2018/7/22 前端/VUEJS/day1

**第一天**

第一天 01

1. 验证码机制

验证码是从后端生成的，随机生成的，后端永远认为前端可能会伪造验证码

1. 后端调用相关的绘图第三方类库，或是系统平台php.net.java 系统核心绘图类库进行验证图片的绘制
2. 绘制的那些随机数字字母都是后端预先定义好的。
3. 将绘制的图片url地址通过网络发送给客户端，然后，客户端可以使用img标签，去应用这个验证码的地址；
4. 后端在绘制完毕验证码之后，注意，随机选择生成的字母，不能丢弃，而是需要保存到session中
5. 当客户端输入验证码完毕后，会提交表单，后端服务器会拿到客户端提交过来的验证码，与服务器端的session中的验证码进行比较
6. 前端不一定要会数据库

后端一定要会数据库

第一天 02

Vue课程介绍

打包工具webpack gulp

项目驱动

第一天 03

Vuejs介绍

Vue.js是目前最火的一个前端框架，react是最流行的一个前端框架（react除了开发网站，还可以开发手机app，vue也是可以进行手机app开发，需要借助于weex）

Vue.js是前端的主流框架之一，和angular.js、react.js并称为前端三大主流框架

Vue.js是一套构建用户界面的框架，只关注视图层，它不仅易于上手，还便于与第三方库或既有项目整合

前端的主要工作是负责mvc中的v这一层，主要工作就是和界面打交道，来制作前端页面效果。

1. 04

为什么要学习前端框架

提高开发效率。

提高开发效率历程：

原生JS（浏览器兼容性）🡪JQUERY之类的类库（频繁操作DOM）->前端模板引擎（操作DOM）->angular.js/vue.js（帮助我们减少不必要的DOM操作，提高渲染效率，双向数据绑定的概念【通过框架提供的指令，我们只需要关注数据的业务逻辑，不再关心DOM是如何渲染的了】）。

在vue中，一个核心的概念就是让用户不再操作DOM元素，解放了用户的双手，让程序员可以更多时间去关注业务逻辑

第一天 05

框架和库的区别

框架：是一套完整的解决方案，对项目的侵入性比较大，项目如果需要更换框架，这需要重新架构整个项目。

Node中的express

库：插件，提高某一个小功能，对项目的侵入性较小，如果某个库无法完成某些需求，可以很容易切换到其他库实现需求

1. 从jquery切换到zepto
2. 从ejs切换到art-template

第一天 06

Node(后端)中的mvc与前端中的MVVM之间的区别

MVC是后端的分层开发概念

MVVM是前台视图层的概念，主要关注于视图层的分离，也就是说：MVVM吧前端视图层分为ile三部分Model View VM(viewmodel)



第一天 07

VUE.js基本代码和MVVM之间的对应关系

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <!--1.导入vue的包-->  <script src="./lib/vue.js"></script>  </head>  <body>  <!--将来new的vue实例会控制这个元素中的所有内容-->  <!--vue实例所要控制的这个元素区域，就是我们的v-->  <div id="app">  <p>{{msg}}</p>  </div>  <script>  //创建一个Vue的实例  //当我们导入包之后，在浏览器的内存中，就多了一个vue的构造函数  //注意，我们new出来的这个vm对象，就是我们mvvm中的vm调度者  var vm = new Vue({  el:'#app',//表示，当前我们new的这个vue实例，要控制页面上的那个区域  //这里的data就是mvvm中的m，专门用来保存每个页面的数据的  data:{//data：属性中，存放的是el中要用的数据  msg:"欢迎学习vue"//通过vue提供的指令很方便的把数据渲染到页面上，不需要操作dom元素了，前端的vue子类的框架不提前手动操作dom元素  }  })  </script>  </body>  </html> |

第一天 08

v-cloak

<!-- 使用v-cloak可以解决插值表达式闪烁的问题-->

v-text

<!--默认v-text是没有闪烁问题的-->

<!--v-text 会覆盖元素中与岸边的内容，但是插值表达式只会替换自己的占位符不会把整个内容清空-->

v-html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <style type="text/css">  [v-cloak}{  display: none}  </style>    </head>  <body>    <div id="app">  <!-- 使用v-cloak可以解决插值表达式闪烁的问题-->  <p v-cloak>+++++++{{msg}}----</p>  <h4 v-text="msg">+++++</h4>  <!--默认v-text是没有闪烁问题的-->  <!--v-text 会覆盖元素中与岸边的内容，但是插值表达式只会替换自己的占位符不会把整个内容清空-->  <div>{{msg2}}</div>  <div v-text="msg2"></div>  <div v-html="msg2">111</div>  </div>  <script src="./lib/vue.js"></script>  <script>  var vm = new Vue({  el:"#app",  data:{  msg:"123",  msg2:"<h1>我是一个h1</h1>"  }  })  </script>  </body>  </html> |

第一天 09

v-bind:

<!-- v-bind:是一个vue中提供的用于绑定属性的指令 -->

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <style type="text/css">  [v-cloak}{  display: none}  </style>    </head>  <body>    <div id="app">  <!-- 使用v-cloak可以解决插值表达式闪烁的问题-->  <p v-cloak>+++++++{{msg}}----</p>  <h4 v-text="msg">+++++</h4>  <!--默认v-text是没有闪烁问题的-->  <!--v-text 会覆盖元素中与岸边的内容，但是插值表达式只会替换自己的占位符