

1. **用v-model指令在表单控件元素上创建双向数据绑定；**
2. **this在methods属性的方法里指向当前Vue实例；**

**外部访问：vm.name/vm.$data.name**

**内部访问：this.name**

|  |
| --- |
| <**div id="app"**>  <**h1**>msg={{msg}},ts={{ts}}</**h1**>  *<!--  3.示例三：  1）用v-model指令在表单控件元素上创建双向数据绑定  2）this在methods属性的方法里指向当前Vue实例   -->* <**button v-on:click="doIncrement"**>+</**button**><**br**/> </**div**>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#app'**,  **data**:{  **count**:10,  **msg**:**'hello Vue!'**,  **ts**:**new *Date***().getTime()  },  **methods**:{  doIncrement:**function**(){  *//内部访问(this)：这里this代表当前Vue实例对象* **this**.**count**++;  *//外部访问(Vue实例名)* ***console***.log(**"count=%s"**,***vm***.**count**);  },  doClick:**function**(){  ***console***.log(111);  ***console***.log(**"count=%s"**,**this**.**count**);  }  }  });  </**script**> |

#### 示例四

**案例：v-once单向数据绑定方式，用处不大！！！**

|  |
| --- |
| <**div id="app"**>  <**h1**>msg={{msg}},ts={{ts}}</**h1**>  *<!--4.示例四：v-once单向数据绑定方式，用处不大！！-->* <**div v-once=""**><**label**>v-once单向数据绑定方式：</**label**>count={{count}}</**div**> </**div**> |

### **实例生命周期钩子**

每个Vue实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到DOM并在数据变化时更新DOM等。同时在这个过程中也会运行一些叫做生命周期钩子的函数，这给了用户在不同阶段添加自己的代码的机会。**这个过程类似于Servlet生命周期及相关方法。**

**Vue生命周期图：**

****

# 第二天：Vue之基础语法01

1. 课前准备
2. 插值
3. 指令
4. 过滤器
5. 计算属性
6. 监听属性

## 课前准备

1. Date.js

## 插值

### 文本

**示例一：插值之文本：{{}}**

|  |
| --- |
| <**li**>  <**h2**>1.1 文本({{}})</**h2**>  <**span**>ts={{ts}}</**span**> </**li**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **ts**:**new *Date***().getTime()}  }); |

### HTML

**示例二：插值之HTML：使用v-html指令用于输出html代码**

|  |
| --- |
| <**li**>  <**h2**>1.2 html(v-html)</**h2**>  <**div v-html="html"**></**div**> </**li**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **html**:**'<input type="text" value="0"/>'** }  }); |

### 属性

**示例三：插值之属性：HTML属性中的值应使用v-bind指令,使用class样式**

|  |
| --- |
| <**style**>  .**cl**{**font-size**: 20**px**;**color**:**red**;} </**style**>  <**li**>  <**h2**>1.3 属性(v-bind)</**h2**>  <**div v-bind:class="className"**>aaaaa</**div**> </**li**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **className**:**'cl'** }  }); |

### 表达式

**示例四：插值之表达式：Vue提供了完全的JavaScript表达式支持**

|  |
| --- |
| <**li**>  <**h2**>1.4 表达式</**h2**>  <**p**>{{str.substr(0,6).toUpperCase()}}</**p**>  <**p**>{{number+1}}</**p**>  <**p**>{{ok?'Yes':'No'}}</**p**>  <**p**>  *<!--   js动态生成id属性  v-bind:id="id"  v-bind:id="'username'+id"  -->* <**input v-bind:id="id" type="text" placeholder="请输入用户名"**/>  </**p**> </**li**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **str**:**'hello vue!!!'**,  **number**:12,  **ok**:**true**,  **id**:**'username'** }  }); |

## 指令

### v-if|v-else-if|v-else

v-if|v-else|v-else-if：根据其后表达式的bool值进行判断是否渲染该元素。他们只能是兄弟元素。

**注意：**

1. **v-else-if上一个兄弟元素必须是v-if；**
2. **v-else上一个兄弟元素必须是v-if或者是v-else-if；**

|  |
| --- |
| <**li**>  <**h2**>1.1 v-if|v-else-if|v-else</**h2**>  <**div v-if="sex==1"**>  男人  </**div**>  <**div v-else-if="sex==0"**>  女人  </**div**>  <**div v-else**>  不男不女  </**div**> </**li**>  **var *vm=*new** Vue({  **el**:**'#div'**,  **data**:{  **sex**:**'1'** }  }); |

### v-show

与v-if类似，只是会渲染其身后表达式为false的元素，而且会给这样的元素添加css代码：style="display:none"。

|  |
| --- |
| <**li**>  <**h2**>1.2 v-show</**h2**>  <**input type="checkbox" v-model="show"**/><**br**/>  <**div v-show="show"**>  show==true  </**div**> </**li**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **show**:**true** }  }); |

### v-for

类似JS的遍历。

遍历数组：v-for="item in items", items是数组，item为数组中的数组元素；

遍历对象：v-for="(value,key,index) in stu", value属性值,key属性名，index下标。

|  |
| --- |
| <**li**>  <**h2**>1.3 v-for</**h2**>  *<!-- 直接打印结果 -->* <**div v-for="e in dept"**>  id={{e.id}},name={{e.name}}<**br**/>  </**div**>  *<!-- 动态绑定select -->* <**p**>  <**select style="width**: 180**px**;**"**>  <**option value=""**>---请选择---</**option**>  <**option v-for="(v,i) in dept" v-bind:value="v.id"**>  {{v.name}}  </**option**>  </**select**>  </**p**> </**li**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **dept**:[{**id**:1,**name**:**'市场部'**},{**id**:2,**name**:**'研发部'**},{**id**:3,**name**:**'人事部'**}]}  }); |

### v-bind

|  |
| --- |
| <**li**>  <**h2**>1.4 v-bind</**h2**>  <**select style="width**: 180**px**;**"**>  <**option value=""**>---请选择---</**option**>  <**option v-for="(v,i) in dept" v-bind:value="v.id"**>  {{v.name}}  </**option**>  </**select**> </**li**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **dept**:[{**id**:1,**name**:**'市场部'**},{**id**:2,**name**:**'研发部'**},{**id**:3,**name**:**'人事部'**}]}  }); |

### v-on&v-model

v-model：用来在 input、select、textarea、checkbox、radio 等表单控件元素上创建双向数据绑定，根据表单上的值，自动更新绑定的元素的值。

|  |
| --- |
| <**li**>  <**h2**>1.5 v-on&v-model</**h2**>  <**div v-for="e in dept"**>  <**input type="checkbox" v-bind:value="e.id" v-model="did"** />{{e.name}}  </**div**>  <**button v-on:click="doClick"**>确 定</**button**> </**li**>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **dept**:[{**id**:1,**name**:**'市场部'**},{**id**:2,**name**:**'研发部'**},{**id**:3,**name**:**'人事部'**}],  **did**:[]},  **methods**:{  doClick:**function**(){  ***console***.log(**this**.did);  }  }  }); </**script**> |

### 参数之v-bind:href

一些指令能够接收一个“参数”，在指令名称之后以冒号表示，例如：

**示例一：**<a v-bind:href="url">...</a>

在这里href是参数，告知v-bind指令将该元素的href属性与表达式url的值绑定

**示例二：**<a v-on:click="doSomething">...</a>

在这里click参数是监听的事件名。

|  |
| --- |
| <**li**>  <**h2**>1.6 参数之v-bind:href</**h2**>  <**a v-bind:href="url"**>新浪网</**a**> </**li**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **url**:**'http://www.sina.com'** }  }); |

### 动态参数

从2.6.0开始，可以用方括号括起来的JavaScript表达式作为一个指令的参数

<a v-bind:[attrname]="url"> ... </a>

同样地，你可以使用动态参数为一个动态的事件名绑定处理函数

<a v-on:[evname]="doSomething"> ... </a>

**注1：动态参数表达式有一些语法约束，evName无效,evname有效，回避大写**

|  |
| --- |
| <**li**>  <**h2**>1.7 动态参数</**h2**>  *<!-- 注1：动态参数表达式有一些语法约束，evName无效,evname有效，回避大写 -->* <**button v-on:[evname]="doClick"**>点我</**button**> </**li**>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **evname**:**'click'** },  **methods**:{  doClick:**function**(){  ***console***.log(**this**.did);  }  }  }); </**script**> |

## 过滤器

**1）全局过滤器**

Vue.filter('filterName', function (value) {

// value 表示要过滤的内容

});

**2）局部过滤器**

new Vue({

filters:{'filterName':function(value){}}

});

|  |
| --- |
| <**body**>  *<!-- 1.示例一：局部过滤器之首字母大写 -->  <!-- 2.示例二：全局过滤器之日期格式化 -->* <**div id="div"**>  <**h1**>1.过滤器 </**h1**>  <**ul**>  <**li**>  <**h2**>1.1 局部过滤器</**h2**>  <**div**>{{name}}</**div**>  <**div**>{{name|cap}}</**div**>  </**li**>  <**li**>  <**h2**>1.2 全局过滤器</**h2**>  <**div**>{{ts}}</**div**>  <**div**>{{ts|formatDate}}</**div**>  </**li**>  </**ul**>  </**div**> </**body**> <**script**>  *//全局过滤器定义:* Vue.filter(**'formatDate'**,**function**(d){  **return** *fmtDate*(d,**'yyyyMMdd'**);  })  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **name**:**'tomcat'**,  **ts**:**new *Date***()  },  **filters**:{ *//局部过滤器,直接在Vue实例中定义* cap:**function**(v){  **return** v.substr(0,1).toUpperCase()+v.substr(1);  }  }  }); </**script**> |

## 计算属性

计算属性可用于快速计算视图（View）中显示的属性。这些计算将被缓存，并且只在需要时更新computed:{}

注意：let和var的区别？

**1）ES6新增的let，可以声明块级作用域的变量(局部变量)  
2）var定义全局变量,容易产生数据污染**

|  |
| --- |
| <**body**>  *<!-- 1.示例一：计算属性(订单购物车计算总金额) -->  <!-- 使用优先级：计算属性 -> 监听属性 -> methods -->* <**div id="div"**>  <**h1**>1.计算属性,ts={{ts}}</**h1**>  <**ul**>  <**li**>  <**div v-for="(e,i) in results"**>  {{e.course}},{{i}}<**input type="text" v-bind:value="e.score"** />  </**div**>  <**div**>{{sum}}</**div**>  </**li**>  </**ul**>  </**div**> </**body**> <**script**>    **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **ts**: **new *Date***().getTime(),  **results**:[  {**course**:**'语文'**,**score**:80},  {**course**:**'数学'**,**score**:100},  {**course**:**'英语'**,**score**:70}  ]  },  **computed**:{  sum:**function**(){  *//理解let和var的区别  //1.ES6新增的let，可以声明块级作用域的变量(局部变量)  //2.var定义全局变量,容易产生数据污染* **let** sum=0;  **for**(**let** i=0;i<**this**.results.length;i++){  sum+=*parseInt*(**this**.results[i].score);  }  **return** sum;  }  }  }); </**script**> |

## 监听属性

监听属性 watch，我们可以通过 watch 来响应数据的变化watch:{}

|  |
| --- |
| <**body**>  *<!-- 1.示例一：监听属性 -->* <**div id="div"**>  <**h1**>1.监听属性,ts={{ts}}</**h1**>  <**ul**>  <**li**>  km:<**input type="text" v-model="km"**/><**br** />  m:<**input type="text" v-model="m"**/><**br** />  </**li**>  </**ul**>  </**div**> </**body**> <**script**>    **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **ts**: **new *Date***().getTime(),  **km**:1,  **m**:1000  },  **watch**:{ *//监控属性* km:**function**(v){ *//监控中定义的方法与属性名相同* **this**.m=*parseInt*(v)\*1000;  },  m:**function**(v){  **this**.km=*parseInt*(v)/1000;  }  }  }); </**script**> |

# 第三天：Vue之基础语法01

1. 课前准备
2. 样式绑定
3. 事件处理器
4. 表单
5. 自定义指令
6. Vue组件

## 课前准备

1. readme.txt

## 样式绑定

### Css绑定

使用方式：v-bind:class="expression"。expression的类型：字符串、数组、对象。完整代码如下所示：

|  |
| --- |
| <**style**>  .**cl**{  **font-size**:30**px**;  }  .**cl1**{  **color**:**red**;  } </**style**>  <**li**>  <**h2**>1.1 css绑定</**h2**>  *<!-- 原始方式 -->* <**div class="cl"**>aaaaa</**div**>  *<!-- Vue之css绑定方式 -->* <**div v-bind:class="cn"**>bbbbb</**div**>  *<!-- Vue之css绑定多个类样式 -->* <**div v-bind:class="cns"**>ccccc</**div**> </**li**>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **cn**:**'cl1'**,  **cns**:[**'cl1'**,**'cl'**]  }  }); </**script**> |

**示例一：**使用HTML原始方式class属性绑定类样式；

**示例二：**使用Vue的v-bind:class=”单个样式属性”方式绑定；

**示例三：**使用Vue的v-bind:class=”样式属性数组”方式绑定；

### Style绑定

v-bind:style="expression"。expression的类型：字符串、数组、对象。完整代码如下所示：

**注意：CSS 属性名可以用驼峰式 (camelCase) 或 短横线分隔 (kebab-case，记得用引号括起来) 来命名。**

|  |
| --- |
| <**li**>  <**h2**>1.2 style绑定</**h2**>  *<!-- 原始方式 -->* <**div style="font-size**: 30**px**;**"**>style</**div**>  *<!-- Vue之Style绑定 -->  <!--   注：CSS 属性名可以用驼峰式 (camelCase) 或  短横线分隔 (kebab-case，记得用引号括起来) 来命名   -->* <**div v-bind:style="{color:color,fontSize:fontSize+'px'}"**>style</**div**>  *<!-- Vue之Style绑定多属性 -->* <**div v-bind:style="styleObject"**>style</**div**> </**li**>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **color**:**"red"**,  **fontSize**:30,  **styleObject**:{  **fontSize**:**"12px"**,  **color**:**"blue"**,  **fontWeight**:**"bold"** }  }  }); </**script**> |

**示例一：**使用HTML原始方式style属性绑定样式；

**示例二：**使用Vue的v-bind:style=”{}”方式绑定；

|  |
| --- |
| <**div v-bind:style="{color:color,fontSize:fontSize+'px'}"**>style</**div**> |

**示例三：**使用Vue的v-bind:style=”样式Json对象”方式绑定；

|  |
| --- |
| <**div v-bind:style="styleObject"**>style</**div**> |

## 事件处理器

事件监听可以使用v-on指令。

### 事件修饰符

Vue通过由点(.)表示的指令后缀来调用修饰符。如：.stop/.prevent/.capture/.self/.once等。完整代码实现如下所示：

|  |
| --- |
| <**h2**>1.1 事件修饰符</**h2**>  *<!-- 示例一：事件修饰符之发送消息 -->* <**div**>{{msg}}</**div**>  <**div**>  <**input type="text" v-model="content"**/><**button v-on:click="doSend"**>发送</**button**>  </**div**>  *<!-- 示例二：事件修饰符之只能点击一次once -->* <**div**>  <**button v-on:click.once="doSend"**>只能点击一次哦</**button**>  </**div**>  *<!-- 示例三：事件修饰符之阻止表单提交prevent -->* <**div**>  <**form action="bookAction.action" v-on:submit.prevent="doSubmit"**>  <**label**>书本名称：</**label**><**input type="text"** />  <**input type="submit" value="提交"**/>  </**form**>  </**div**> </**li**>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **ts**:**new *Date***().getTime(),  **msg**:**null**,  **content**:**null** },  **methods**:{  doSend:**function**(){  ***console***.log(**'doSend'**);  **this**.**msg**=**this**.**content**;  },  doSubmit:**function**(){  ***console***.log(**'doSubmit'**);  }  }  }); </**script**> |

**示例一：**事件修饰符之模拟发送消息；

**示例二：**事件修饰符之只能点击一次，使用事件修饰符.once；

**示例三：**事件修饰符之阻止表单提交，使用时间修饰符.prevent；

### 按键修饰符

Vue允许为v-on在监听键盘事件时添加按键修饰符。

全部的按键别名：.enter/.tab/.delete/.esc/.space/.up/.down/.left/.right/.ctrl/.alt/.shift/.meta。

|  |
| --- |
| <**li**>  <**h2**>1.2 按键修饰符</**h2**>  <**div**>{{msg}}</**div**>  <**div**>  <**input type="text" v-model="content" v-on:keyup.enter="doSend"**/><**button v-on:click="doSend"**>发送</**button**>  </**div**> </**li**>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **ts**:**new *Date***().getTime(),  **msg**:**null**,  **content**:**null** },  **methods**:{  doSend:**function**(){  ***console***.log(**'doSend'**);  **this**.**msg**=**this**.**content**;  }  }  }); </**script**> |

## 表单

用v-model指令在表单控件元素上创建双向数据绑定。常用控件：文本框、密码框、文本域、隐藏域、单选框、复选框、下拉框等。以下为完整代码实现：

|  |
| --- |
| <**li**>  <**h2**>1.1 常用控件&修饰符</**h2**>  <**div**>  <**label**>用户账号：</**label**><**input type="text" v-model="username"** />  </**div**>  <**div**>  <**label**>用户密码：</**label**><**input type="password" v-model="password"**/>  </**div**>  <**div**>  <**label**>年龄：</**label**><**input type="text" v-model.number="age"**/>  </**div**>  <**div**>  <**label**>性别：</**label**>  <**input type="radio" v-model="sex" value="1"** />男  <**input type="radio" v-model="sex" value="0"** />女  </**div**>  <**div**>  <**label**>所在部门：</**label**>  <**select v-model="deptid"**>  <**option value=""**>---请选择---</**option**>  <**option v-for="(e,i) in dept" v-bind:value="e.id"**>  {{e.name}}  </**option**>  </**select**>  </**div**>  <**div**>  <**label**>个人爱好：</**label**>  <**div style="display**:**inline" v-for="e in hobbies"**>  <**input type="checkbox" v-model="hobby" v-bind:value="e"** />{{e}}  </**div**>  </**div**>  <**div**>  <**label**>备注信息：</**label**>  <**textarea v-model="remark"**></**textarea**>  </**div**>  <**div**>  *<!--   协议同意效果一  <input type="checkbox" v-on:click="flag=!flag" />同意以上内容  <button v-on:click="doSubmit" v-bind:disabled="!flag">确 定</button>  -->  <!--   协议同意效果二之使用属性计算方式(computed)  <input type="checkbox" v-model="flag" />同意以上内容  <button v-on:click="doSubmit" v-bind:disabled="allowed">确 定</button>  -->* <**input type="checkbox" v-model="flag"** />同意以上内容  </**div**>  <**div**>  <**button v-on:click="doSubmit" v-bind:disabled="allowed"**>确 定</**button**>  </**div**> </**li**>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **ts**:**new *Date***().getTime(),  **username**:**null**,  **password**:**null**,  **age**:**null**,  **sex**:**'1'**,  **deptid**:**""**,  **hobby**:[],  **remark**:**null**,  **flag**:**false**,  **dept**:[  {**id**:**'1'**,**name**:**'研发部'**},  {**id**:**'2'**,**name**:**'市场部'**},  {**id**:**'3'**,**name**:**'人事部'**}  ],  **hobbies**:[**'写代码'**,**'打游戏'**,**'睡觉'**]  },  **methods**:{  doSubmit:**function**(){  ***console***.log(**"doSubmit"**);  **let** obj={  **username**:**this**.**username**,  **password**:**this**.**password**,  **age**:**this**.**age**,  **sex**:**this**.**sex**,  **deptid**:**this**.**deptid**,  **hobby**:**this**.**hobby**,  **remark**:**this**.**remark** }  **let** age=**this**.age+10;  ***console***.log(**"age="**+age);  ***console***.log(obj);  ***console***.log(**this**.$data);  }  },  **computed**:{  allowed:**function**(){  **return** !**this**.flag;  }  }  }); </**script**> |

**示例一：**使用v-model实现常用控件的数据双向绑定功能；

**示例二：**实现同意协议效果（2种方式）；

1. 定义flag:true属性，用于控制选中同意协议后按钮的启用禁用效果。**红色部分**为关键代码：

|  |
| --- |
| <**input type="checkbox" v-on:click="flag=!flag"** />同意以上内容 <**button v-on:click="doSubmit" v-bind:disabled="!flag"**>确 定</**button**> |

1. 定义flag:false属性，通过计算属性(computed)方式控制同意协议效果。**红色部分**为关键代码：

|  |
| --- |
| <**input type="checkbox" v-model="flag"** />同意以上内容 <**button v-on:click="doSubmit" v-bind:disabled="allowed"**>确 定</**button**>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **ts**:**new *Date***().getTime()  **flag**:**false**  },  **computed**:{  allowed:**function**(){  **return** !**this**.flag;  }  }  }); </**script**> |

**示例三：**修饰符之数字修饰符(.number)，通过年龄字段演示效果；**红色部分**为关键代码：

|  |
| --- |
| <**div**><**label**>年龄：</**label**><**input type="text" v-model.number="age"**/></**div**> |

## 自定义指令

除了Vue自带的指令(v-on|v-model)等, Vue也允许注册自定义指令，根据作用范围又分为：全局指令/局部指令。

1. **指令定义函数提供的钩子函数**

|  |  |
| --- | --- |
| **bind** | 只调用一次，指令第一次绑定到元素时调用，用这个钩子函数可以定义一个在绑定时执行一次的初始化动作。 |
| **inserted** | 被绑定元素插入父节点时调用（父节点存在即可调用，不必存在于 document 中） |
| **update** | 被绑定元素所在的模板更新时调用，而不论绑定值是否变化。通过比较更新前后的绑定值，可以忽略不必要的模板更新（详细的钩子函数参数见下） |
| **componentUpdated** | 被绑定元素所在模板完成一次更新周期时调用 |
| **unbind** | 只调用一次，指令与元素解绑时调用 |

1. **钩子函数的参数（仅供参考）**

|  |  |  |
| --- | --- | --- |
| **el** | | 指令所绑定的元素，可以用来直接操作 DOM |
| **binding** | **name** | 指令名，不包括 v- 前缀 |
| **value** | 指令的绑定值， 例如： v-my-directive="1 + 1", value 的值是 2 |
| **oldValue** | 指令绑定的前一个值，仅在 update 和 componentUpdated 钩子中可用。无论值是否改变都可用 |
| **expression** | 绑定值的表达式或变量名。 例如 v-my-directive="1 + 1" ， expression 的值是 "1 + 1" |
| **arg** | 传给指令的参数。例如 v-my-directive:foo， arg 的值是 "foo" |
| **modifiers** | 一个包含修饰符的对象。 例如： v-my-directive.foo.bar, 修饰符对象 modifiers 的值是 { foo: true, bar: true } |
| **vnode** | | Vue 编译生成的虚拟节点 |
| **oldVnode** | | 上一个虚拟节点，仅在 update 和 componentUpdated 钩子中可用 |

### 局部指令

|  |
| --- |
| <**input type="text" v-fcs**/>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **ts**:**new *Date***().getTime()  },  **directives**:{ *//局部指令* **fcs**:{ *//自定义指令名称* inserted:**function**(){ *//自定义指令之钩子函数* ***console***.log(**'inserted'**);  }  }  }   }); </**script**> |

### 全局指令

|  |
| --- |
| <**input type="text" v-fcs**/>  <**script**>  Vue.directive(**'fcs'**,{ *//全局指令* inserted:**function**(){ *//自定义指令之钩子函数* ***console***.log(**'inserted'**);  }  });  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **ts**:**new *Date***().getTime()  }  }); </**script**> |

## 组件

组件(Component)是Vue最强大的功能之一，组件可以扩展HTML元素，封装可重用的代码，组件系统让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型的应用的界面都可以抽象为一个组件树。

**定义组件名的方式有两种：**

1. **短横线分隔命名（推荐使用）：**

Vue.component('my-component-name', { /\* ... \*/ })，引用方式：<my-component-name>

1. **首字母大写命名：**

Vue.component('MyComponentName', { /\* ... \*/ })，引用方式： <my-component-name>和<MyComponentName>都是可接受的

### 局部组件

局部组件: new Vue({el:'#d1',components:{...}})

注册后，我们可以使用以下方式来调用组件：<tagName></tagName>。完整代码如下所示：

|  |
| --- |
| *<!-- 局部组件 --> <!-- 示例一：在Vue实例中使用components:{'组件名称':{..}}定义局部组件 示例二：定义template属性，指定输出内容 示例三：定义data属性，一定要使用函数方式（多例模式） 示例四：通过props属性将外部数据传入内部组件 示例五：自定义事件之onClick -->* <**kfc-button name="张三"**></**kfc-button**>  <**script**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **ts**:**new *Date***().getTime()  },  **methods**:{  abcd:**function**(count){  ***console***.log(**'abcd,count='**+count);  }  },  *//局部组件* **components**:{   **'kfc-button'**:{ *//自定义组件名称(自定义HTML标签)* **props**:[**'name'**],  *//自定义组件输出内容* **template**:**'<button v-on:click="doClick">【局部组件】有种点我，{{name}}点击了{{count}}次</button>'**,  *//必须使用函数!!!  //1) data:{} 是单例模式;   //2) data:function(){return{};} 是多例模式,在每次调用时都会new一个新的对象* data:**function**(){   **return** {  **count**:0  };  },  *//自定义组件的事件定义* **methods**:{  doClick:**function**(){  **this**.**count**++;  ***console***.log(**this**.**count**);  }  }  }  }  }); </**script**> |

**示例一：**在Vue实例中使用components:{'组件名称':{..}}定义局部组件。**红色部分**为关键代码：

|  |
| --- |
| <**script**>  **var *vm***=**new** Vue({  **el**:**'#div'**,  *//局部组件* **components**:{   **'kfc-button'**:{  }  }  }); </**script**> |

**示例二：**定义template属性，指定输出内容。**红色部分**为关键代码：

|  |
| --- |
| **components**:{   **'kfc-button'**:{ *//自定义组件名称(自定义HTML标签)*  *//自定义组件输出内容* **template**:**'<button v-on:click="doClick">【局部组件】有种点我，{{name}}点击了{{count}}次</button>'**,  } } |

**示例三：**定义data属性，一定要使用函数方式（**多例模式**）。**红色部分**为关键代码：

|  |
| --- |
| *//局部组件* **components**:{   **'kfc-button'**:{ *//自定义组件名称(自定义HTML标签)*  *//必须使用函数!!!  //1) data:{} 是单例模式;   //2) data:function(){return{};} 是多例模式,在每次调用时都会new一个新的对象* data:**function**(){   **return** {  **count**:0  };  }  } } |

**示例四：**通过props属性将外部数据传入内部组件。**红色部分**为关键代码：

|  |
| --- |
| <**kfc-button name="张三"**></**kfc-button**>  *//局部组件* **components**:{   **'kfc-button'**:{ *//自定义组件名称(自定义HTML标签)* **props**:[**'name'**]，  *//自定义组件输出内容* **template**:**'<button v-on:click="doClick">【局部组件】有种点我，{{name}}点击了{{count}}次</button>'**,  } } |

**示例五：**自定义事件之onClick。**红色部分**为关键代码：

|  |
| --- |
| *//局部组件* **components**:{   **'kfc-button'**:{ *//自定义组件名称(自定义HTML标签)*  *//自定义组件输出内容* **template**:**'<button v-on:click="doClick">【局部组件】有种点我，{{name}}点击了{{count}}次</button>'**,  *//自定义组件的事件定义* **methods**:{  doClick:**function**(){  **this**.**count**++;  ***console***.log(**this**.**count**);  }  }  } } |

### 全局组件

全局组件：Vue.component(tagName, options)，tagName为组件名，options为配置选项。

注册后，我们可以使用以下方式来调用组件： <tagName></tagName>

**案例：自定义组件如何调用Vue实例中的事件方法：**

|  |
| --- |
| *<!-- 全局组件 --> <!-- 示例六：自定义事件之组件调用Vue实例中的事件方法 --> <!--  流程：点击自定义组件按钮 -> 触发组件内部点击事件  -> 通过$emit(事件名称,参数)调用Vue实例中定义的方法 -->* <**my-button v-on:tree-click="abcd"**></**my-button**>  <**script**>  *//全局组件* Vue.component(**'my-button'**,{  **props**:[**'name'**],  *//自定义组件输出内容* **template**:**'<button v-on:click="threeClick">【全局组件】有种点我，{{name}}点击了{{count}}次</button>'**,  *//必须使用函数!!!  //1) data:{} 是单例模式;   //2) data:function(){return{};} 是多例模式,在每次调用时都会new一个新的对象* data:**function**(){   **return** {  **count**:0  };  },  *//自定义组件的事件定义* **methods**:{  threeClick:**function**(){  **this**.**count**++;  **if**(**this**.**count**%3==0){  ***console***.log(**"count="**+**this**.**count**);  **this**.$emit(**'tree-click'**,**this**.**count**);  }  }  }  });  **var *vm***=**new** Vue({  **el**:**'#div'**,  **data**:{  **ts**:**new *Date***().getTime()  },  **methods**:{  abcd:**function**(count){  ***console***.log(**'abcd,count='**+count);  }  },  });  </**script**> |

**自定义组件调用Vue实例中的事件方法流程分析：**

1. 在自定义组件中定义自定义事件，通过v-on:自定义事件名称。一般自定义事件名称定义使用“短横线分隔命名”，例如：tree-click
2. 在自定义组件中通过v-on:自定义名称=”调用Vue实例中的事件”，例如：v-on:three-click=”abcd”
3. 在自定义组件methods属性中定义click方法，并将方法加入到template的输出元素的点击事件中，例如：template:’<button v-on:click=”doClick”></button>’
4. 在自定义组件methods的click方法中通过$emit(自定义组件事件名称,参数)，调用Vue实例中的事件方法。注：$emit中的自定义组件事件名称要与组件中的事件名称一致！！！

# 第四天：Vue之路由器

1. 课前准备
2. SPA
3. Vue之路由

## 课前准备

1. readme.txt
2. 资料
3. Vue项目中的文件及文件夹命名规范

## SPA

### 什么是SPA

单页Web应用（single page application，SPA），就是只有一个Web页面的应用，是加载单个HTML页面，并在用户与应用程序交互时动态更新该页面的Web应用程序。

* **单页面应用程序：**

只有第一次会加载页面, 以后的每次请求, 仅仅是获取必要的数据.然后, 由页面中js解析获取的数据, 展示在页面中

* **传统多页面应用程序：**

对于传统的多页面应用程序来说, 每次请求服务器返回的都是一个完整的页面

**优势：**减少了请求体积，加快页面响应速度，降低了对服务器的压力，更好的用户体验，让用户在web app感受native app的流畅。

### SPA实现思路和技术点

1. Ajax
2. 锚点的使用（window.location.hash #）
3. hashchange 事件 window.addEventListener("hashchange",function () {})
4. 监听锚点值变化的事件，根据不同的锚点值，请求相应的数据
5. 原本用作页面内部进行跳转，定位并展示相应的内容

## Vue之路由

### 什么是路由

Vue.js 路由允许我们通过不同的 URL 访问不同的内容。通过 Vue.js 可以实现多视图的单页Web应用（single page web application，SPA）

* **传统页面访问和路由的区别？**

1. 传统方式页面访问是通过请求路径跳转到访问页面。
2. Vue路由是根据路由路径访问不同的组件内容。

* **路由和路由器的区别？**

1. 路由即路线，类似于12306网站中的火车车次线路；
2. 路由器包含多个路由，类似12306网站中汇集的所有火车车次线路信息；

### Vue配置路由

#### 示例一：路由基本实例

##### 引入依赖库

|  |
| --- |
| <**script src="https://cdn.bootcss.com/vue/2.6.10/vue.js"**></**script**> <**script src="https://cdn.bootcss.com/vue-router/3.0.6/vue-router.js"**></**script**> |

##### 创建自定义组件

|  |
| --- |
| *//创建自定义组件,通过Vue.extend方式,并用const常用接收实例对象 //可通过路由的path或name属性,定位到具体的组件实例* **const *Home***=Vue.extend({  *//注：必须包裹一个根元素* **template**:**'<div><h1>Home组件</h1><div>Home组件的内容区域</div></div>'** }); **const *About***=Vue.extend({  **template**:**'<div><h1>About组件</h1><div>About组件的内容区域</div></div>'** }); |

##### 自定路由(即路线)

|  |
| --- |
| *//定义路由即路线* **var *routes***=[  *//{path:"/",component:Home}, //默认显示效果* {**path**:**"/Home"**,**component**:***Home***},  {**path**:**"/About"**,**component**:***About***} ]; |

##### 创建路由器实例

|  |
| --- |
| *//定义路由器* **const *router***=**new** VueRouter({**routes**:***routes***}); |

##### 创建和挂载根实例

|  |
| --- |
| *//创建和挂载根实例,通过$mount方式挂载到指定的管理区域(重要)* **var *vm***=**new** Vue({  *//el:'#app',* **router**:***router***,  data:**function**(){  **return** {  **ts**:**new *Date***().getTime()  }  } }).$mount(**"#app"**); |

##### 使用RouterLink和RouterView组件导航和显示

|  |
| --- |
| <**div id="app"**>  <**h1**>1.路由器,ts={{ts}}</**h1**>  <**div**>  <**router-link to="/Home"**>go to home</**router-link**>  <**router-link to="/About"**>go to about</**router-link**>  </**div**>  <**div**>  <**router-view**></**router-view**>  </**div**> </**div**> |

#### 示例二：vue中导航中的后退-前进-编程式导航

**Step1：**后退：this.$router.go(-1)

**Step2：**前进：this.$router.go(1)

**Step3：**切换组件：this.$router.push({ 切换到name为home的路由

name:'home'

});

|  |
| --- |
| <**div id="app"**>  <**h1**>1.路由器,ts={{ts}}</**h1**>  <**div**>  <**router-link to="/Home" replace**>go to home</**router-link**>  <**router-link to="/About"** >go to about</**router-link**>  </**div**>  <**div**>  <**button v-on:click="previous"**>向前</**button**>  <**button v-on:click="next"**>向后</**button**>  <**button v-on:click="gotopage"**>切换指定组件</**button**>  </**div**>  <**div**>  <**router-view**></**router-view**>  </**div**> </**div**>  **var *vm***=**new** Vue({  *//el:'#app',* **router**:***router***,  data:**function**(){  **return** {  **ts**:**new *Date***().getTime()  }  },  **methods**:{  previous:**function**(){  **this**.$router.go(1);  ***console***.log(**"previous"**);  },  next:**function**(){  **this**.$router.go(-1);  ***console***.log(**"next"**);  },  gotopage:**function**(){  **this**.$router.push({  **path**:**"/About"** });  ***console***.log(**"gotopage"**);  }  } }).$mount(**"#app"**); |

#### 示例三：Router-link之replace属性

|  |
| --- |
| <**div id="app"**>  <**h1**>1.路由器,ts={{ts}}</**h1**>  <**div**>  <**router-link to="/Home" replace**>go to home</**router-link**>  <**router-link to="/About"** >go to about</**router-link**>  </**div**>  <**div**>  <**button v-on:click="previous"**>向前</**button**>  <**button v-on:click="next"**>向后</**button**>  <**button v-on:click="gotopage"**>切换指定组件</**button**>  </**div**>  <**div**>  <**router-view**></**router-view**>  </**div**> </**div**> |

#### 示例四：路由之显示默认组件

|  |
| --- |
| *//定义路由即路线* **var *routes***=[  {**path:"/"**,**component:Home**}, *//默认显示效果* {**path**:**"/Home"**,**component**:***Home***},  {**path**:**"/About"**,**component**:***About***} ]; |

#### Router-link相关属性介绍

##### to

to表示目标路由的链接

<router-link to="/home">Home</router-link><!-- 字符串 -->

<router-link v-bind:to="'home'">Home</router-link><!-- 使用 v-bind 的 JS 表达式 -->

##### replace

设置 replace 属性的话，当点击时，会调用 router.replace() 而不是 router.push()，导航后不会留下 history 记录。<router-link :to="{ path: '/home'}" replace></router-link>

vue中导航中的后退-前进-编程式导航：

1. this.$router.go(-1) 代表着后退
2. this.$router.go(1) 代表着前进
3. this.$router.push({ 切换到name为home的路由

name:'home'

});

##### append

设置 append 属性后，则在当前 (相对) 路径前添加基路径。例如，我们从 /a 导航到一个相对路径 b，如果没有配置 append，则路径为 /b，如果配了，则为 /a/b。**append属性类似于html中的base标签，可用于将相对路径转换成绝对路径。**

<router-link :to="{ path: 'relative/path'}" append></router-link>

##### 渲染标签

**<router-link>默认会被渲染成一个a标签**。如果有时候想要 <router-link> 渲染成某种标签，例如 <li>。可通过<router-link>中的tag属性自定义渲染标签类型，例如：

<router-link to="/foo" tag="li">foo</router-link>

<!-- 渲染结果 -->

<li>foo</li>

# 第五天：Node.Js环境搭建

1. 课前准备
2. html+Vue+ElementUI开发
3. NodeJs环境搭建

## 课前准备

1. nmgwap-vueproject-master.zip
2. node-v10.15.3-win-x64.zip
3. readme.txt

## html+Vue+ElementUI开发

## NodeJs环境搭建

### 什么是Node.js

Node.js是一个基于Chrome V8引擎的[JavaScript运行环境]。Node.js使用了一个事件驱动、非阻塞式I/O 的模型。Node.js是一个让JavaScript运行在服务端的开发平台，它让JavaScript成为与PHP、Python、Perl、Ruby等服务端语言平起平坐的脚本语言。

1）Node.js依赖JavaScript运行环境，开发语言是：Javascript

2）J2EE依赖Java运行环境，开发语言是：Java

文档地址：<http://nodejs.cn/api/>

### 什么是npm

npm其实是Node.js的包管理工具（package manager）。**注：npm有点类似于maven**

理解：为啥我们需要一个包管理工具呢？因为我们在Node.js上开发时，会用到很多别人写的JavaScript代码。如果我们要使用别人写的某个包，每次都根据名称搜索一下官方网站，下载代码，解压，再使用，非常繁琐。于是一个集中管理的工具应运而生：大家都把自己开发的模块打包后放到npm官网上，如果要使用，直接通过npm安装就可以直接用，不用管代码存在哪，应该从哪下载。更重要的是，如果我们要使用模块A，而模块A又依赖于模块B，模块B又依赖于模块X和模块Y，npm可以根据依赖关系，把所有依赖的包都下载下来并管理起来。否则，靠我们自己手动管理，肯定既麻烦又容易出错。

### Node.js环境搭建

#### 下载

下载地址：<https://nodejs.org/zh-cn/download/>

选择相应的版本下载，本章使用的是：node-v10.15.3-win-x64.zip

注1：Node有两个版本线: LTS是长期维护的稳定版本Current是新特性版本



#### 解压

将下载的node-v10.15.3-win-x64.zip解压到指定目录中，并在解压后的目录下建立**node\_global**和**node\_cache**这两个目录。

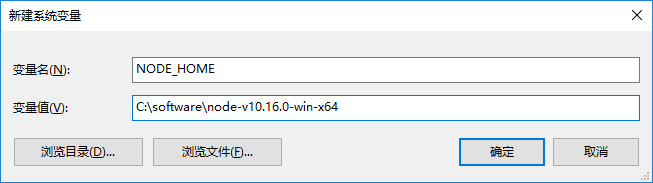
1. **node\_global**：npm全局安装位置
2. **node\_cache**：npm缓存路径



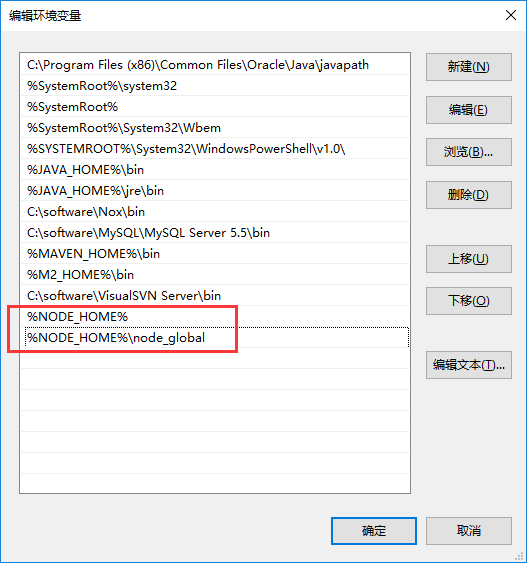
#### 配置环境变量

新增NODE\_HOME，值为：D:\tools\node-v10.15.3-win-x64（视自己安装目录而定），修改PATH并在最后添加：;%NODE\_HOME%;%NODE\_HOME%\node\_global;

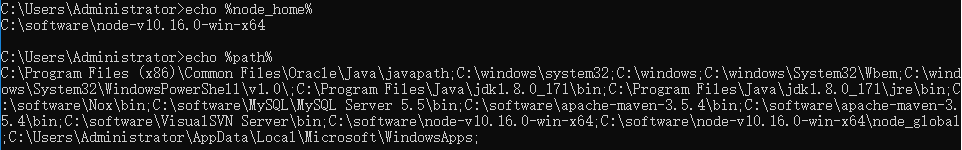
* **NODE\_HOME：**



* **Path配置：**



* **环境变量查看echo %node\_home%和echo %path%**



* **测试安装是否成功，node -v (NodeJs版本)和npm -v (npm版本)**



#### 配置npm之node\_cache和node\_global

打开CMD，分开执行如下命令：

|  |
| --- |
| npm config set cache "D:\tools\node-v10.15.3-win-x64\node\_cache"  npm config set prefix "D:\tools\node-v10.15.3-win-x64\node\_global" |



**注意：**

1. 将步骤一创建的node\_global(npm全局安装位置)和node\_cache(npm缓存路径)与npm联系起来
2. 如果执行命令卡死，可以删除C:\Users\用户名\.npmrc 后重新执行。（用户名：为当前电脑的用户名）
3. "D:\tools\node-v10.15.3-win-x64\node\_global"，双引号不能少

#### 修改npm镜像

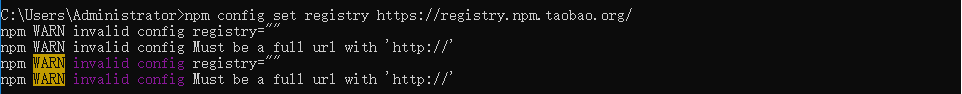
修改npm镜像提高下载速度(可以使用 cnpm或直接设置--registry，推荐设置--registry)

1. **--Registry**

|  |
| --- |
| **## 设置淘宝源**  **npm config set registry <https://registry.npm.taobao.org/>**  **## 查看源，可以看到设置过的所有的源**  **npm config get registry** |

注：其实此步骤的内容就是将以下代码添加到

C:\Users\用户名\.npmrc文件中registry=https://registry.npm.taobao.org



1. **Cnpm**

|  |
| --- |
| **npm install -g cnpm --registry=https://registry.npm.taobao.org** |

**注意：**

1. cnpm安装完成后，以后就可以用cnpm命令代替npm命令, 此时npm还是会用官方镜像，cnpm会用国内镜像；
2. 如果要恢复成原来的设置，执行如下：npm config set registry <https://registry.npmjs.org/>

#### 验证安装结果

1. **版本验证**



1. **查看淘宝镜像设置情况**



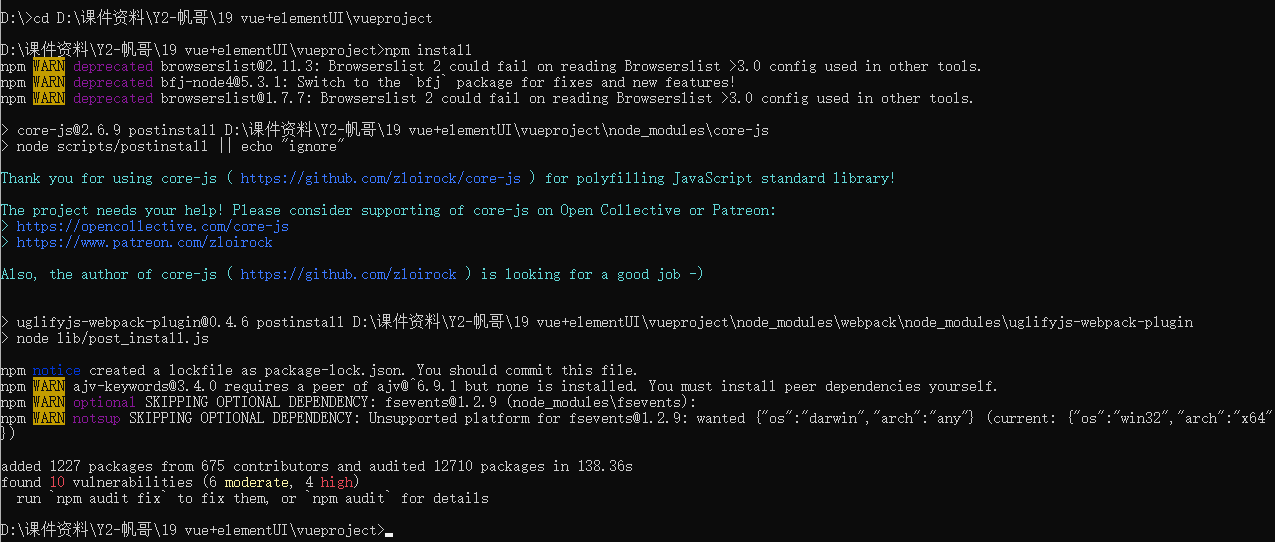
1. **查看npm全局路径设置情况**



以上命令执行完毕后，会生成如下文件：%node\_home%\node\_global\node\_modules\webpack

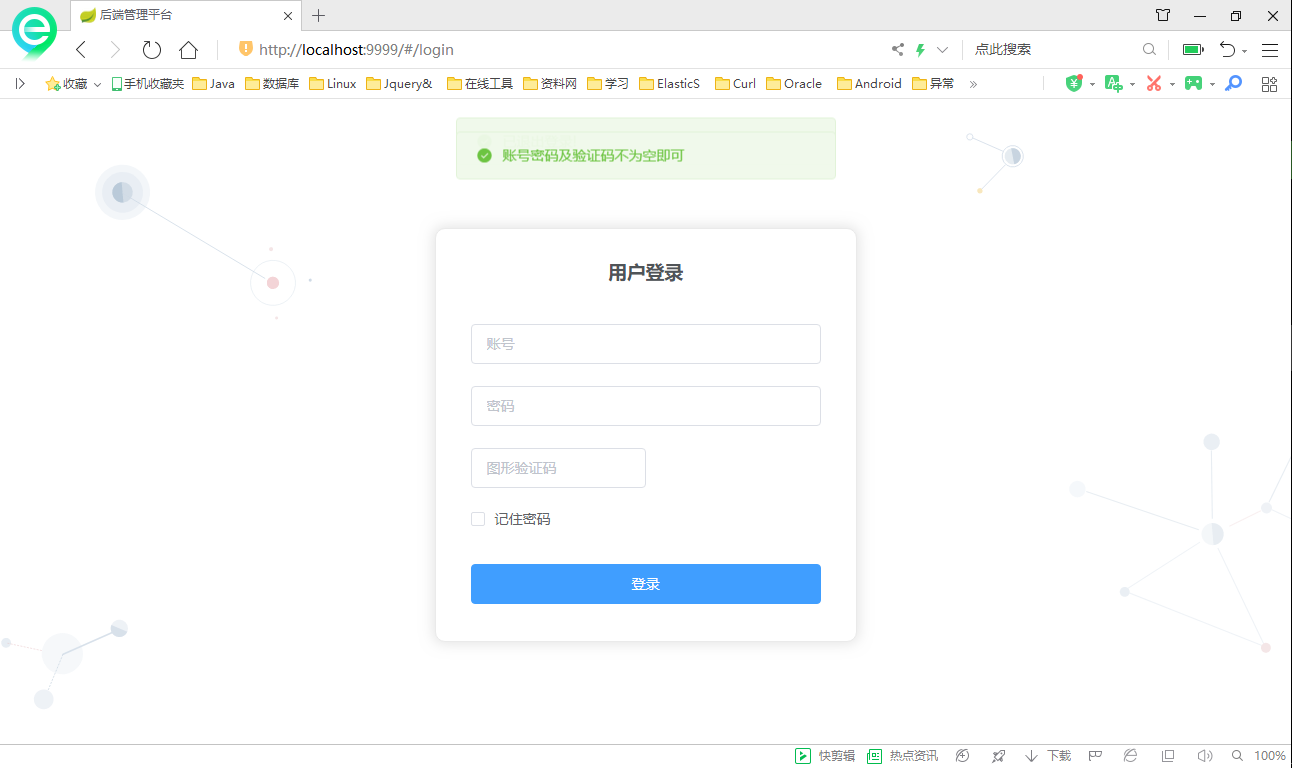
#### 运行下载的Node.js项目

将下载的项目解压到指定目录，本例是解压到：D:\temp\vueproject。





最终显示结果：



# 第六天：使用vue-cli搭建SPA项目

1. 课前准备
2. 什么是vue-cli
3. 安装vue-cli
4. 使用vue-cli构建项目
5. Package.json详解
6. Vue项目结构说明
7. 什么是\*.vue文件
8. 综合案例

## 课前准备

1）readme.txt

## 什么是vue-cli

vue-cli是vue.js的脚手架，用于自动生成vue.js+webpack的项目模板。创建命令如下：

|  |
| --- |
| vue init webpack XXXX |

**注：**

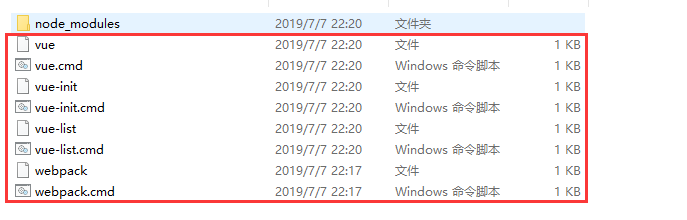
1. **XXXX为自己创建项目的名称；**
2. **必须先安装vue，vue-cli，webpack，node等一些必要的环境**

## 安装vue-cli

执行命令：

|  |
| --- |
| npm install -g vue-cli |

安装成功后，D:\XXXX\node-v10.16.0-win-x64\node\_global目录下将会出现如下内容：



安装完成之后打开命令窗口并输入 vue -V（注意这里是大写的“V”），如果出现相应的版本号，则说明安装成功。



## 使用vue-cli构建项目

### 使用脚手架创建项目骨架

**此步骤可理解成：使用eclipse创建一个maven的web项目。**

**Step1：**打开CMD命令窗口

**Step2：**切换到D盘

**Step3：**进入d:\vueProject目录

**Step4：**使用命令【vue init webpack spa1】，创建SPA项目，它会在当前目录生成一个以“spa1”命名的文件夹。spa1即为项目名，项目名不能用中文或大写字母。

**然后终端会出现“一问一答”模式：**

**Step1：**Project name：项目名，默认是输入时的那个名称spa1，直接回车

**Step2：**Project description：项目描述，直接回车

**Step3：**Author：作者，随便填或直接回车

**Step4：**Vue build：选择题，一般选第一个

1. **Runtime + Compiler: recommended for most users** //运行加编译，官方推荐，就选它了
2. 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 //仅运行时，已经有推荐了就选择第一个了

**Step5：**Install vue-router：是否需要vue-router，Y选择使用，这样生成好的项目就会有相关的路由配置文件

**Step6：**Use ESLint to lint your code：是否用ESLint来限制你的代码错误和风格。N 新手就不用了，但实际项目中一般都会使用，这样多人开发也能达到一致的语法

**Step7：**Set up unit tests：是否安装单元测试 N

**Step8：**Setup e2e tests with Nightwatch?：是否安装e2e测试 N

**Step9：**Should we run `npm install` for you after the project has been created? (recommended) (Use arrow keys)

**> Yes, use NPM**

Yes, use Yarn

No, I will handle that myself //选择题：选第一项“Yes, use NPM”是否使用npm install安装依赖

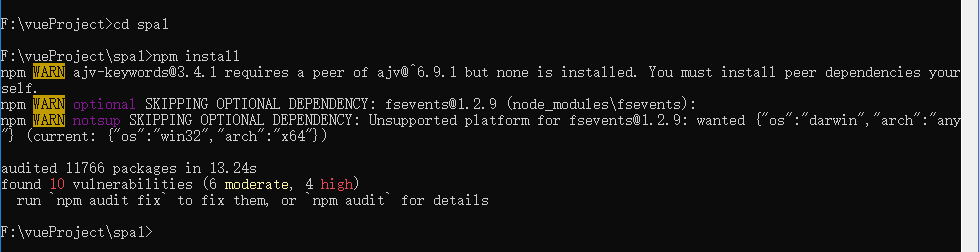


### 使用NPM安装所需模块

运行完上面的命令后，我们需要将当前路径改变到SPA这个文件夹内，然后安装需要的模块。**此步骤可理解成：maven的web项目创建成功后，修改pom文件添加依赖。**

**Step1：cd spa1 #改变路径到spa1文件夹下**

**Step2：npm install #安装所有项目需要的npm模块**



**补充说明：**

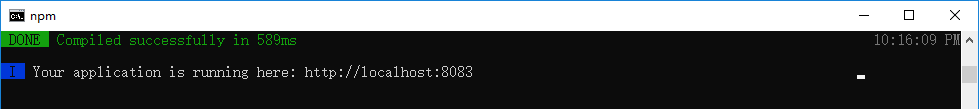
1. **npm install：**下载“package.json”中dependencies和devdependencies中配置的所有依赖模块，并保存到项目的node\_modules目录；
2. **npm install xxx -g：**全局安装，下载依赖模块，并保存到%node\_home%\node\_global\node\_modules目录下；
3. **npm install xxx -S：**写入到package.json的dependencies对象，并保存到项目的node\_modules目录；
4. **npm install xxx -D：**写入到package.json的devDependencies对象，并保存到项目的node\_modules目录；

### 启动并访问项目

启动并访问项目。**此步骤可理解成：启动tomcat，并通过浏览器访问项目**

**Step1：cd spa1 #改变路径到spa1文件夹下**

**Step2：npm run dev #启动项目**



**修改启动项目运行端口，进入项目的config/index.js目录：**

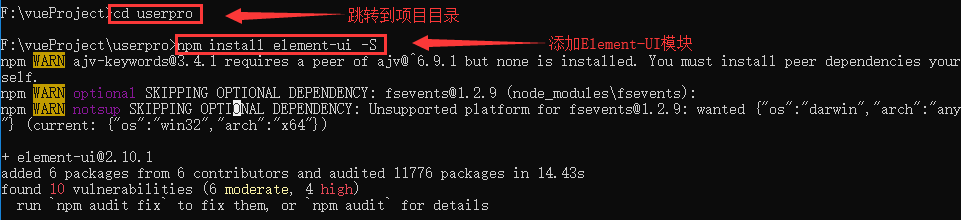


### 停止项目添加element-ui模块

停止项目添加element-ui模块。

**Step1：cd spa1 #改变路径到spa1文件夹下**

**Step2：npm install element-ui -S**



## package.json详解

每个项目的根目录下面，一般都有一个package.json文件，定义了这个项目所需要的各种模块，以及项目的配置信息（比如名称、版本、许可证等元数据）。npm install命令根据这个配置文件，自动下载所需的模块，也就是配置项目所需的运行和开发环境。

详情见资料“package-详解.json”中的相关注释



注：

1. dependencies：在生产环境中需要用到的依赖
2. devDependencies：在开发、测试环境中用到的依赖，在上线打包后的代码是不存在的

## vue项目结构说明



## 什么是\*.vue文件

\*.vue 文件，是一个自定义的文件类型，用类似HTML的语法描述一个Vue组件。每个.vue文件包含三种类型的顶级语言块 <template>, <script> 和 <style>。这三个部分分别代表了 html,js,css。

**注：**

1. 不能直接把html代码包裹在<template></template>中，而是必须在里面方置一个html标签来包裹所有的代码。一般情况是使用<div></div>标签包含其它的代码(保证只有一个根元素)；
2. js代码写在<script>标签中，并通过export导出；
3. 样式与以前的写法一样

## 综合案例

**案例一：基本实例**

1. 将vue-cli构建的项目导入到HBuilder中；
2. 在src目录下新建views目录，用于存放页面模块；
3. 在components目录下复制一份HelloWorld.vue，并重命名为AppMain.vue，删除掉<template>中<div>下的所有标签信息，并在<script>标签中的data属性中定义系统时间戳；
4. 修改router目录下的index.js模块主文件，将默认显示组件HelloWorld更换成AppMain，重启项目运行查看效果；

**案例二：组件嵌套**

1. 在views目录中分别定义以下组件：About.vue、Center.Vue、UpdatePwd.vue、Register.vue，用于实现关于我们和用户中心效果；
2. 将定义的组件分别导入到router目录下的index.js中，并定义组件的路由线路，其中将UpdatePwd和Register定义成Center的子组件也就是嵌套组件。**注意：嵌套子组件的path属性不需要加/**
3. 在AppMain.vue中使用<router-link>定义About和Center的路由连接；
4. 在Center.vue中使用<router-link>定义UpdatePwd和Register的路由连接，并定义<router-view>显示子组件内容；**注意：使用嵌套组件时，<router-link>中请使用“/父级路由/子级路由”方式访问子组件；**
5. 在\*.vue文件中<style>标签中使用scoped表示样式只在当前页面使用；

# 第七天：登录+注册

1. 课前准备
2. 使用vue-cli构建项目
3. ElementUI实现登录+注册的静态效果
4. 后台交互

## 课前准备

1. readme.txt
2. css.txt(登录页面的样式效果)
3. CorsFilter.java(axios跨域问题解决方案)

## 使用vue-cli构建项目

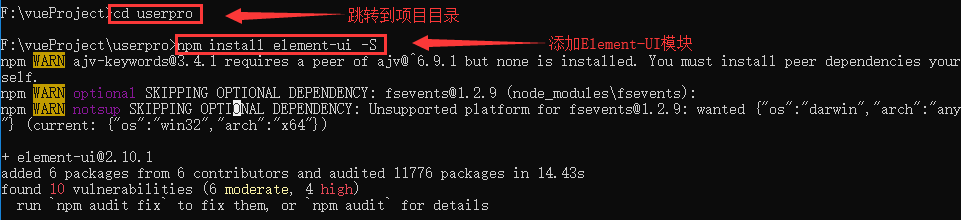
**第一步：**使用CMD命令窗口，并跳转到指定项目目录下；



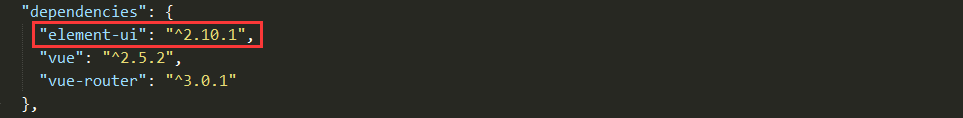
**第二步：**使用命令vue init webpack 项目名，构建一个vue项目；



**第三步：**使用命令npm install element-ui -S，添加Element-UI模块；

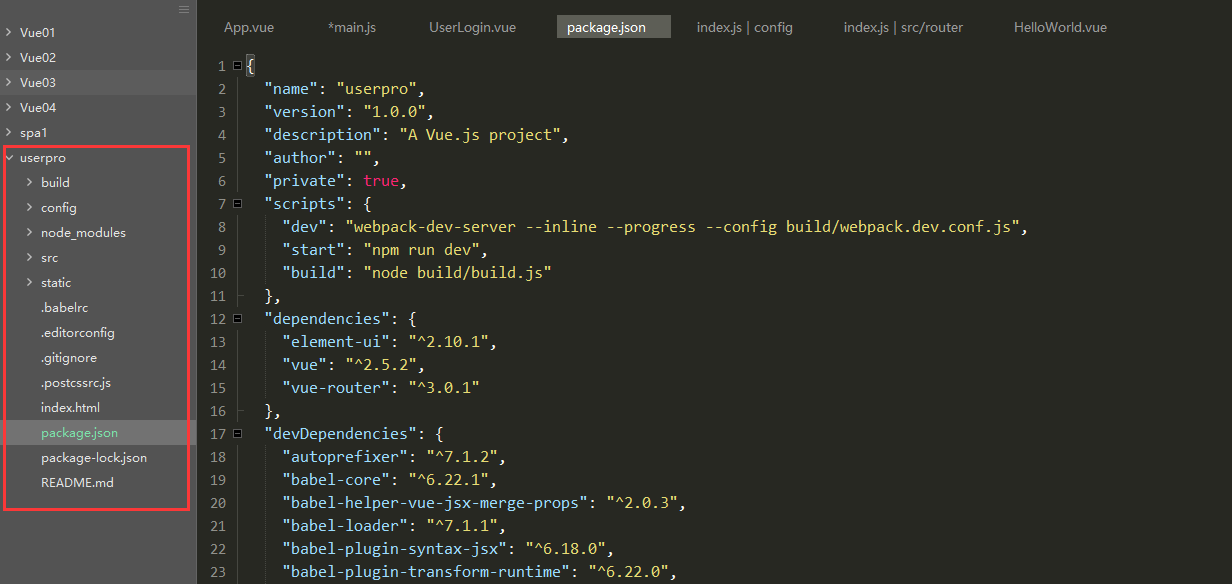


打开项目的package.json文件可查看具体添加模块信息。



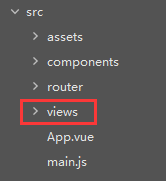
## ElementUI实现登录+注册的静态效果

### 导入项目到HBuilder



### 创建views目录

在src目录下创建views目录，用于存放页面组件。



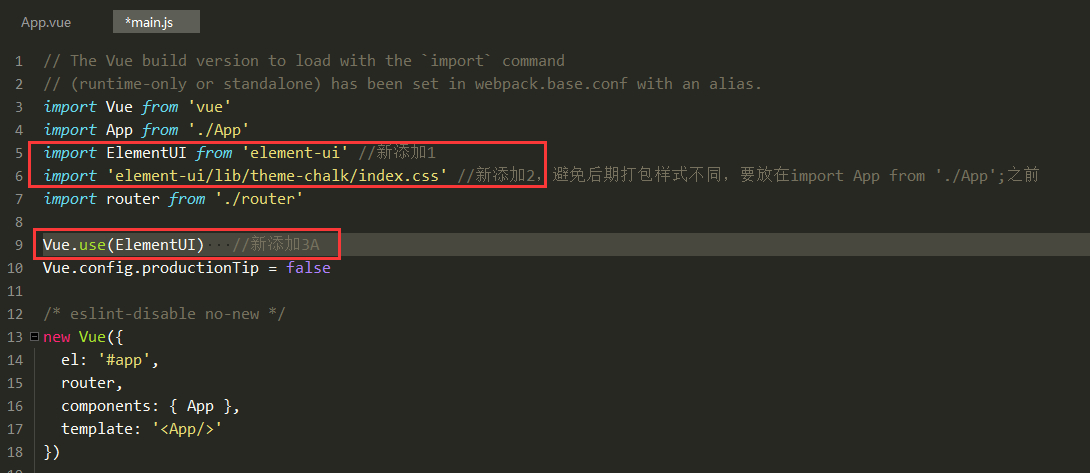
### 在main.js中引入Element-UI

在项目中src目录下找到main.js，并在指定位置添加三行代码（main.js是入口文件，所以在这里引入就行，页面就不用引入了）。

**重要的事情说三遍：在指定位置!!!在指定位置!!!在指定位置!!!~~~添加三行代码**

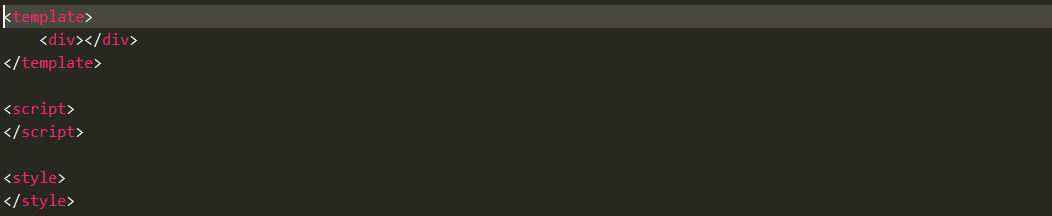
**重要的事情说三遍：在指定位置!!!在指定位置!!!在指定位置!!!~~~添加三行代码**

**重要的事情说三遍：在指定位置!!!在指定位置!!!在指定位置!!!~~~添加三行代码**

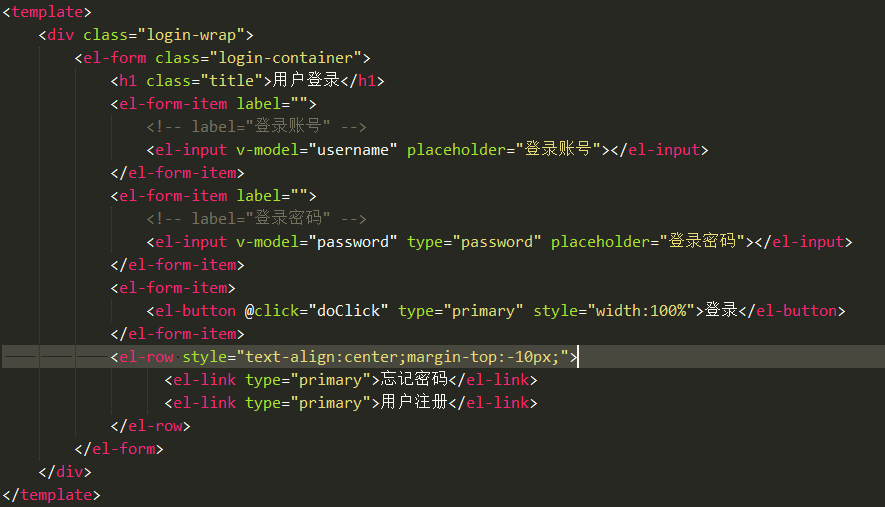


### 创建UserLogin.vue

在views目录下，使用HBuilder创建一个空包的UserLogin.vue模板组件。



在<template>中定义登录+注册静态页面。



在<script>中创建Vue实例化对象。

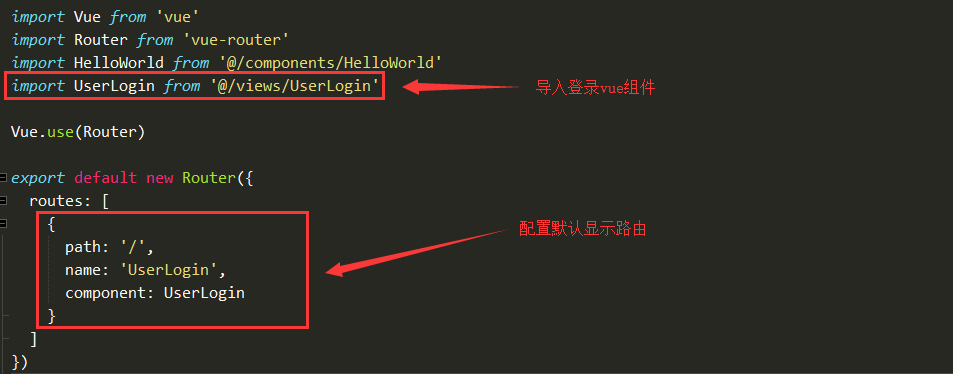


在<style scoped>中定义页面组件的样式。**注：scoped代表只在本页面中使用样式**



### 修改vue项目默认显示路由

在router/index.js中修改vue项目默认显示路由。



### 修改项目端口并启动

在config/index.js目录下修改vue项目运行端口，并启动项目查看运行效果：



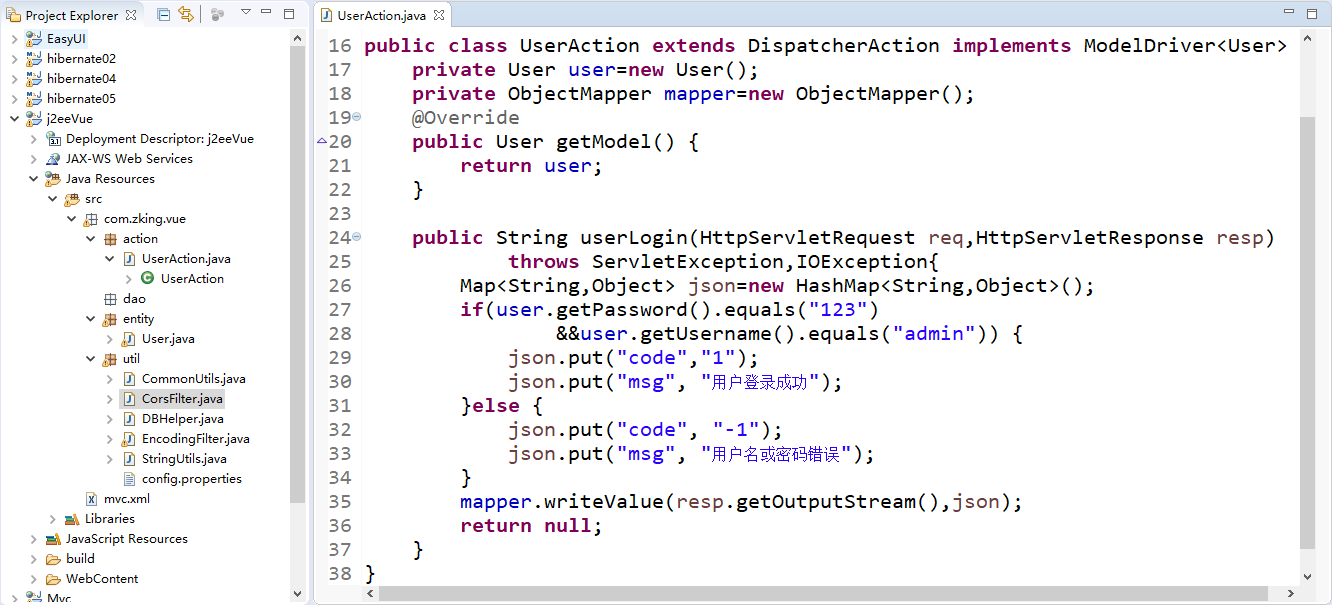
界面运行效果：



## 后台交互

### 使用自定义MVC构建Java后台

使用自定义MVC构建Java后台，模拟提供一个用户登录的action地址，Vue通过请求指定的用户登录action地址判断是否登录成功。

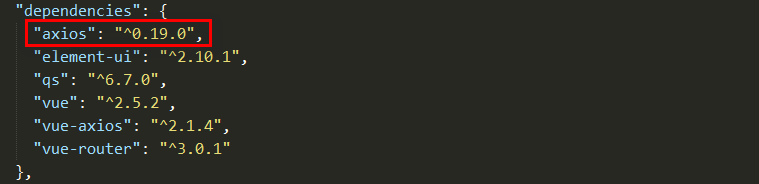


### 安装axios

axios是vue2提倡使用的轻量版的ajax。它是基于promise的HTTP库。它会从浏览器中创建XMLHttpRequests，与Vue配合使用非常好。

安装axios

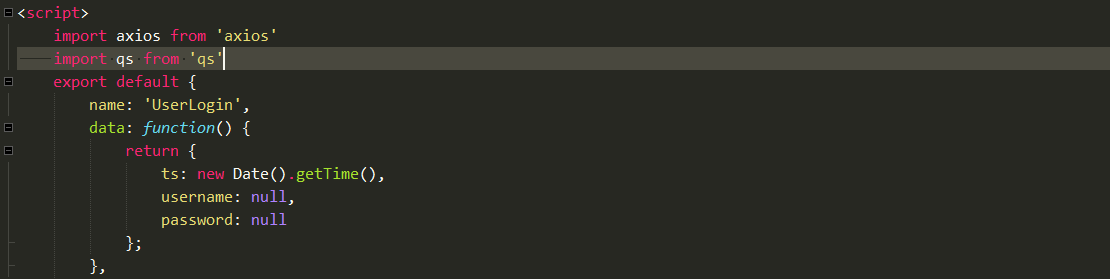
安装成功后，进入vue项目在package.json中查看是否安装成功。



### axios之get请求

在UserLogin.vue中的<script>中导入axios，即可实现当前页面的局部Ajax功能。**在这里请先使用axios的get请求方式，注意参数设置格式，将参数保存到params属性中。**

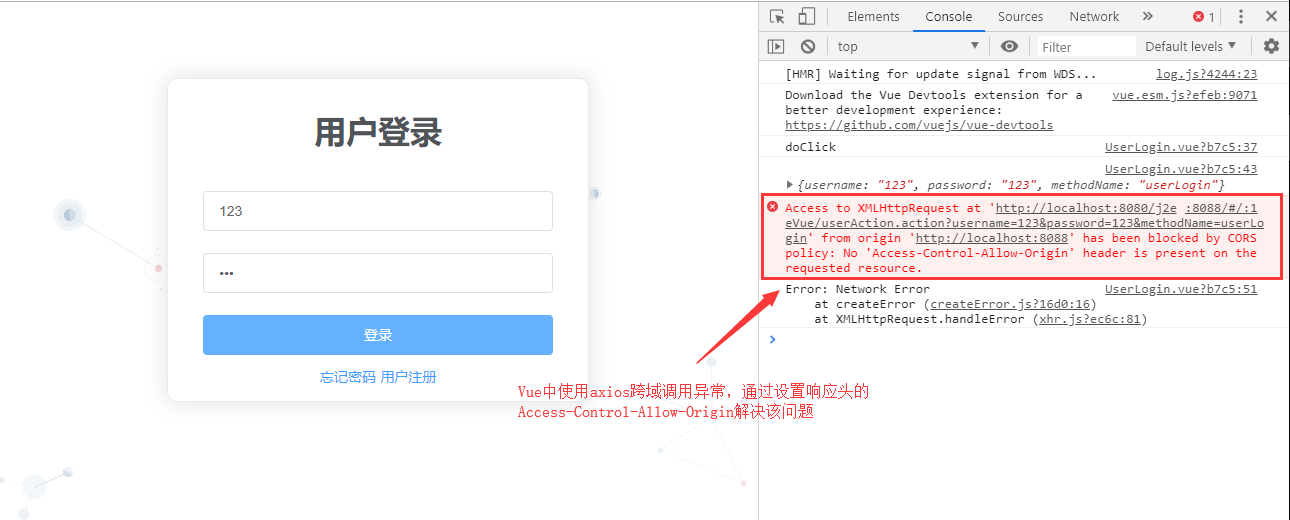
**第一步：配置局部axios**



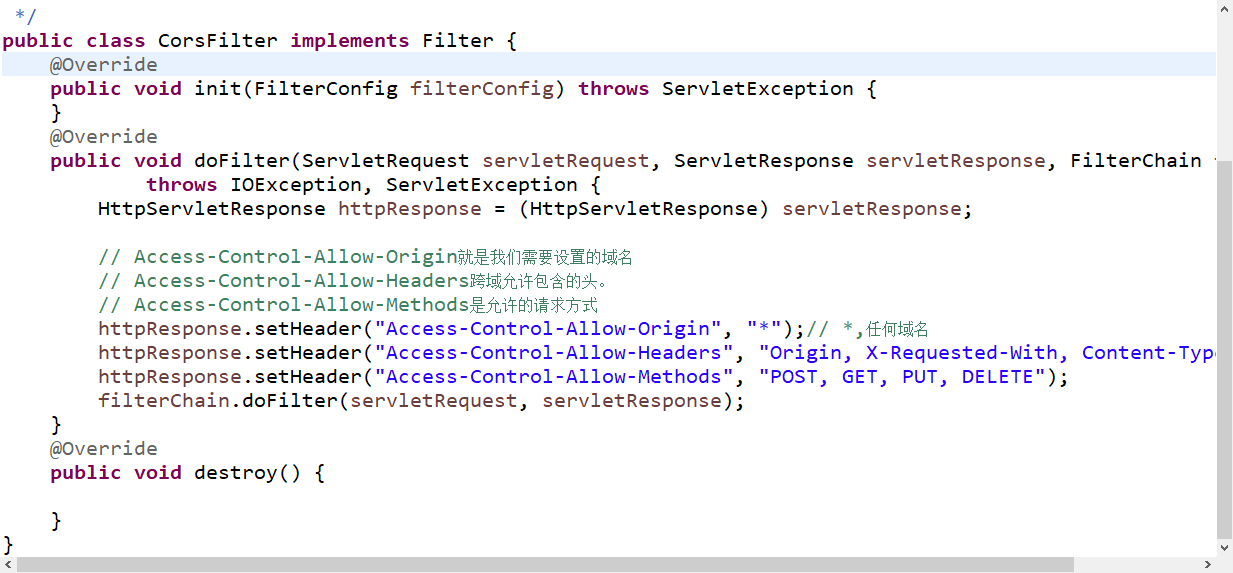
**第二步：使用axios的get请求方式**

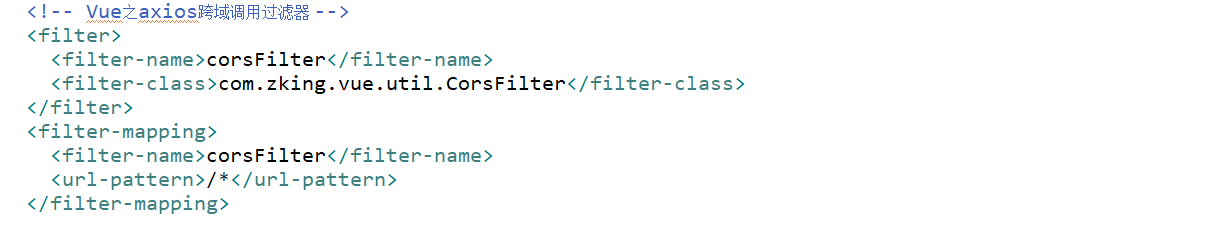


**第三步：运行vue项目，提示ajax跨域异常：**



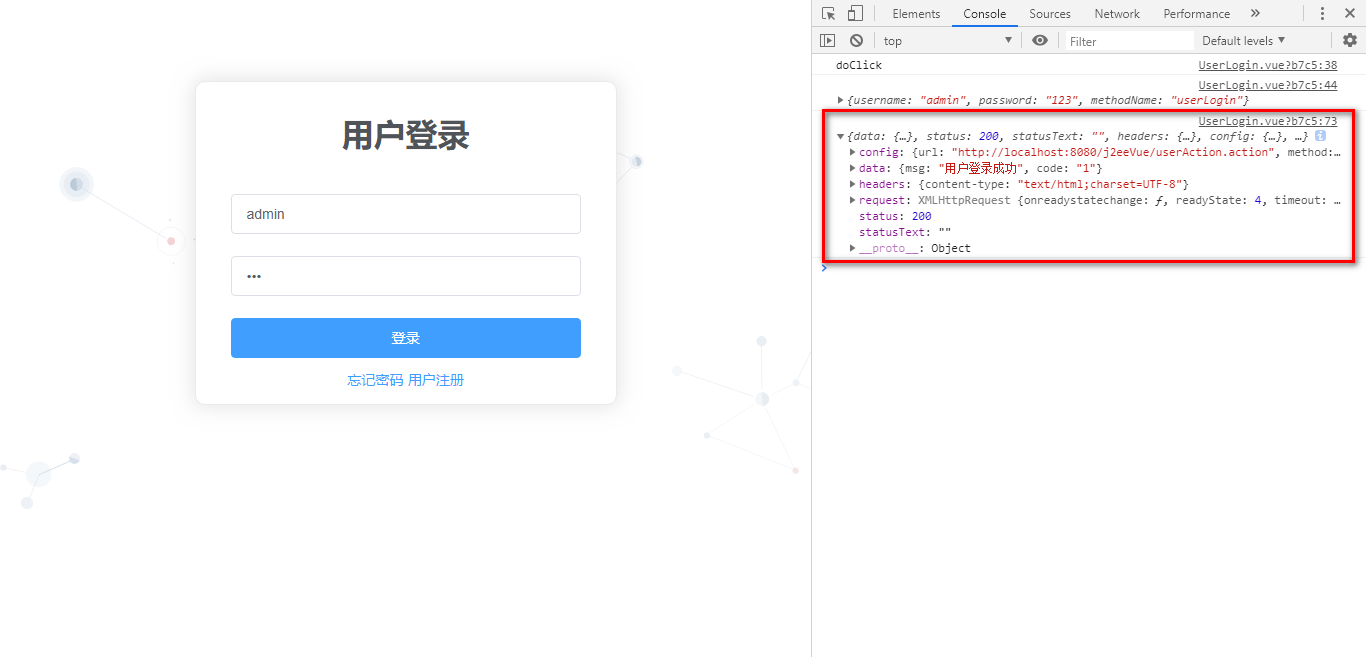
**第四步：在服务端加入CorsFilter过滤器并添加到web.xml中，从而解决ajax跨域问题：**





**注：因为使用了前后端分离开发，跨域访问了。需要配置tomcat允许跨域访问。**

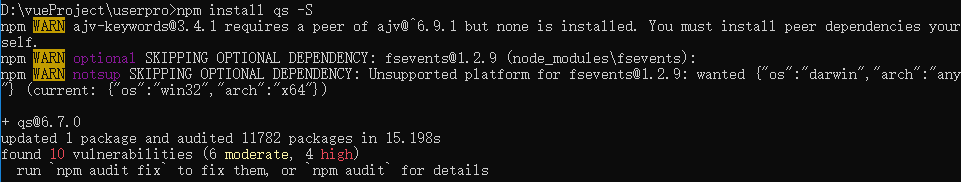
**第五步：再次运行vue项目效果如下：**



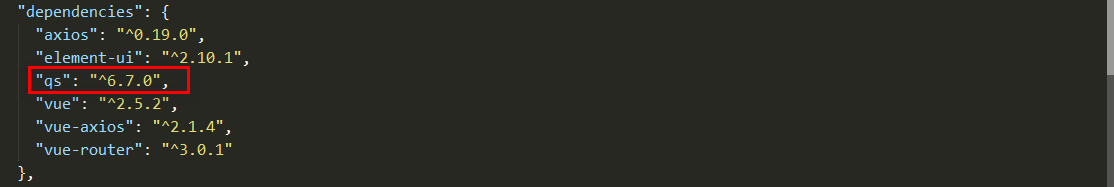
### axios之post请求

axios.get提交没有问题，axios.post提交后台接收不到数据，因为POST提交的参数的格式是Request Payload，这样后台取不到数据的。解决方案：使用qs.js库，将{a:'b',c:'d'}转换成'a=b&c=d'。

**第一步：下载QS模块**

****

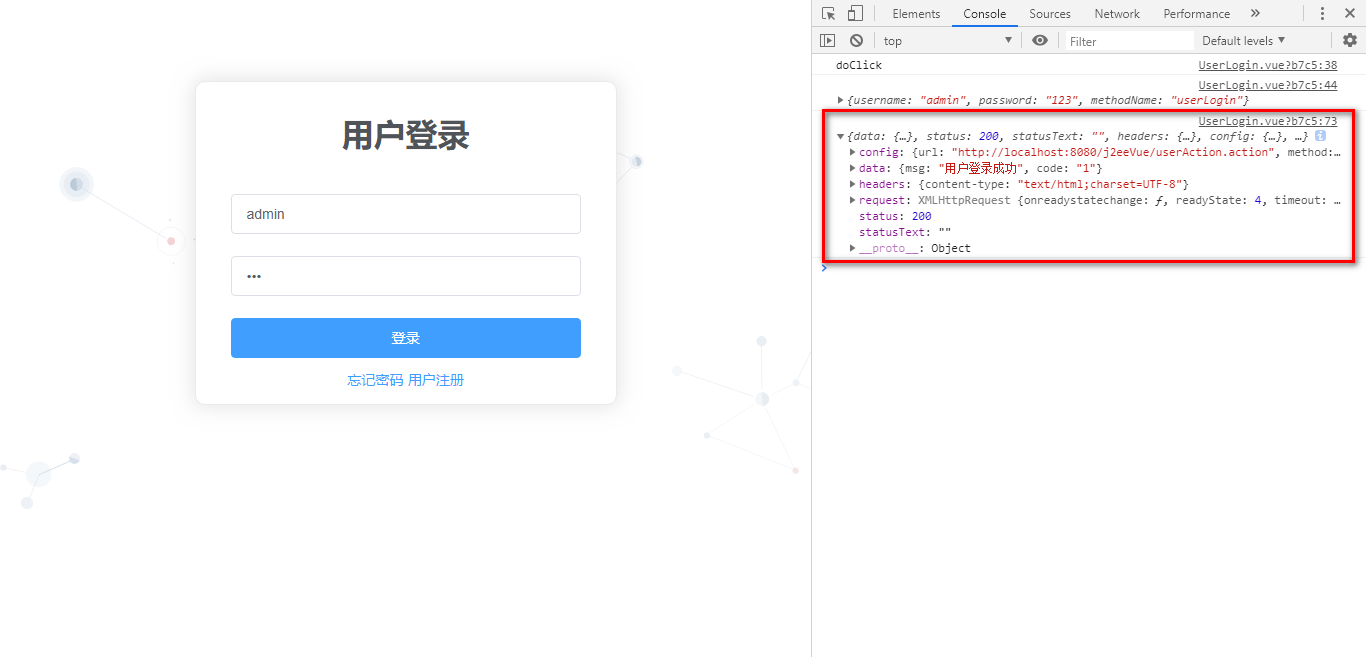
安装成功后，进入vue项目在package.json中查看是否安装成功。



**第二步：在使用axios的post请求提交之前，使用qs库中的qs.stringify(params)方法转换参数格式**

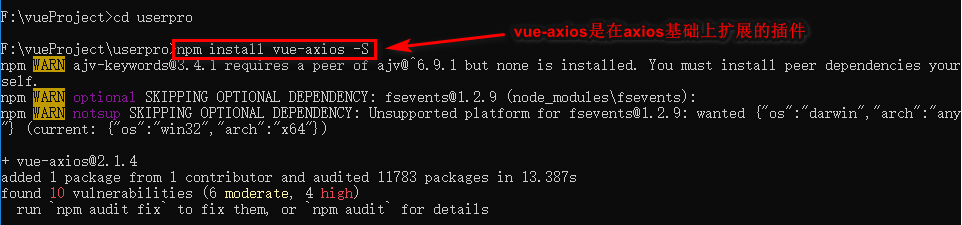


**第三步：运行vue项目展示效果：**



### 安装vue-axios，简化开发

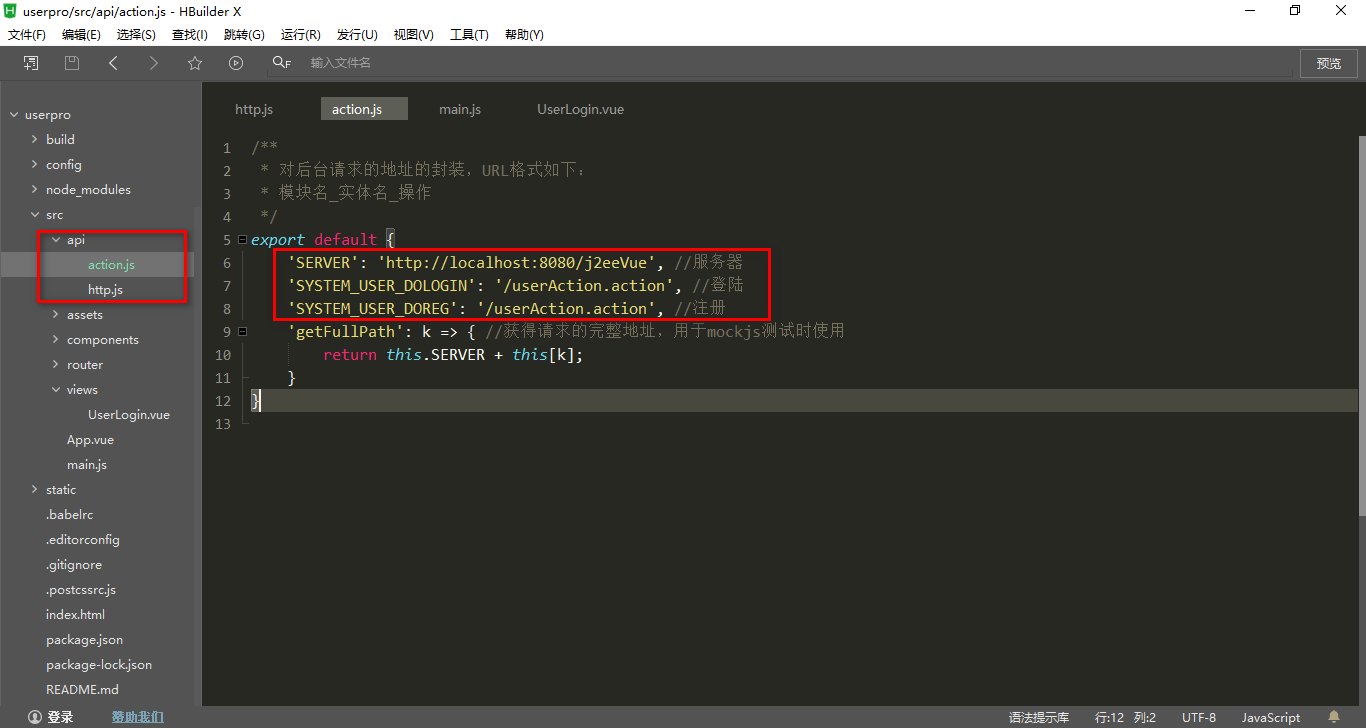
第一步：安装vue-axios



安装成功后，进入vue项目在package.json中查看是否安装成功。



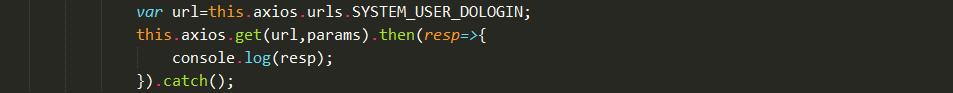
**第二步：在src目录下导入api/action.js、api/http.js，并修改配置action.js**

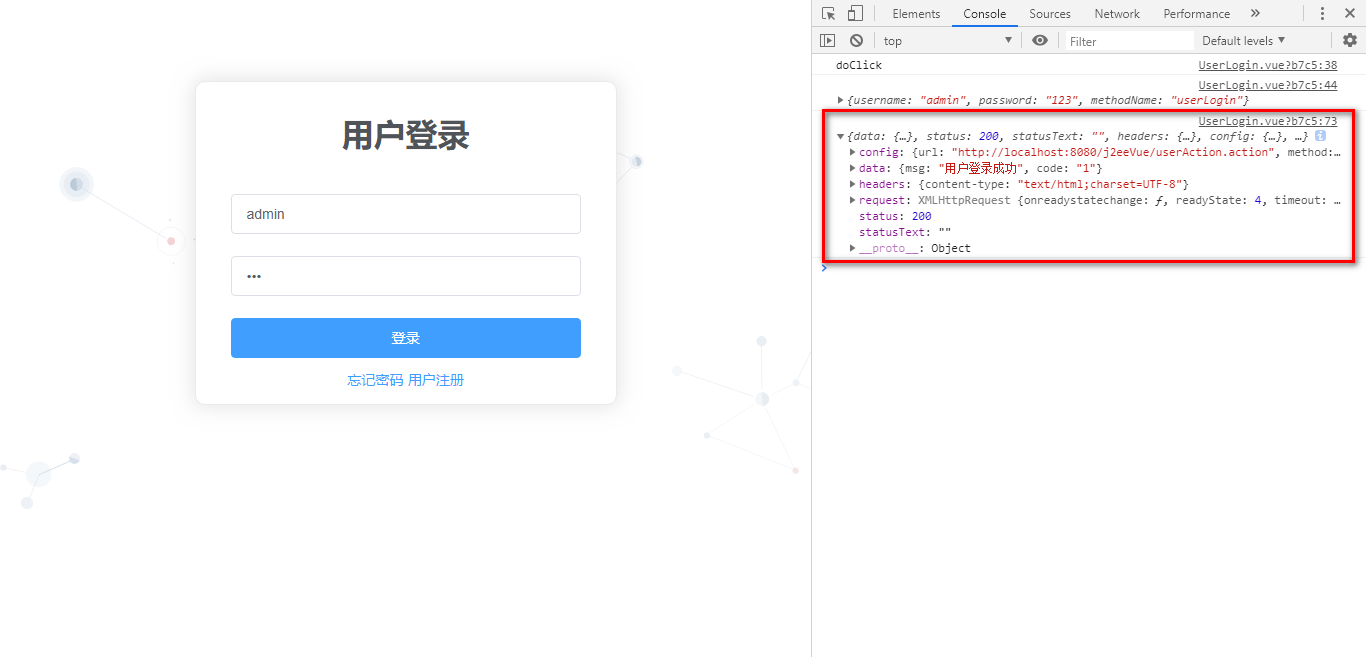


**第三步：在main.js中配置使用vue-axios**



**第三步：使用封装后axios，发送请求**





# 第八天：左侧菜单+导航条

1. 课前准备
2. MockJs
3. 用户注册
4. 左侧菜单+导航条

## 课前准备

1. readme.txt
2. 资料

## MockJs

### 前言

前后端分离开发开发过程当中，经常会遇到以下几个尴尬的场景：

1. 老大，接口文档还没输出，我的好多活干不下去啊！
2. 后端小哥，接口写好了没，我要测试啊！

前后端分离之后，前端迫切需要一种机制，不再需要依赖后端接口开发，而今天的主角mockjs就可以做到这一点。

### 什么是MockJs

Mock.js是一个模拟数据的生成器，用来帮助前端调试开发、进行前后端的原型分离以及用来提高自动化测试效率。众所周知Mock.js因为两个重要的特性风靡前端：

1. 数据类型丰富：支持生成随机的文本、数字、布尔值、日期、邮箱、链接、图片、颜色等。
2. 拦截Ajax请求：不需要修改既有代码，就可以拦截Ajax请求，返回模拟的响应数据。

官网地址：<http://mockjs.com/>

注：easy-mock是一个在线模拟后台的数据平台。

### 安装MockJs

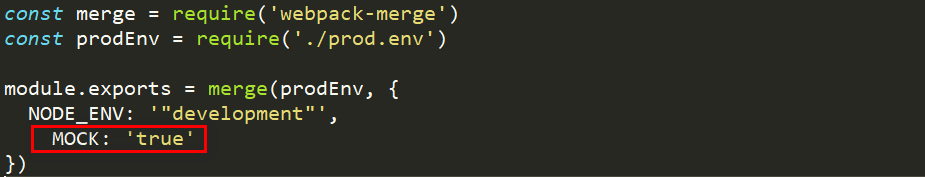
安装MockJs依赖，使用命令：**npm install mockjs -D**，-D表示只在开发环境中使用。

### 配置使用MockJs

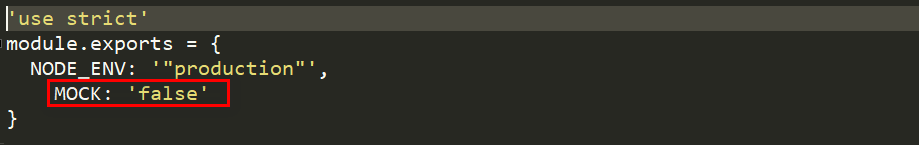
#### 引入MockJs

为了只在开发环境使用mock，而打包到生产环境时自动不使用mock，我们可以在env中做一个配置。

**第一步：配置dev.env（在config/dev.env.js中）**



**第二步：配置prod.env（在config/prod.env.js中）**

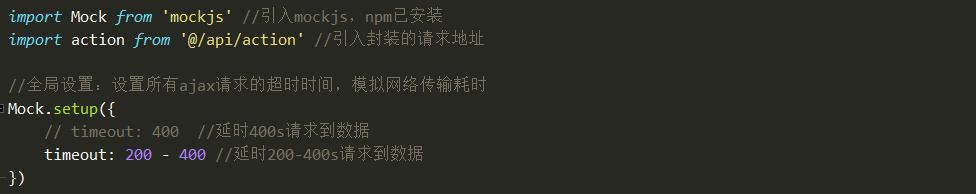


**第三步：引入mockjs（在main.js中）**



#### 目录和文件创建

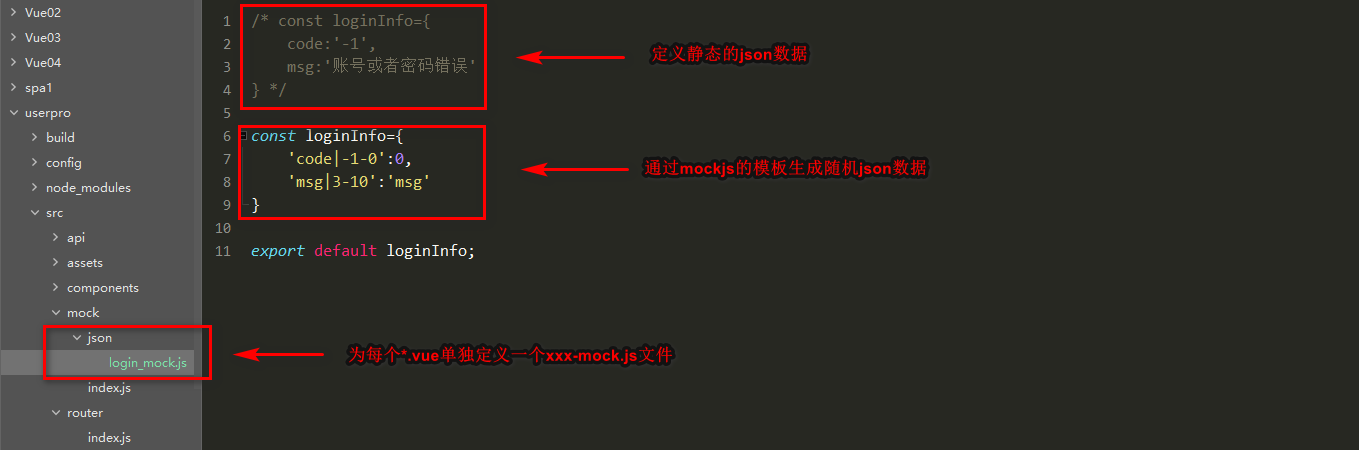
在src目录下创建mock目录，定义mock主文件index.js，并在该文件中定义拦截路由配置。并**导入公共模块及mockjs全局设置**



**注：index.js文件的作用很显然,就是将分散的xxx-mock文件集合起来.后面再添加新的mock文件,都需要在这里引入。**

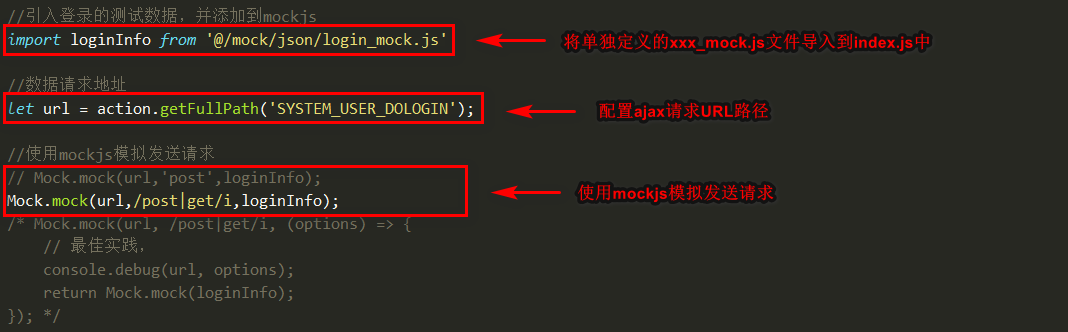
#### 定义xxx-mock.js文件

为每个\*.vue定义单独的xxx-mock.js文件，并在其中添加自定义的json数据，还可以通过mockjs的模块生成随机数据信息，动态测试ajax请求效果。



#### 导入xxx-mock.js

将单独定义xxx-mock.js导入到mock/index.js中，并配置请求路径实现ajax模拟请求效果。



配置完成之后，直接使用axios配置post或get请求，模拟测试效果。

## 用户注册

### 创建用户注册Vue

在views目录下新建一个用户注册vue组件，并定义用户姓名、旧密码及新密码等输入元素，在并最后定义两个按钮，分别是：注册和返回登录。

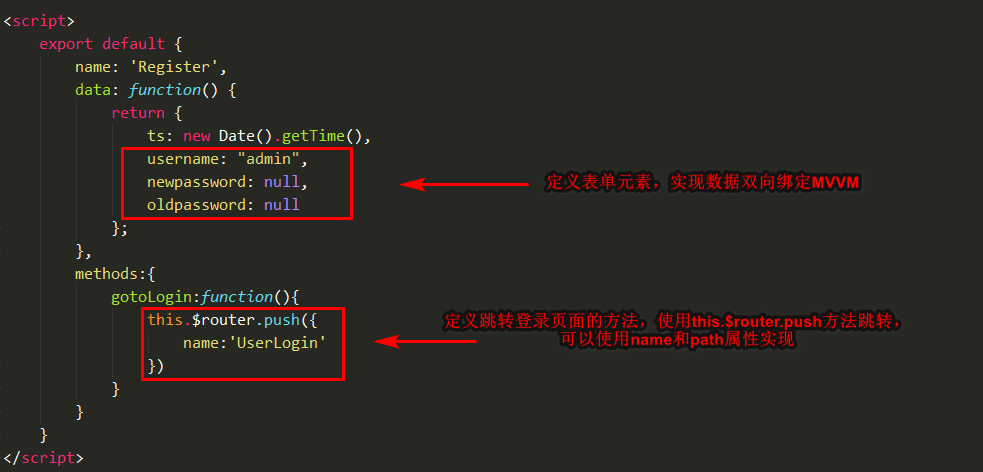
#### Template定义

注册页面定义与登录页面一模一样。



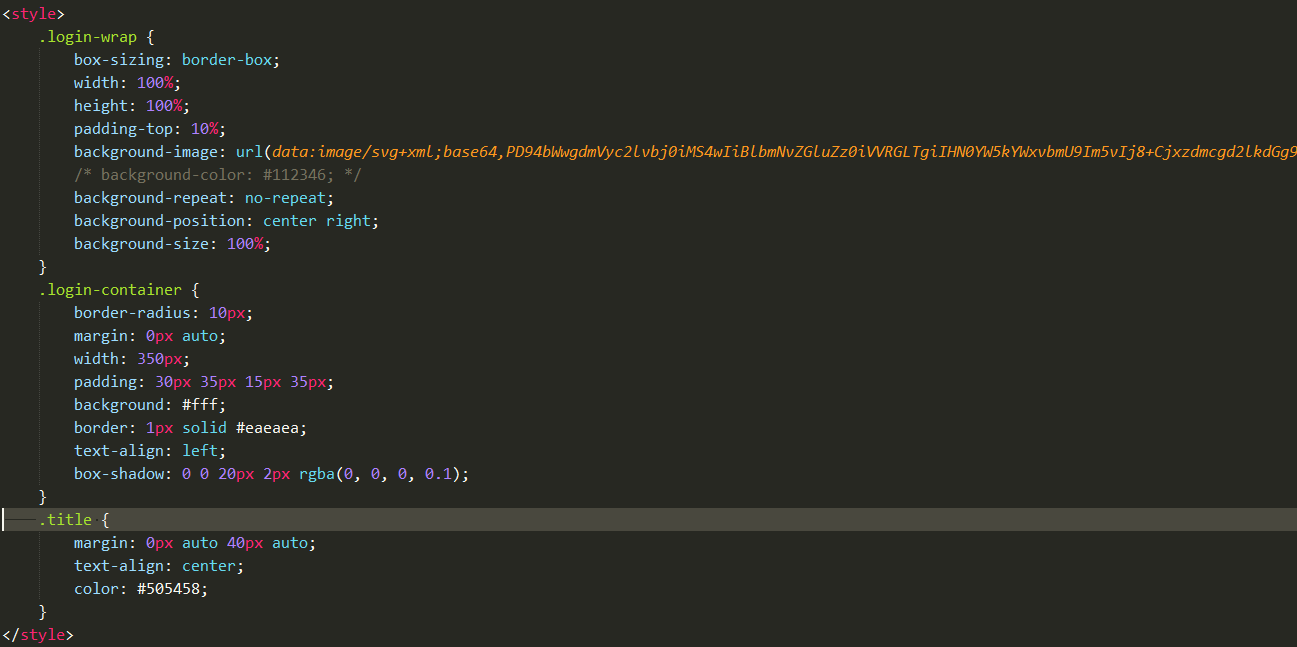
#### Script定义

使用this.$router.push方法实现页面组件的跳转，可以通过push方法中的name和path属性跳转页面。**注意要与router/index.js中的路由设置对应方可实现页面跳转。**



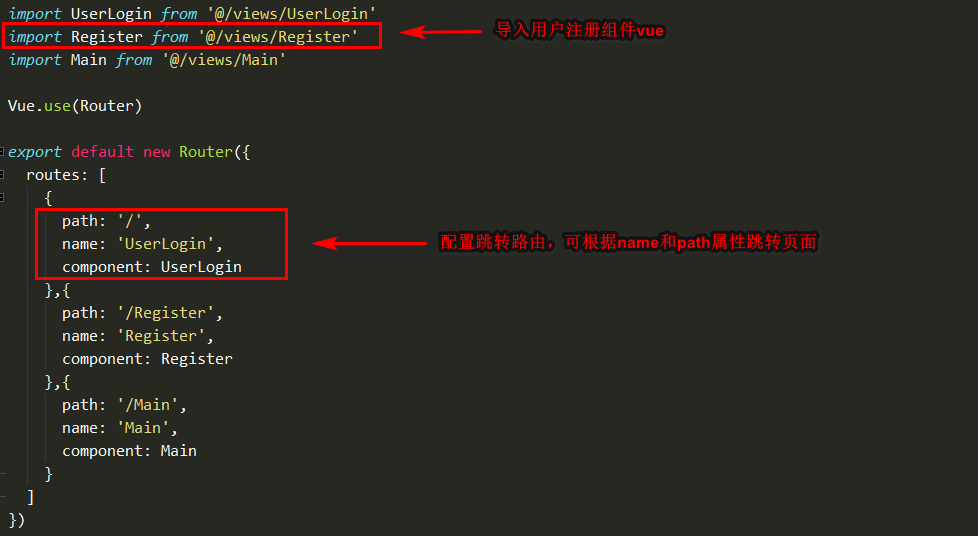
#### Style定义

注册页面Vue组件中的style样式定义与登录页面的style样式定义完全一样。



### 配置路由

在router/index.js中使用import命令导入用户注册组件Register.vue，并在路由器中配置注册页面的跳转路由设置，分别为name、path和component属性，可根据name和path属性跳转页面。



### 运行效果



### this.$router.push跳转组件方式

1. 字符串：this.$router.push('/home/first')
2. 对象：this.$router.push({ path: '/home/first' })
3. 命名的路由：this.$router.push({ name: 'home', params: { userId: wise }})

**问题：this.$router.push、replace、go的区别？**

**1）this.$router.push()：**

跳转到不同的url，但这个方法会向history栈添加一个记录，点击后退会返回到上一个页面

**2）this.$router.replace()：**

同样是跳转到指定的url，但是这个方法不会向history里面添加新的记录，点击返回，会跳转到上上一个页面。上一个记录是不存在的。

**3）this.$router.go(n)：**

相对于当前页面向前或向后跳转多少个页面,类似 window.history.go(n)。n可为正数可为负数。负数返回上一个页面

## 左侧菜单+导航条

### 创建Main组件

在views目录下创建Main组件，用于登录成功后跳转至Main组件。在template中使用element-ui的Container布局容器实现页面布局效果，主要包括：外层容器**(<el-container>)**、顶部容器**(<el-header>)**、侧边栏容器**(<el-aside>)**、主要区域容器**(<el-main>)**和底栏容器**(<el-footer>)**等组成。

#### Template定义

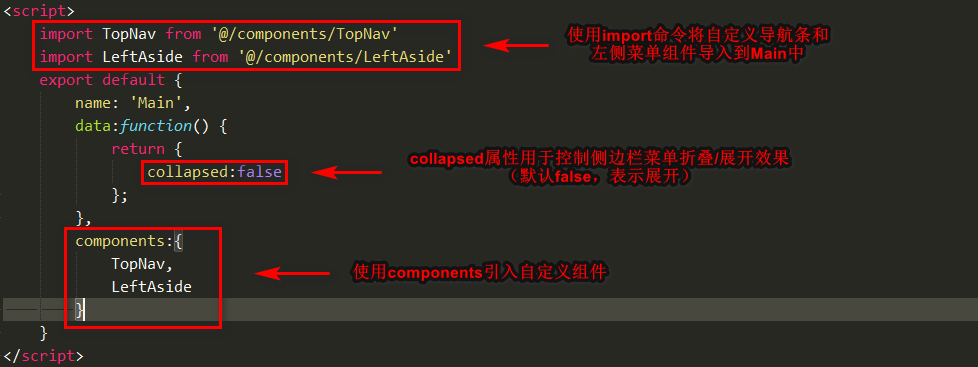
在这里使用的是element-ui的container布局容器中的外层容器、顶部容器、侧边栏容器和主要区域容器实现的布局效果。其中顶部容器用于导入自定义的导航条组件**（TopNav.vue）**，侧边栏容器则用于导入自定义的左侧带单组件**（LeftAside.vue）**。



#### Script定义

使用import命令将自定义导航条和左侧菜单组件导入到Main中，这里采用的局部导入方式，因为自定义导航条和左侧菜单组件只需要在Main中使用。然后使用components将导入的组件引入中Vue实例中。

**注：定义collapsed属性是用于控制左侧侧边栏菜单的折叠和展开，false表示展开、true表示折叠。**



#### Style定义

在style中定义element-ui中container布局容器的样式效果。

**注：使用scoped属性，控制当前组件的style样式只适用于本页面。**



#### 导入图片资源

将课件中的assets目录下的所有图片资源导入到vue项目中src/assets目录下。

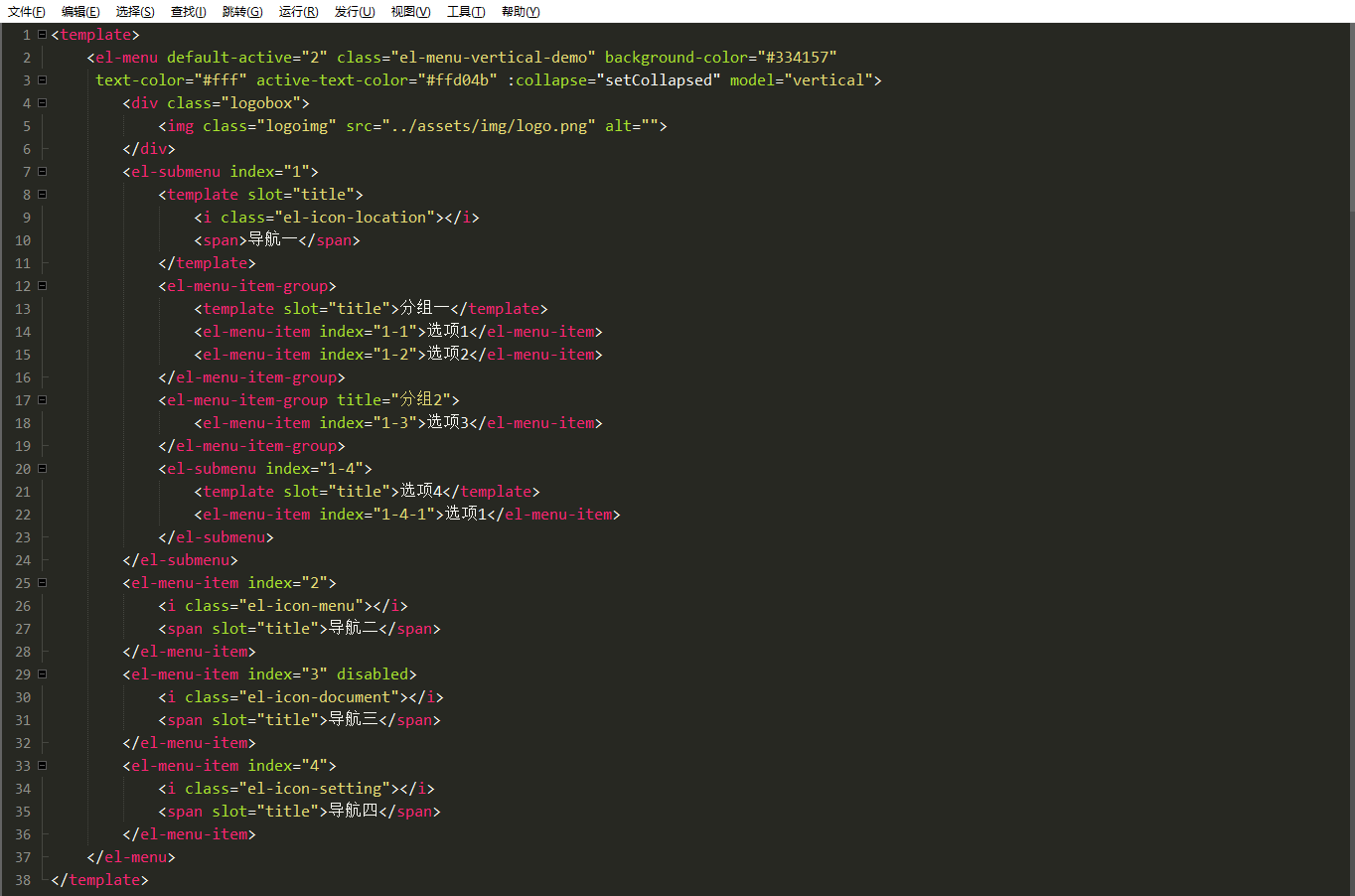
#### 运行效果



### 创建左侧菜单Vue

在components目录下创建左侧菜单Vue组件LeftAside.vue

#### Template定义



#### Script定义



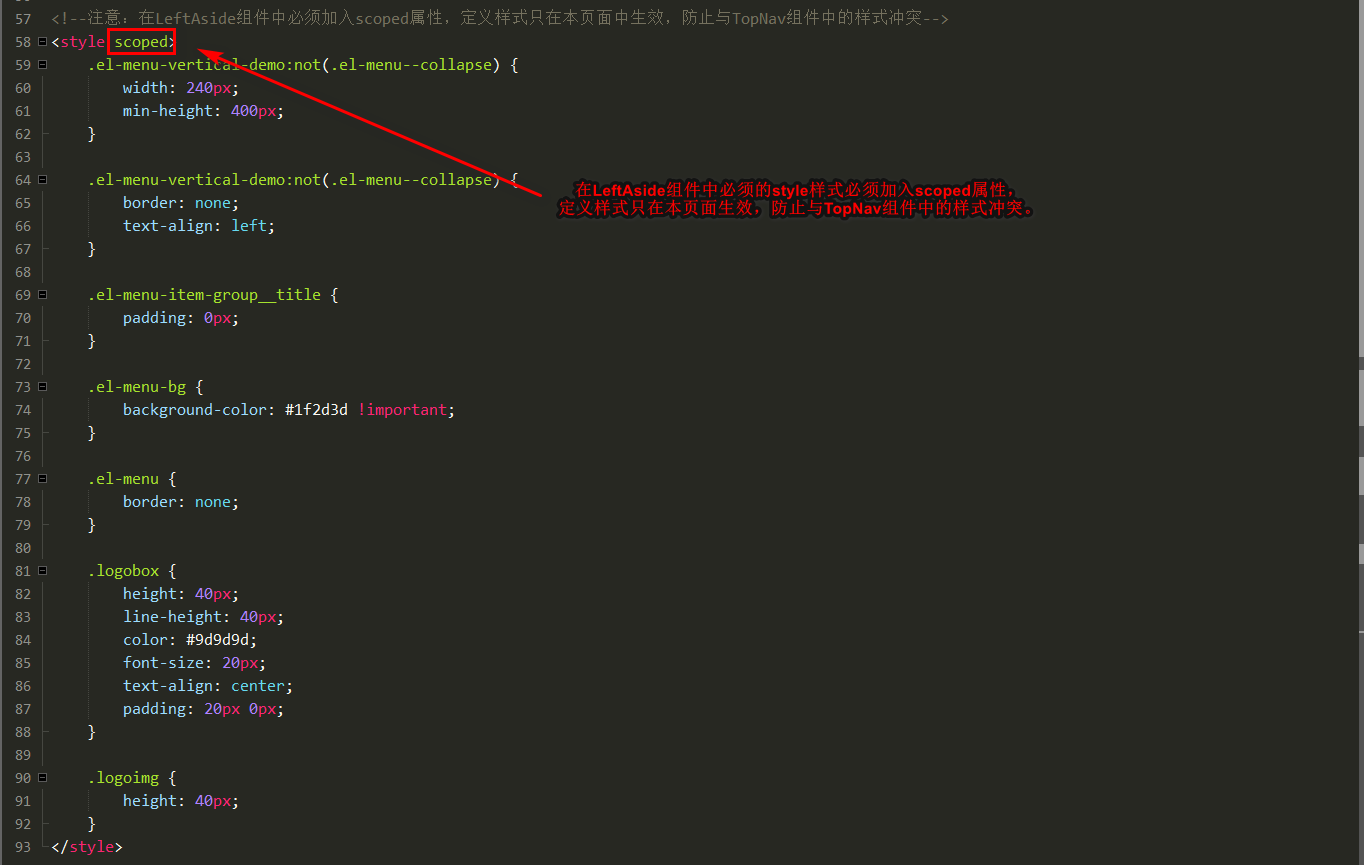
定义props属性，用于实现父组件往子组件传递参数（**在子组件中定义属性必须满足驼峰命名法，在父组件中使用子组件的属性必须满足短横线命名**）；使用computed属性监听方式用于实时监控props中定义属性的状态变化，从而实现左侧菜单组件折叠、展开效果！

#### Style定义

**LeftAside组件的style中必须加入scoped属性，定义样式只在本页面中生效，防止与TopNav组件中的样式冲突！！！**

**LeftAside组件的style中必须加入scoped属性，定义样式只在本页面中生效，防止与TopNav组件中的样式冲突！！！**

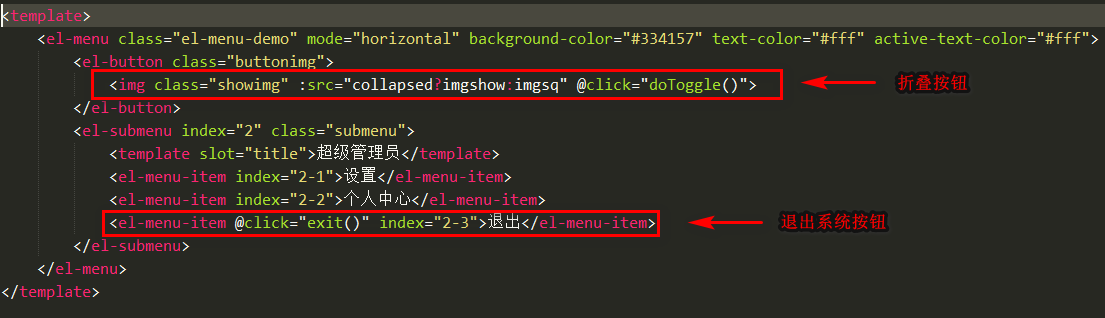
**LeftAside组件的style中必须加入scoped属性，定义样式只在本页面中生效，防止与TopNav组件中的样式冲突！！！**



### 创建导航条Vue

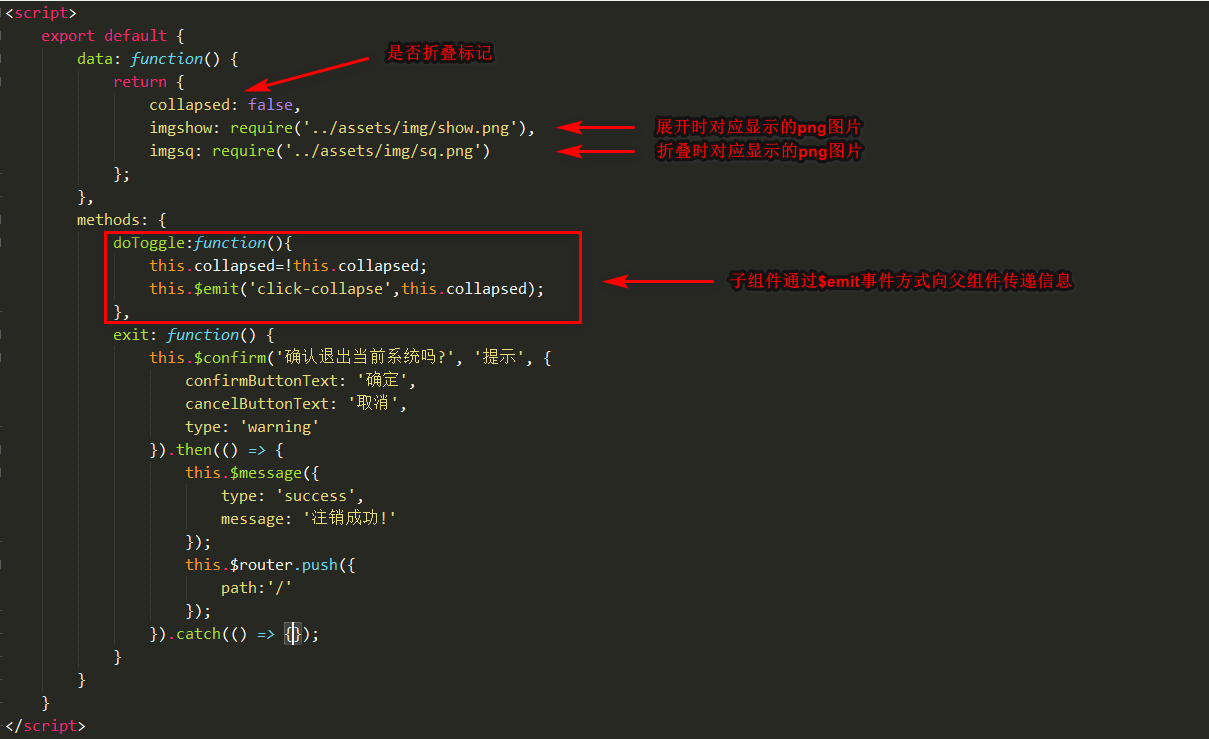
在components目录下创建左侧菜单Vue组件TopNav.vue

#### Template定义



1. 定义导航条的折叠、展开按钮，通过点击按钮实现左侧侧边栏菜单的折叠、展开效果；
2. 定义退出按钮，通过点击按钮实现confirm弹出框提示是否退出系统；

#### Script定义



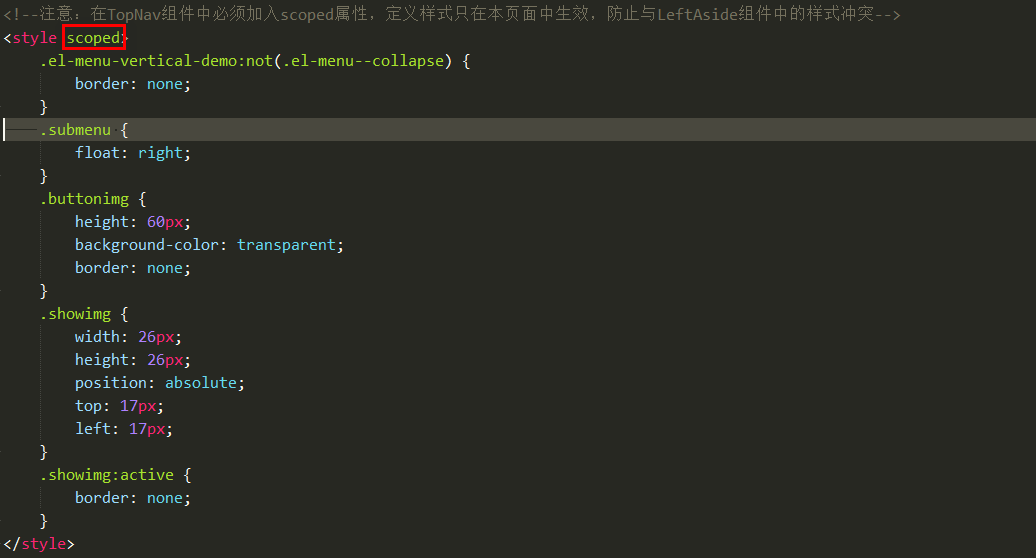
1. 定义collapsed属性表示是否折叠。默认false，表示展开；
2. 在折叠按钮img标签中使用**v-bind:src=”collapsed?imgshow:imgsq”**方式控制按钮折叠、展示时的图片切换显示效果；
3. 定义exit方法实现退出系统功能，通过**this.$router.push({path:’/’})**方式跳转到登录页面；
4. 定义折叠按钮的**v-on:click=”doToggle()”**方法，在方法中通过this.$emit方式实现事件数据传递，从子组件往父组件传递折叠按钮的点击折叠状态collapsed；

#### Style定义

**NavAside组件的style中必须加入scoped属性，定义样式只在本页面中生效，防止与LeftAside组件中的样式冲突！！！**

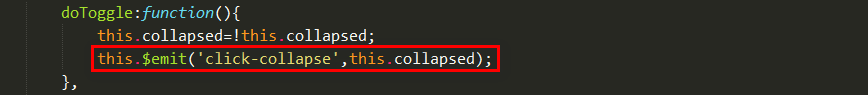
**NavAside组件的style中必须加入scoped属性，定义样式只在本页面中生效，防止与LeftAside组件中的样式冲突！！！**

**NavAside组件的style中必须加入scoped属性，定义样式只在本页面中生效，防止与LeftAside组件中的样式冲突！！！**

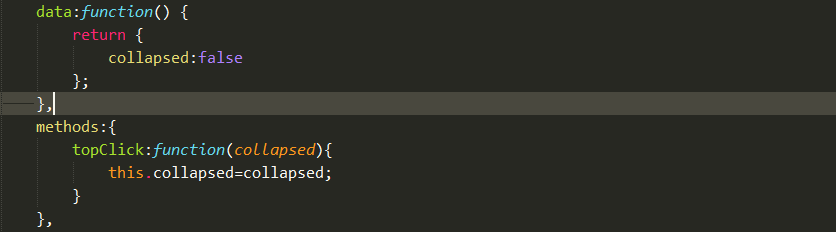


### 实现菜单折叠效果

**第一步：**在TopNav组件的data中定义collapsed属性，表示折叠、展开状态；并通过this.$emit事件方法由子组件往父组件中的传递数据。关键代码：

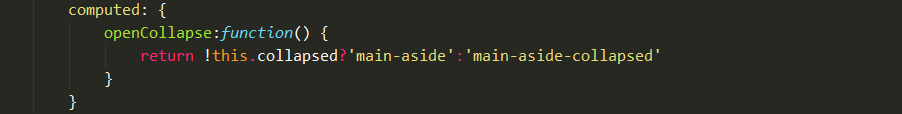


在Main组件导入TopNav组件，并在TopNav组件中通过**<TopNav @click-collapse=”topClick”></TopNav>**方式调用父组件的topClick方法获取子组件中传递过来的参数。关键代码：

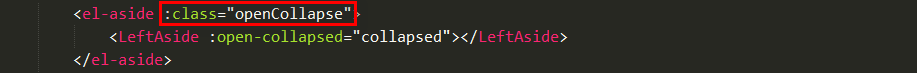


**第二步：**在Main组件中使用computed属性监听方式监控TopNav子组件传递过来的折叠状态collapsed。

1. 使用computed属性监听方式监控collapsed：



1. 在侧边栏<el-aside>中使用v-bind:class方式实时获取collapsed折叠状态：

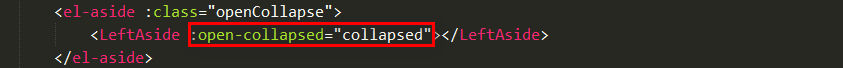


**第三步：**将TopNav组件传递到Main组件的collapsed属性，通过props方式将父组件中的数据传递到子组件中。子组件LeftAside中通过computed属性监控由父组件中传递过来的collapsed折叠状态，并将collapsed属性通过v-bind绑定到<el-menu>中的collapse属性中（注：设置<el-menu>中的model=”vertical”）。

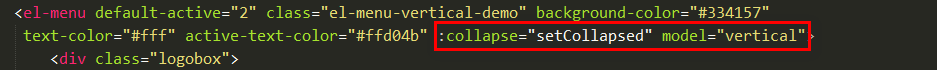
1. 在LeftAside组件中定义props，用于接收父组件传递过来的属性（注意：props中属性定义要使用驼峰命名法）

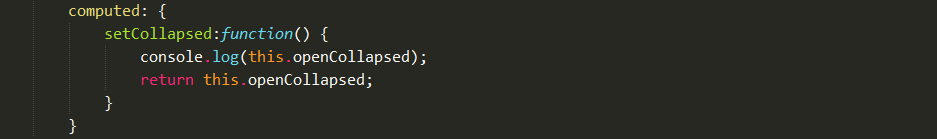


1. 在Main组件中使用LeftAside子组件定义的属性传递动态数据（注意：属性定义要使用短横线命名法）



1. 最后再在LeftAside组件中通过computed属性监听方式实时监控父组件中传递过来的折叠状态，从而控制左侧菜单折叠、展开效果。





# 第九天：动态树+表格+分页

1. 课前准备
2. 后台数据表
3. 动态树（只支持2级）
4. 表格
5. 分页

## 课前准备

1. 后台数据表
2. 后台代码（自定义MVC版）
3. 资料

## 后台数据表

### 后台数据表

t\_user\_vue（用户信息表）、t\_module\_vue（权限信息表）、t\_book\_vue（书本信息表）



### 服务端Demo

**这里服务端采用的自定义MVC模式，真实情况下需要使用SSH2框架来完成。**

1. **提供动态树的后台请求方法，采用递归绑定方式（只支持2级菜单）**

|  |
| --- |
| /\*\*  \* 根据父类id查询子类pid的数据（递归）  \* **@param** pid  \* **@param** mod  \*/  **public** **void** queryModuleByPid(Integer pid,Module mod){  String sql="select id,pid,text,icon,url,sort from t\_module\_vue where"  + " pid="+pid;  List<Module> lst=**super**.executeQuery(sql, **null**, **new** CallBack<Module>() {  @Override  **public** List<Module> forEach(ResultSet rs) **throws** SQLException {  List<Module> lst=**new** ArrayList<Module>();  Module node=**null**;  **while**(rs.next()) {  node=**new** Module();  **int** id=rs.getInt("id");  node.setId(id);  node.setPid(rs.getInt("pid"));  node.setText(rs.getString("text"));  node.setIcon(rs.getString("icon"));  node.setUrl(rs.getString("url"));  lst.add(node);  //queryModuleByPid(id, root);  }  **return** lst;  }  });  mod.setChildren(lst);  }    /\*\*  \* 获取Tree根节点  \* **@return**  \*/  **public** List<Module> queryRootNode(){  String sql="select id,pid,text,icon,url,sort from t\_module\_vue where"  + " pid=-1";  **return** **super**.executeQuery(sql, **null**, **new** CallBack<Module>() {  @Override  **public** List<Module> forEach(ResultSet rs) **throws** SQLException {  List<Module> lst=**new** ArrayList<Module>();  Module root=**null**;  **while**(rs.next()) {  root=**new** Module();  **int** id=rs.getInt("id");  root.setId(id);  root.setPid(rs.getInt("pid"));  root.setText(rs.getString("text"));  root.setIcon(rs.getString("icon"));  root.setUrl(rs.getString("url"));  lst.add(root);  queryModuleByPid(id, root);  }  **return** lst;  }  });  } |

1. **提供数据表格分页查询方法**

|  |
| --- |
| **public** List<Book> queryBookPager(Book book,PageBean pageBean){  String sql="select id,bookname,price,booktype from t\_book where 1=1";  **if**(StringUtils.*isNotBlank*(book.getBookname()))  sql+=" and bookname like '%"+book.getBookname()+"%'";  sql+=" order by id desc";    **return** **super**.executeQuery(sql, pageBean, **new** CallBack<Book>() {  @Override  **public** List<Book> forEach(ResultSet rs) **throws** SQLException {  **return** CommonUtils.*toList*(rs, Book.**class**);  }  });  } |

## 动态树

登录成功后，主页侧边栏采用动态树的方式进行数据绑定。在这里需要使用element-ui中的NavMenu导航菜单的侧栏方式实现。**注：后台采用递归方式进行数据绑定，这里只支持2级菜单即可。**

**结构说明：**

**<el-menu>：**NavMenu导航菜单根节点标签；

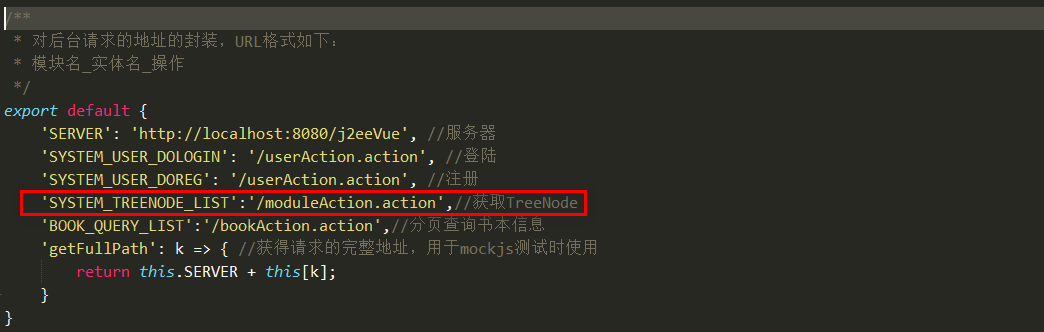
**<el-submenu>：**NavMenu导航菜单子节点标签，下级可以出现子叶节点标签；

**<el-menu-item>：**NavMenu导航菜单叶节点标签；

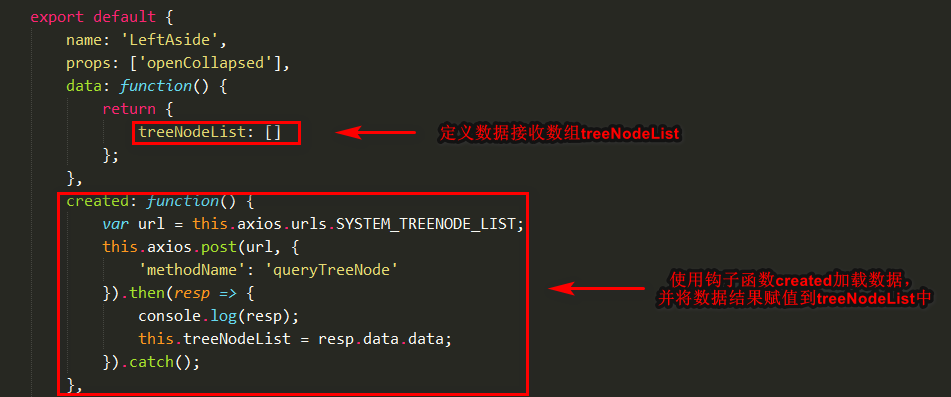
### 加载数据

采用vue实例中的钩子函数**created**，在vue实例加载渲染完成之后调用Axios请求后台数据进行左侧菜单栏初始化加载数据操作。

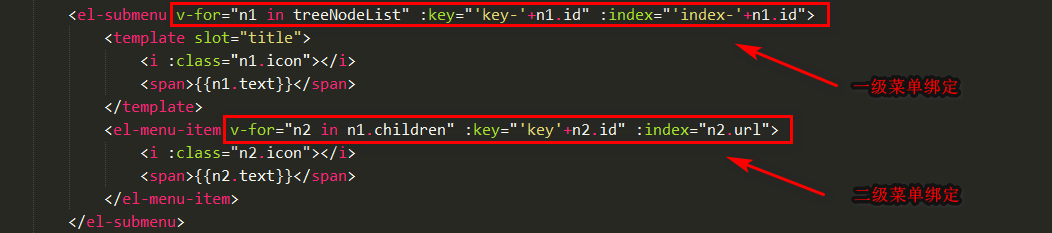
1. **在/api/action.js中定义数据请求路径**



1. **在侧边栏组件LeftAside.vue中调用created方法加载数据**



### 数据绑定



注：

1. 采用v-for方式循环绑定各级菜单节点信息；
2. 可在<i>标签中的使用v-bind:class方式动态绑定节点图标；
3. 使用v-bind方式绑定**<el-submenu>**和**<el-menu-item>**的key。key属性设置为菜单节点的id属性值（对应数据表t\_module中的id列），确保唯一；
4. 采用v-bind方式绑定**<el-submenu>**和**<el-menu-item>**的index。其中一级菜单**<el-submenu>**的index属性绑定菜单节点的id属性值（对应数据表t\_module中的id列）；其中二级菜单**<el-menu-item>**的index属性绑定菜单节点的url属性值（对应数据表t\_module中的url列），**而二级菜单的页面跳转是根据index属性来决定的，所以要确保有值且唯一。**

### 页面跳转及当前项设置

**页面跳转：**要实现路由跳转，先要在el-menu标签上添加router属性，然后只要在每个**<el-menu-item>**标签内的index属性设置一下url即可实现点击el-menu-item实现路由跳转。

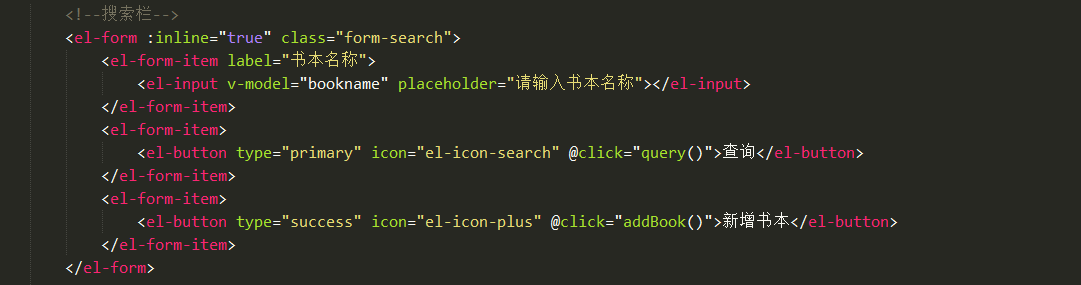


**当前项设置：**导航当前项，在el-menu标签中绑定:default-active="$route.path",注意是绑定属性，不要忘了加“:”，当$route.path等于el-menu-item标签中的index属性值时则该item为当前项。

## 表格

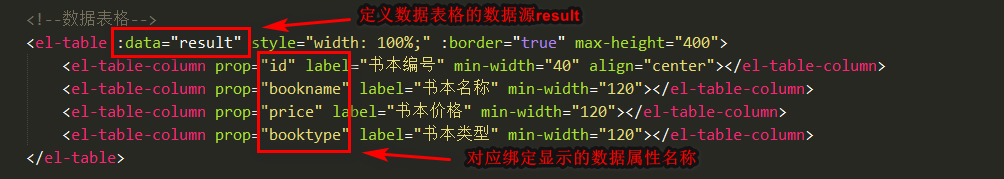
在views目标下创建book目录，并在book目录下创建BookList.vue组件。然后在/router/index.js中配置组件路由，将BookList组件加入到Main组件的children属性中。**子父组件路由设置采用children方式。**

### 搜索栏



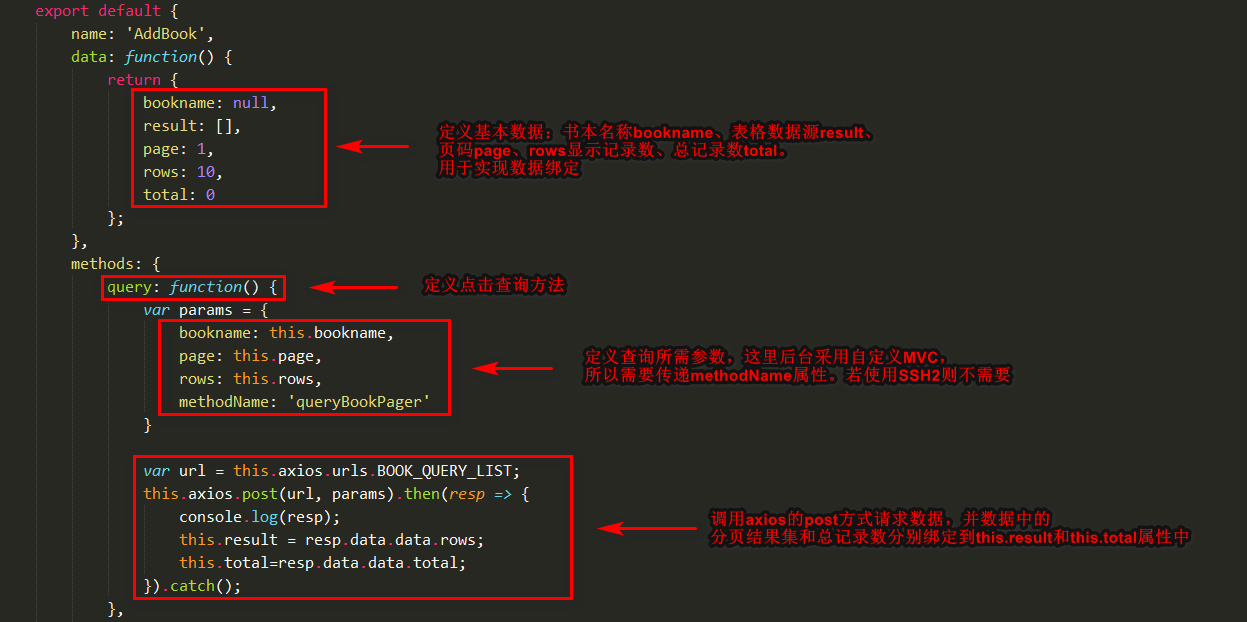
定义query()和addBook()方法，从而实现点击查询书本信息和新增书本功能。

### 数据表格

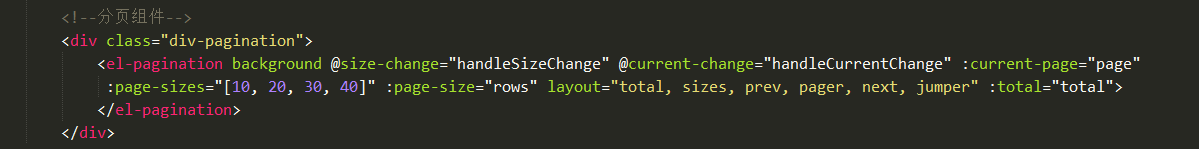


### 数据绑定

数据绑定可分为2种方式进行绑定。第一种直接使用created钩子函数在页面组件初始化的时候即实现进入查询数据效果；第二种采用按钮点击事件方式实现查询数据效果（本章使用该种方式）；



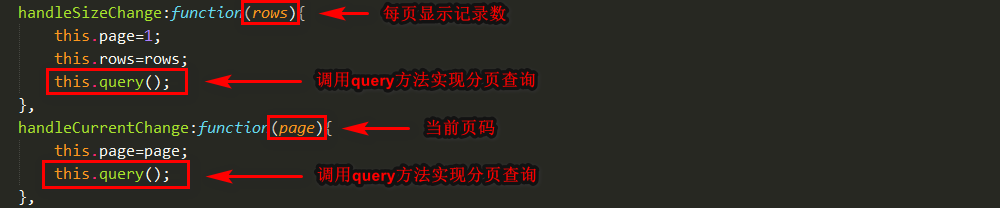
## 分页



**补充说明：**

1. 定义size-change事件，用于监听监听分页显示记录数改变后实现数据分页效果；
2. 定义current-change事件，用于监听分页页码改变后实现数据分页效果；
3. current-page：当前页码，对应data属性中定义的page属性，使用v-bind绑定；
4. page-sizes：用于控制页面显示记录数，是一个数组数据；
5. page-size：每页显示记录数，对应data属性中定义的rows属性，使用v-bind绑定；
6. total：总记录数，对应data属性中定义的total属性，使用v-bind绑定；

**Size-change和current-change事件定义：**



# 第十天：CUD+表单验证

1. 课前准备
2. 后台数据接口准备
3. 弹出框
4. 表单
5. CUD

## 课前准备

1）readme.txt

## 后台数据接口准备

新增、修改以及删除后台数据接口

## 弹出框

### 基本结构

|  |
| --- |
| <el-dialog :title="title" :visible.sync="dialogFormVisible" @close="dialogClose">  ...  </el-dialog> |

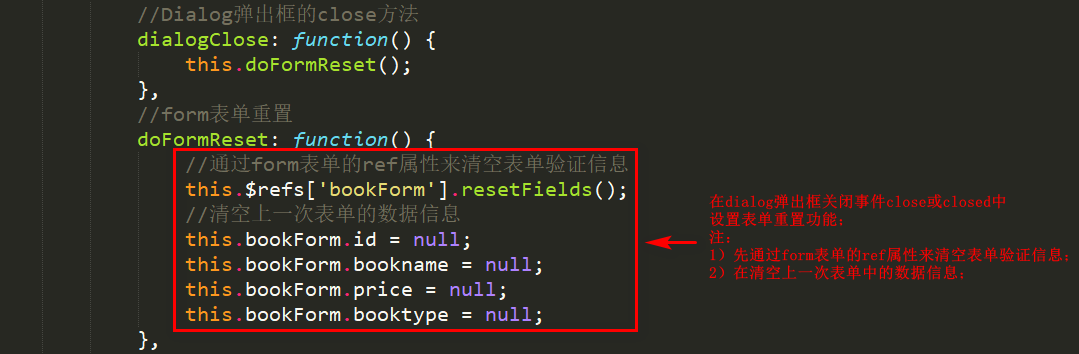
**补充说明：**

1. 隐藏显示设置，通过Vue实例对象中的dialogFormVisible="true|false"来控制dialog显示隐藏；

**:visible.sync="dialogFormVisible"**

1. 通过close或closed事件，在关闭dialog弹出框时清空form表单数据和验证信息；

**@close="dialogClose"**

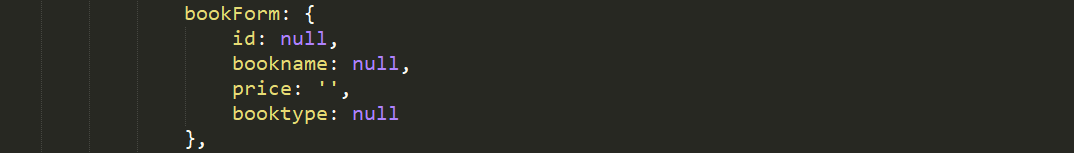


## 表单

### 基本结构

|  |
| --- |
| <el-form :model="bookForm" :label-position="labelPosition">  <el-form-item label="活动名称" :label-width="formLabelWidth">  <el-input v-model="bookForm.name" autocomplete="off"></el-input>  </el-form-item>  ...  </el-form> |

**Vue实例中的form表单数据对象定义：**



**补充说明：**

1. **<el-form:label-position="labelPosition">**设置表单对齐方式：left/center/right；
2. 通过**<el-form :model="bookForm">**设置表单元素属性双向绑定的对象属性名称；
3. 在el-input/el-select等elemnt-ui表单元素中使用**v-model="bookForm.name"**绑定元素属性值；

**注：在el-form中使用model属性绑定表单数据对象，并在form表单中的表单元素通过表单数据对象.属性名的方式进行数据绑定。**

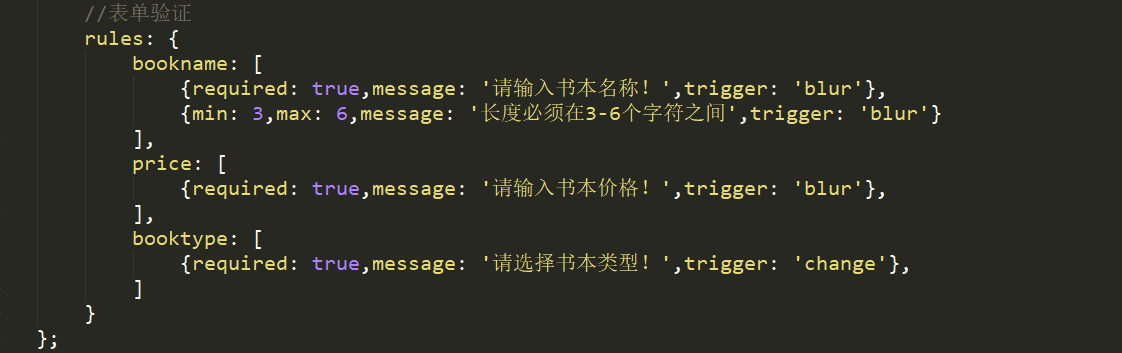
### 表单验证

|  |
| --- |
| <el-form :model="bookForm" **:rules="rules" :ref="bookForm"**>  <el-form-item label="活动名称" :label-width="formLabelWidth">  <el-input v-model="bookForm.name" **prop="bookname"** autocomplete="off"></el-input>  </el-form-item>  ...  </el-form> |

**补充说明：**

1. **:rules="rules"**表单验证定义，只需要通过 rules 属性传入约定的验证规则。并将el-form-item的prop属性设置为需校验的字段名即可。

**在Vue实例的data属性中定义表单验证规则，如下：**



1. 有多个表单，怎么在提交进行区分？

我们在rules这里写了对表单的验证规则，但是我们如何在methods里进行指定的表单进行认证，所以我们一开始就在el-form里写了 ref="ruleForm"，我们在methods里就可以用。

1. 清空表单验证信息

**this.$refs[formName].resetFields();**

**注：formName是指<el-form ref="bookForm">中定义的ref属性名**

1. 表单重置时必须先清空表单验证信息，再清空表单数据信息



## CUD

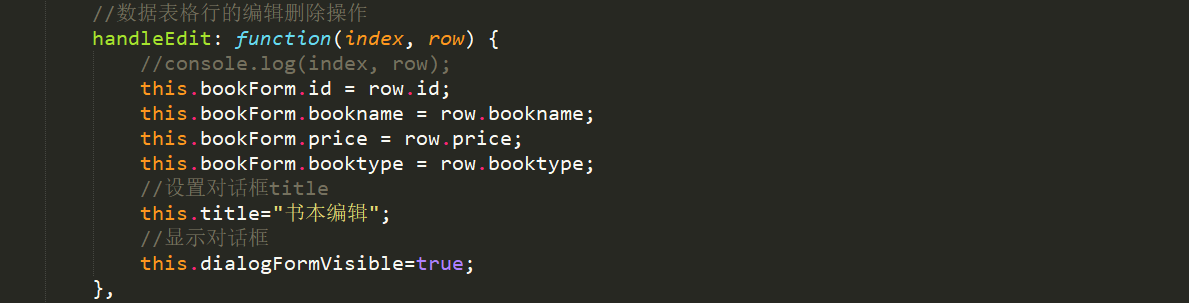
### 表格编辑删除操作

首先在数据表格中的最后一列添加一个表格编辑删除操作一栏。



定义表格行编辑和删除方法：**handleEdit**和**handleDelete**。

**handleEdit**行编辑方法，点击打开dialog弹出框，将修改信息赋值到子页面中表单元素中：

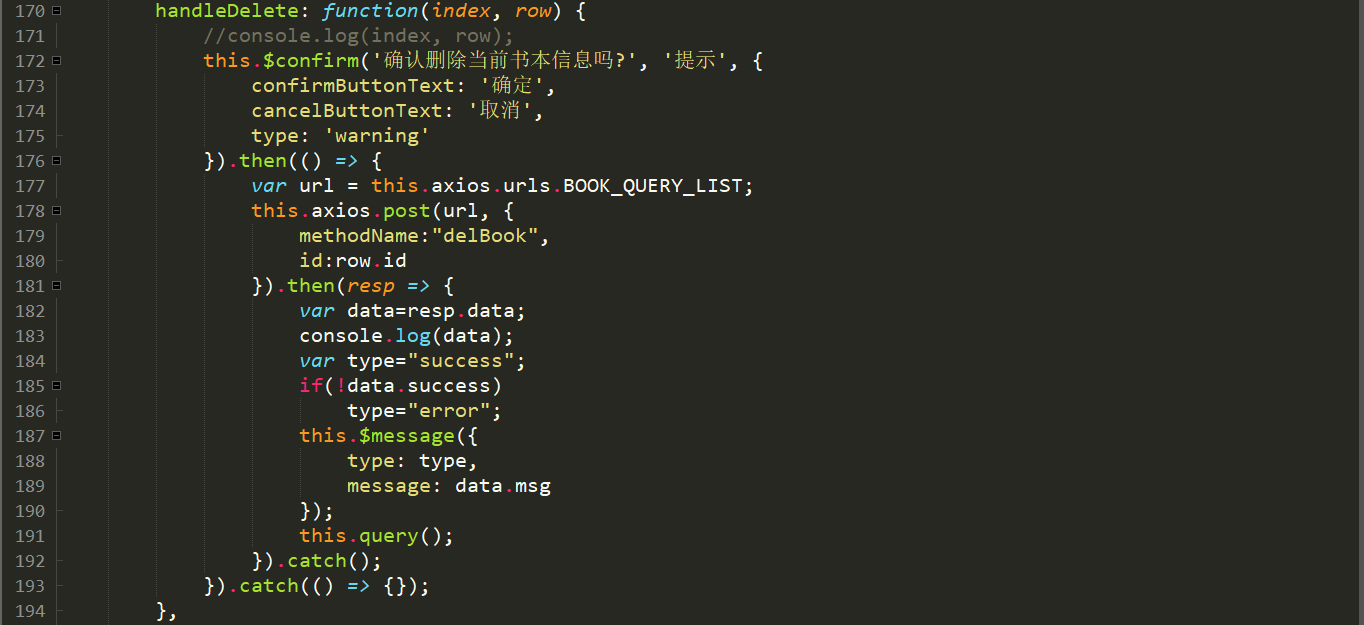


### 新增&修改

这里将新增和修改功能放置到一个方法中，通过dialog弹出框的标题title属性来区分。



### 删除



# 第十一天：Vuex

1. 课前准备
2. Vue中各组件之间传值
3. 什么是Vuex
4. Vuex使用步骤
5. Vuex核心概念
6. Vuex的常用辅助函数

## 课前准备

1. readme.txt
2. 资料

## Vue中各组件之间传值

1. **父子组件**

父组件-->子组件，通过子组件的自定义属性：**props**

子组件-->父组件，通过自定义事件：**this.$emit**('事件名',参数1,参数2,...);

1. **非父子组件或父子组件**

通过数据总数Bus，this.$root.$emit('事件名',参数1,参数2,...)

1. **非父子组件或父子组件**

更好的方式是在vue中使用vuex

## 什么是Vuex

官方解释：Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。可以想象为一个“前端数据库”（数据仓库），让其在各个页面上实现数据的共享包括状态，并且可操作。

Vuex分成五个部分：

1. **State：**单一状态树；
2. **Getters：**状态获取；
3. **Mutations：**触发同步事件；
4. **Actions：**提交mutation，可以包含异步操作；
5. **Module：**将vuex进行分模块；

## Vuex使用步骤

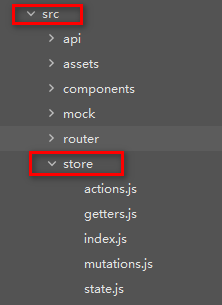
### 安装

通过命令：**npm install vuex -S**

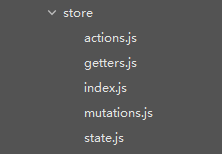
### 创建store模块

创建store模块，分别维护state/actions/mutations/getters。

1. **在项目的src目录下创建store目录（store模块）**



1. **然后在store目录下分别创建index.js/state.js/actions.js/mutations.js/getter.js**



1. **在state.js/actions.js/mutation.js/getter.js，添加export default {}**



### 创建vuex的store实例

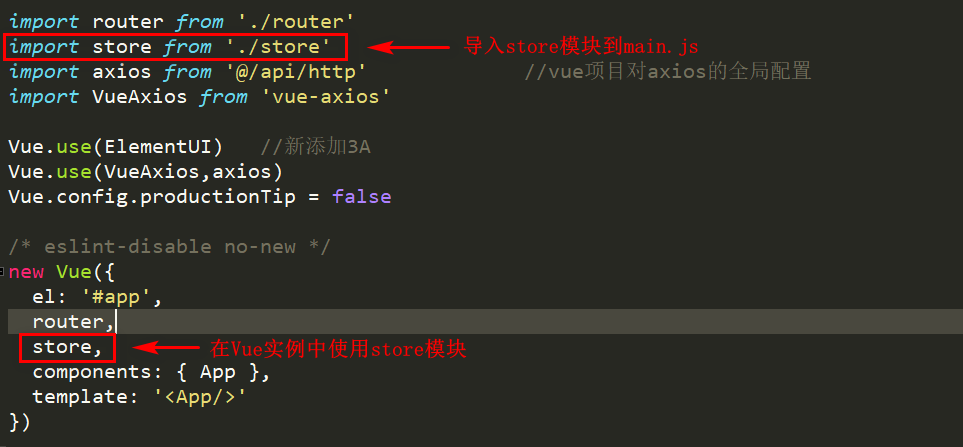
在store/index.js文件中新建vuex的store实例，并注册上面引入的各大模块。

const store = new Vuex.Store({state,getters,actions,mutations})



### 导入并使用store实例

在main.js中导入并使用store实例。



## Vuex核心概念

创建Page1、Page2组件，并将组件导入到router/index.js中，配置路由；再将Page1、Page2组件加入到左侧菜单。

### Store

每一个Vuex应用的核心就是store(仓库),store基本上就是一个容器，它包含着你的应用中大部分的状态 (state)。

const store = new Vuex.Store({

state, // 共同维护的一个状态，state里面可以是很多个全局状态

getters, // 获取数据并渲染

actions, // 数据的异步操作

mutations // 处理数据的唯一途径，state的改变或赋值只能在这里

})

### State

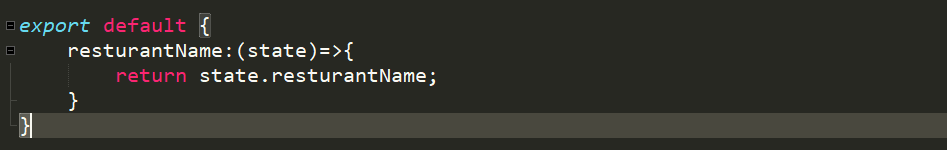
state是保存数据的容器。状态，即要全局读写的数据。



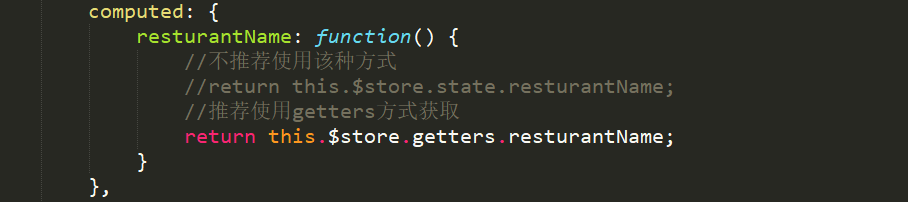
### Getters

Gettters获取数据并渲染。getters将state中定义的值暴露在this.$store.getters对象中，我们可以通过如下代码访问：**this.$store.getters.resturantName**；state状态存储是响应式的，从store实例中读取状态最简单的方法就是在计算属性中返回某个状态。

**getters.js定义：**



**在组件中通过计算属性方法获取状态信息：**



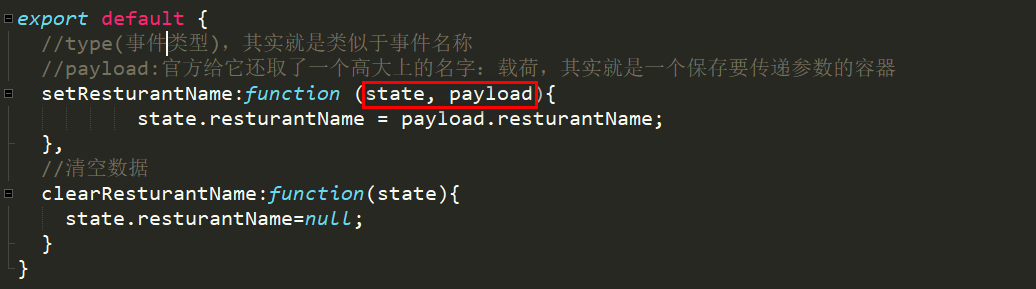
**2种获取数据方式：**

1. **this.$store.state.resturantName;(不推荐)**
2. **this.$store.getters.resturantName;(推荐)**

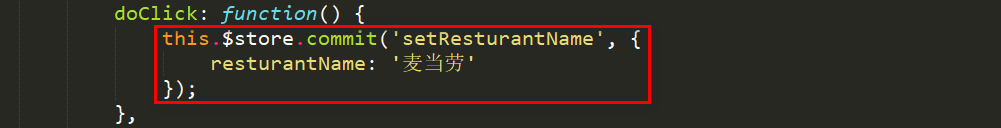
### Mutations(同步)

Mutations 处理数据的唯一途径，state的改变或赋值只能在这里。

**mutations.js定义：**



**在页面中定义按钮点击事件来改变state中的数据状态信息：**



**注：mutation中的方法调用方式语法：this.$store.commit(‘事件类型’,载荷)。**

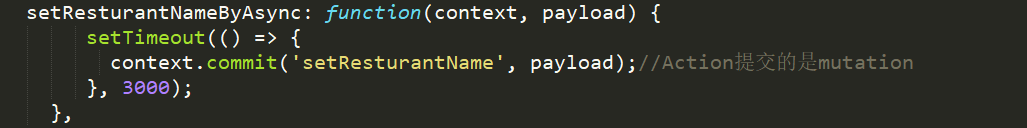
### Actions(异步)

Actions数据的异步(async)操作。Action类似于mutation，不同在于：

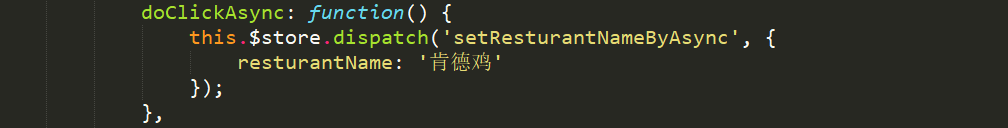
1. Action提交的是mutation，而不是直接变更状态
2. Action可以包含任意异步操作
3. Action的回调函数接收一个 context 上下文参数。注意：这个参数可不一般，它与 store 实例有着相同的方法和属性但是他们并不是同一个实例，context 包含：1. state、2. rootState、3. getters、4. mutations、5. actions 五个属性，所以在这里可以使用 context.commit 来提交一个 mutation，或者通过 context.state 和 context.getters 来获取 state 和 getters。

#### Actions之静态数据处理

**actions.js定义：**



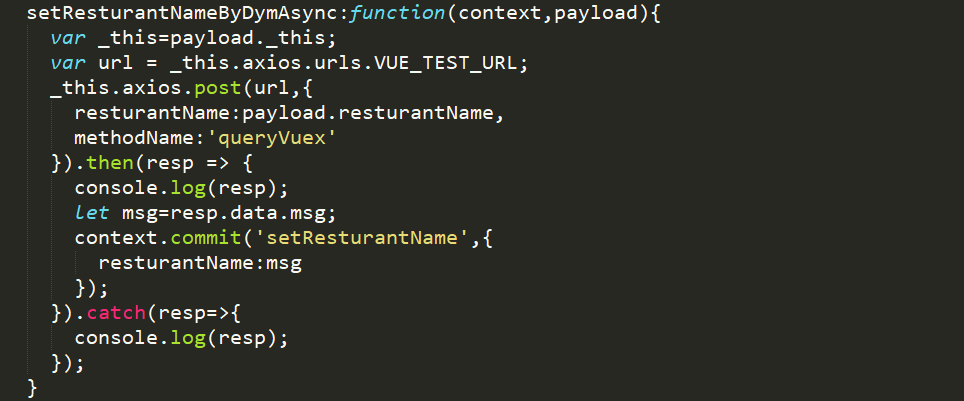
**在页面组件中通过按钮点击事件获取actions的静态数据：**



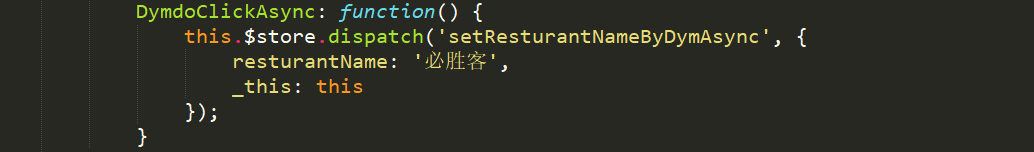
**注：actions中方法的调用方式语法：this.$store.dispath(type,payload);**

#### Actions之动态数据处理

**actions.js定义：**



**在页面组件中通过按钮点击事件获取actions的动态数据：（注：将Vue实例的当前this对象通过载荷的方式传入到actions中）**



**后台数据接口定义：**



## Vuex的常用辅助函数

mapState/mapGetters/mapMutations/mapActions，通过以上这些方式可简化操作。



# 第十二天：JWT

1. 课前准备
2. JWT
3. JWT验证过程

## 课前准备

1. JwtDemo.java、JwtUtils.java、JwtFilter.java
2. readme.txt

## JWT

### 什么是JWT

**JSON Web Token (JWT)**，它是目前最流行的跨域身份验证解决方案。

### 为什么使用JWT

JWT的精髓在于：“去中心化”，数据是保存在客户端的。直白的说，就是将以前保存在服务端(Session)的数据，通过JWT的方式保存到客户端；客户端每次请求的时候都会将JWT的token令牌带到服务端进行验证。

### JWT的工作原理

客户端提交用户登录身份验证请求，在身份验证通过之后，将生成一个JSON对象并将其发送回用户；之后，每当用户与服务器通信时，客户在请求中回发JSON对象，服务端解析回发的JSON对象并判断JWT中的token令牌是否超过有效时间，如果没有超过，则延迟token令牌失效时间；如果超过了，直接提示token令牌超过即可。

**注：为了防止用户篡改数据，服务器将在生成对象时添加签名，并对发回的数据进行验证。**

### JWT组成

一个JWT实际上就是一个字符串，它由三部分组成：头部(Header)、载荷(Payload)与签名(signature)。如下图所示：



JWT实际结构：

eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ7fSIsImlzcyI6InpraW5nIiwiZXhwIjoxNTYyODUwMjM3LCJpYXQiOjE1NjI4NDg0MzcsImp0aSI6ImM5OWEyMzRmMDc4NzQyZWE4YjlmYThlYmYzY2VhNjBlIiwidXNlcm5hbWUiOiJ6c3MifQ.WUfqhFTeGzUZCpCfz5eeEpBXBZ8-lYg1htp-t7wD3I4

它是一个很长的字符串，中间用点（.）分隔成三个部分。注意，JWT 内部是没有换行的，这里只是为了便于展示，将它写成了几行。写成一行，就是下面的样子：Header.Payload.Signature

#### Header

**{"typ":"JWT","alg":"HS256"}**这个json中的typ属性，用来标识整个token字符串是一个JWT字符串；它的alg属性，用来说明这个JWT签发的时候所使用的签名和摘要算法。typ跟alg属性的全称其实是type跟algorithm，分别是类型跟算法的意思。之所以都用三个字母来表示，也是基于JWT最终字串大小的考虑，

同时也是跟JWT这个名称保持一致，这样就都是三个字符了。typ跟alg是JWT中标准中规定的属性名称。

#### Payload

payload用来承载要传递的数据，它的json结构实际上是对JWT要传递的数据的一组声明，这些声明被JWT标准称为claims，它的一个“属性值对”其实就是一个claim(要求)，每一个claim的都代表特定的含义和作用。

根据JWT的标准，这些claims可以分为以下三种类型：

1. **Reserved claims(保留)**

**Reserved claims**属于JWT标准里面规定的一些claim。JWT标准里面定义好的claim有：

iss(Issuser)：代表这个JWT的签发主体；

sub(Subject)：代表这个JWT的主体，即它的所有人；

aud(Audience)：代表这个JWT的接收对象；

exp(Expiration time)：是一个时间戳，代表这个JWT的过期时间；

nbf(Not Before)：是一个时间戳，代表这个JWT生效的开始时间，意味着在这个时间之前验证JWT是会失败的；

iat(Issued at)：是一个时间戳，代表这个JWT的签发时间；

jti(JWT ID)：是JWT的唯一标识。

1. **Public claims（略，不重要）**
2. **Private claims(私有)**

**Private claims**这个指的就是自定义的claim。用于存放自定义的claims，比如：用户个人信息、角色权限信息等等。

#### Signature

签名是把header和payload对应的json结构进行base64url编码之后得到的两个串用英文句点号拼接起来，然后根据header里面alg指定的签名算法生成出来的。算法不同，签名结果不同。以alg: HS256为例来说明前面的签名如何来得到。按照前面alg可用值的说明，HS256其实包含的是两种算法：HMAC算法和SHA256算法，前者用于生成摘要，后者用于对摘要进行数字签名。这两个算法也可以用HMACSHA256来统称。

## JWT验证过程

### JwtDemo测试JWT

在测试JWT中请注意两个异常信息：**ExpiredJwtException（JWT过期异常）**和**SignatureException（签名认证失败异常）**

#### JWT测试之生成JWT

|  |
| --- |
| //定义私有声明  Map<String, Object> claims = **new** HashMap<String, Object>();  claims.put("username", "zss");  claims.put("age", 18);  //根据私有声明和JWT的token有效时间生成JWT  String jwt = JwtUtils.*createJwt*(claims, JwtUtils.***JWT\_WEB\_TTL***);  //获取JWT中的声明信息，包括：私有声明和标准声明  Claims parseJwt = JwtUtils.*parseJwt*(jwt);  **for** (Map.Entry<String, Object> entry : parseJwt.entrySet()) {  System.***out***.println(entry.getKey() + "=" + entry.getValue());  }  //获取token令牌的签发时间  Date d1 = parseJwt.getIssuedAt();  //获取token令牌的过期时间  Date d2 = parseJwt.getExpiration();  System.***out***.println("令牌签发时间：" + sdf.format(d1));  System.***out***.println("令牌过期时间：" + sdf.format(d2)); |

#### JWT测试之解析旧的JWT

|  |
| --- |
| //注：解析旧的JWT。如果超过JWT本身定义的过期时间，则直接抛出ExpiredJwtException  String oldJwt = "eyJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1NjM2NzY0NDQsImlhdCI6MTU2MzY3NDY0NCwiYWdlIjoxOCwianRpIjoiMzE5MmYxOTg4NzFkNGVkZWIyMzU0MmY3NWVhMWI5NDciLCJ1c2VybmFtZSI6InpzcyJ92.4VnkdvGNNe8U1EiKaLz7h6bOJkVGSGtfqojcP\_y-0Cc";  Claims parseJwt = JwtUtils.*parseJwt*(oldJwt);  **for** (Map.Entry<String, Object> entry : parseJwt.entrySet()) {  System.***out***.println(entry.getKey() + "=" + entry.getValue());  }  Date d1 = parseJwt.getIssuedAt();  Date d2 = parseJwt.getExpiration();  System.***out***.println("令牌签发时间：" + sdf.format(d1));  System.***out***.println("令牌过期时间：" + sdf.format(d2)); |

#### JWT测试之复制JWT

|  |
| --- |
| //注：模拟Session功能（Tomcat默认session过期时间：30分钟），  // 在有访问的情况下，Session自动延长过期时间。  // 当有访问的情况下，自动延长jwt中的token令牌过期时间，在原有时间上+30秒（测试环境，真实情况下+30分钟）  String oldJwt = "eyJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1NjM2NzUzMDAsImlhdCI6MTU2MzY3MzUwMCwiYWdlIjoxOCwianRpIjoiYjY5MmE5ZjZkMGZhNDMyN2I1YWY2NTI5OGMzMjQ5MTYiLCJ1c2VybmFtZSI6InpzcyJ9.asHGMPFKURMLnooK29abAuKEdLOHBycAuTovNuGQio0";  String jwt = JwtUtils.*copyJwt*(oldJwt, JwtUtils.***JWT\_WEB\_TTL***);  Claims parseJwt = JwtUtils.*parseJwt*(jwt);  **for** (Map.Entry<String, Object> entry : parseJwt.entrySet()) {  System.***out***.println(entry.getKey() + "=" + entry.getValue());  }  Date d1 = parseJwt.getIssuedAt();  Date d2 = parseJwt.getExpiration();  System.***out***.println("令牌签发时间：" + sdf.format(d1));  System.***out***.println("令牌过期时间：" + sdf.format(d2)); |

#### JWT测试之测试JWT的有效时间

|  |
| --- |
| Map<String, Object> claims = **new** HashMap<String, Object>();  claims.put("username", "zss");  String jwt = JwtUtils.*createJwt*(claims, 3 \* 1000L);  System.***out***.println(jwt);  Claims parseJwt = JwtUtils.*parseJwt*(jwt);  Date d1 = parseJwt.getIssuedAt();  Date d2 = parseJwt.getExpiration();  System.***out***.println("令牌签发时间：" + sdf.format(d1));  System.***out***.println("令牌过期时间：" + sdf.format(d2)); |

#### JWT测试之token令牌超时验证

|  |
| --- |
| // 三秒后再解析上面过期时间只有三秒的令牌，因为过期则会报错io.jsonwebtoken.ExpiredJwtException  String oldJwt = "eyJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1NjI4NTMzMzAsImlhdCI6MTU2Mjg1MzMyNywidXNlcm5hbWUiOiJ6c3MifQ.e098Vj9KBlZfC12QSDhI5lUGRLbNwb27lrYYSL6JwrQ";  Claims parseJwt = JwtUtils.*parseJwt*(oldJwt);  // 过期后解析就报错了，下面代码根本不会执行  Date d1 = parseJwt.getIssuedAt();  Date d2 = parseJwt.getExpiration();  System.***out***.println("令牌签发时间：" + sdf.format(d1));  System.***out***.println("令牌过期时间：" + sdf.format(d2)); |

### Ssh2+Vue+ElementUI实现JWT验证

#### 配置JWTFilter过滤器

导入JwtFilter.java到utils包下，并注意配置顺序问题：**CorsFilter -> JWTFilter -> Struts2核心过滤器**

|  |
| --- |
| <!-- JwtFilter -->  <filter>  <filter-name>jwtFilter</filter-name>  <filter-class>com.zking.ssh.base.jwt.JwtFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>jwtFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

#### 在登录成功后生成JWT

在登录成功后，使用JwtUtil工具类生成JWT，并将JWT包裹设置到响应头（response）中带到前端页面。在这里可以将用户对应用户基本信息和用户角色信息保存到私有claim中。

|  |
| --- |
| **if**(user.getPassword().equals("123")&&user.getUsername().equals("admin")) {  //私有要求claim  Map<String,Object> json=**new** HashMap<String,Object>();  json.put("username", user.getUsername());  json.put("password", user.getPassword());  //生成JWT，并设置到response响应头中  String jwt=JwtUtils.*createJwt*(json, JwtUtils.***JWT\_WEB\_TTL***);  response.setHeader(JwtUtils.***JWT\_HEADER\_KEY***, jwt);  **super**.toJSONMessage("用户登录成功", **true**);  }**else** {  **super**.toJSONMessage("用户名或密码错误", **false**);  } |

#### 配置CorsFilter过滤器

浏览器只能访问默认的响应头，例如：Cache-Control、Content-Language、Content-Type、Expires、Last-Modified、Pragma等，如果想让浏览器能访问到其他的响应头的话，需要在服务器上设置。**在CorsFilter跨域过滤器的doFilter方法中配置其他响应头的支持。**

|  |
| --- |
| //允许客户端发一个新的请求头jwt  httpResponse.setHeader("Access-Control-Allow-Headers","Origin,X-Requested-With, Content-Type, Accept, jwt");  //允许客户端处理一个新的响应头jwt  httpResponse.setHeader("Access-Control-Expose-Headers", "jwt"); |

#### 配置vuex定义JWT存储位置

使用vuex保存后端服务器提交过来的JWT，分别在state、getters、mutation中定义信息。

|  |
| --- |
| **==> state.js**  **export** **default** {  jwt:**null**  }  **==> getters.js**  **export** **default** {  jwt:(state)=>{  **return** state.jwt;  }  }  **==> mutation.js**  **export** **default** {  //type(时间类型)，其实就是类似于事件名称  //payload:官方给它还取了一个高大上的名字：载荷，其实就是一个保存要传递参数的容器  setJwt:**function** (state, payload){  state.jwt = payload.jwt;  }  } |

#### 修改main.js

|  |
| --- |
| **window.vm =** **new** Vue({  el: '#app',  router,  store,  components: { App },  template: '<App/>'  }) |

#### 在vue中配置axios的请求响应头

在vue的http.js中配置axios的请求响应拦截器，用于存取JWT，并将JWT信息保存到Vuex中。

|  |
| --- |
| // 请求拦截器  axios.interceptors.request.use(**function**(config) {  **let** jwt=window.vm.$store.getters.jwt;  //console.log(jwt);  **if**(jwt)  config.headers['jwt']=jwt;  **return** config;  }, **function**(error) {  **return** Promise.reject(error);  });  // 响应拦截器  axios.interceptors.response.use(**function**(response) {  **let** jwt=response.headers['jwt'];  **if**(jwt)  window.vm.$store.commit('setJwt',{jwt:jwt});  **return** response;  }, **function**(error) {  **return** Promise.reject(error);  }); |

#### 使用vuex-along解决刷新state初始化问题

跳转到项目目录，安装vuex-along，使用命令：**npm i vuex-along -S**。修改store/index.js，将vuex中定义的jwt保存到vuex-along中的localeStorage中。从而解决掉刷新浏览器后state中的数据恢复到初始状态的问题。



|  |
| --- |
| **import** Vue from 'vue'  **import** Vuex from 'vuex'  **import** vuexAlong from 'vuex-along'  Vue.use(Vuex)  **import** state from '@/store/state'  **import** actions from '@/store/actions'  **import** getters from '@/store/getters'  **import** mutations from '@/store/mutations'  //创建Vuex的store实例  **const** store=**new** Vuex.Store({  state,  getters,  actions,  mutations,  **plugins:[vuexAlong({ //使用vuex-Along插件用于缓存数据**  **local:{**  **list:['jwt']**  **}**  **})]**  })  **export** **default** store |

# 第十三天：动态Tab+Dialog弹出框优化

## 课前准备

1. 搭建好的vue脚手架（elementui、vuex、axios、qs、vue-axios、vue-along）
2. elementUI（NavMenu导航菜单、Tabs标签页）

## 实现思路

1. 将打开的tab页签保存到vuex的一个栈里(openTabs:[])，而tabs显示遍历openTabs即可；
2. 登录成功，进入Main主页面，将首页推入栈，并设置为激活状态，首页tab页签不可关闭；
3. 点击左侧菜单，往栈里(openTabs)中存入菜单标题和URL路径，在tabs显示遍历openTabs；
4. tabs移除，@tab-remove="removeTab"方法，根据点击的tab标题移除openTabs中的已存在的tab；
5. 将openTabs和激活状态使用vuex-along缓存起来，点击退出系统时清空vuex中的缓存数据；

## 动态Tab

### Vuex状态管理

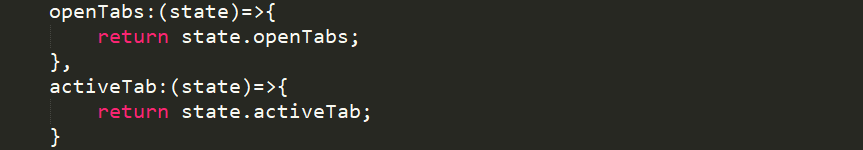
#### 定义存储对象(state.js)

将所有打开的tab选项卡（**openTabs**）和当前tab选项卡的激活状态（**activeTab**）保存到state.js中。



#### 定义对象获取方法(getters.js)

分别定义openTabs和activeTab对象的获取方法。



#### 定义对象设置方法(mutations.js)

##### 添加Tab

添加左侧菜单时，往openTabs中添加一个Tab选项卡。如果openTabs中已经存在，则直接切换到已打开的Tab选项卡并设置为激活状态；如果openTabs中不存在，则直接添加。

**注：因为在退出系统时会清空vuex中的state信息。在添加Tab时，请先判断openTabs是否为空，所以如果为空，则通过new Array()的方法新建一个。**

|  |
| --- |
| *//添加Tab* addTab:**function**(state,data){  let flag=**true**;  *//退出系统后，清空vuex中的state信息，判断openTabs是否为空* **if**(**null**!=state.openTabs){  *//判断openTabs中是否已经存在了tab，如果存在则激活选中，不存在则添加新tab* **for**(let tab of state.openTabs){  **if**(tab.name==data.name){  flag=**false**;  **break**;  }  }  }**else** state.openTabs=**new** Array();  *//如果不存在，则添加到openTabs中；如果存在，则设置当前tab为激活状态* **if**(flag)  state.openTabs.push(data);  **else  this**.commit(**'setTabActive'**,data.name);  } |

##### 删除Tab

先清空已打开的所有Tabs选项卡，再将剩余的tab页签循环添加到openTabs中。

|  |
| --- |
| *//删除Tab* delTab:**function**(state,data){  *//先清空，在将剩余的tab页签加入* state.openTabs=[];  **for**(let tab of data){  state.openTabs.push(tab);  } } |

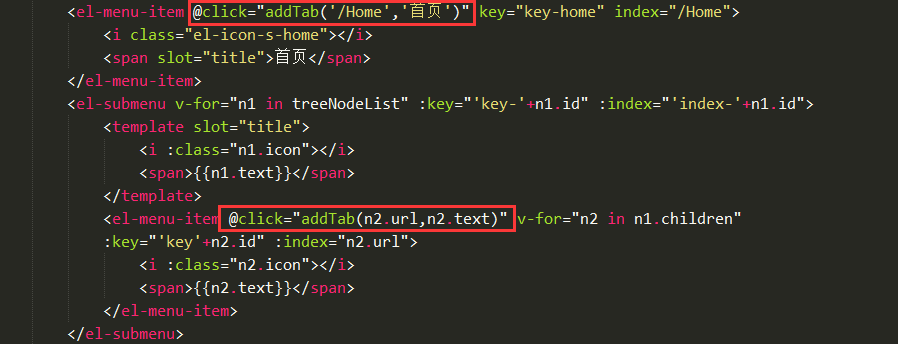
##### 设置Tab激活选中

根据路由路径设置Tab选项卡激活选中。

|  |
| --- |
| *//设置tab激活选中* setTabActive:**function**(state,path){  state.activeTab=path; } |

### 左侧菜单

#### Template定义



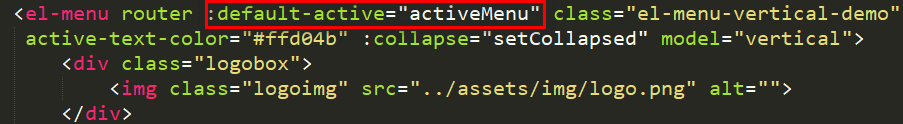
#### 菜单点击事件

点击左侧菜单项触发Tab选项卡的添加事件，将需要被打开的组件信息添加到vuex的openTabs中。并设置当前Tab选项卡为激活选中状态。

|  |
| --- |
| addTab:**function**(url,text){  //调用vuex中的addTab方法，将点击的菜单对应的组件信息添加到已打开的openTabs中  **this**.$store.commit(**'addTab'**,{  title:text, //Tab页签标题  name:url //组件的路由  });  //设置当前Tab面板为选中状态  **this**.$store.commit(**'setTabActive'**,url);  } |

#### 菜单激活状态

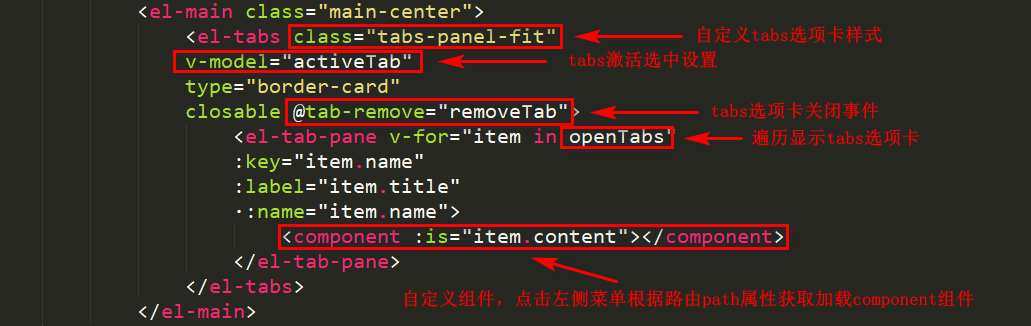
在点击tab选项卡时切换对应选中左侧菜单栏中的菜单项。根据保存在vuex中的tab激活状态来动态设置菜单选中状态。



|  |
| --- |
| computed: {  activeMenu:function(){  return **this**.$store.getters.activeTab;  }  } |

### 内容区域

#### Template定义



重点说明：

1. **class="tabs-panel-fit"**：设置tabs内容面板100%填充父容器；
2. **v-model="activeTab"**：设置tabs默认激活选中具体的页签；
3. **@tab-remove="removeTab"**：定义tabs选项卡的关闭事件；
4. **el-tab-pane v-for="item in openTabs"**：循环遍历显示tabs选项卡；
5. **<component :is="item.content">**：自定义组件模板，点击左侧菜单根据路由的path属性动态获取加载component组件（**核心部分**）；

#### 样式定义

|  |
| --- |
| **/\* 设置tabs内容面板100%填充父容器 \*/**  .tabs-panel-fit {  margin: 0;  padding: 0;  width: 100%;  height: 100%;  object-fit: fill;  }  **/\* 设置tabs内容面板的内边距为0 \*/**  .el-tabs--border-card>.el-tabs\_\_content {  padding: 0;  }  **/\* 设置tab的表框效果（重要） \*/**  .el-tabs.el-tabs--border-card {  box-shadow: none;  border-bottom: none;  border: 0;  }  **/\*设置选项卡中第一个首页选项卡不能关闭\*/**  .el-tabs>.el-tabs\_\_header .el-tabs\_\_item:first-child>span {  display: none;  } |

#### Mounted初始化

登录成功后，进入主界面。调用Vue的钩子函数mounted在页面组件加载完毕后将首页组件添加到openTabs已打开的tab中，实现提交预加载，等待遍历绑定tabs选项卡。

|  |
| --- |
| mounted() {  if (**this**.$route.path == '/Main') {  **this**.$store.commit(**'addTab'**, {  title: '首页',  name: '/Home' **//router路由path属性！！！重要！！重要！！重要！！**  });  **this**.$store.commit(**'setTabActive'**, '/Home');  }  } |

#### Computed计算属性

使用vue的computed计算属性方式，动态绑定tabs选项卡和设置tabs选项卡的激活选中状态。

##### 动态绑定tabs

|  |
| --- |
| openTabs: **function**() {  **//获取所有已打开的tabs**  let openTabs = **this**.$store.getters.openTabs;  **//定义tabs**  let tabs = [];  **//判断已打开的tabs是否为空**  if (null == openTabs)  return tabs;  else {  for (let tab of openTabs) {  **//使用require动态加载component组件**  let component = resolve => require.ensure([], () => resolve(require('@/views' + tab.name + '.vue')));  tabs.push({  title: tab.title,  name: tab.name,  content: component  });  }  return tabs;  }  } |

##### 设置tabs激活选中状态

|  |
| --- |
| **//tabs激活选中状态的方法，并提供的get/set方法**  activeTab: {  get() {  return **this**.$store.getters.activeTab;  },  set(val) {  **this**.$store.commit(**'setTabActive'**, val);  }  } |

#### Tab删除

点击tab选项卡的关闭按钮触发关闭事件，并删除openTabs中已存在的tab选项卡信息。

|  |
| --- |
| removeTab: **function**(targetName) {  **//获取所有已打开的Tabs选项卡**  let tabs = **this**.$store.getters.openTabs;  **//获取当前激活的Tab选项卡**  let activeName = **this**.$store.getters.activeTab;  **//判断当前激活的Tab选项卡和当前被关闭的Tab选项卡是否相同**  if (activeName === targetName) {  **//遍历已打开的选项卡**  tabs.forEach((tab, index) => {  **//判断是否相同，并获取就近需要被激活选项卡**  if (tab.name === targetName) {  let nextTab = tabs[index + 1] || tabs[index - 1];  if (nextTab) {  activeName = nextTab.name;  }  }  });  }  **//激活选项卡**  **this**.$store.commit(**'setTabActive'**, activeName);  **//删除Tab**  **this**.$store.commit(**'delTab'**, tabs.filter(tab => tab.name !== targetName));  } |

## Dialog弹出框优化

|  |
| --- |
| **/\* elementUI dialog弹出框样式优化 \*/**  .el-dialog {  position: absolute;  top: 50%;  left: 50%;  margin: 0 !important;  transform: translate(-50%, -50%);  max-height: calc(100% - 30px);  max-width: calc(100% - 30px);  display: flex;  flex-direction: column;  /\* 圆角效果 \*/  border-radius: 0.4em;  }  .el-dialog>.el-dialog\_\_body {  overflow: auto;  }  .el-dialog\_\_header {  background-color: #F5F7FA;  border-bottom: 1px solid #ccc;  font-size: 14px;  font-weight: bold;  padding: 15px 20px 15px;  /\* 圆角效果 \*/  border-radius: 0.4em 0.4em 0em 0em;  }  .el-dialog\_\_body {  padding: 20px 15px;  }  .el-dialog\_\_footer {  background-color: #F5F7FA;  border-top: 1px solid #ccc;  padding: 15px;  /\* 圆角效果 \*/  border-radius: 0em 0em 0.4em 0.4em;  }  .el-form>.el-form-item:last-child {  margin-bottom: 0px;  } |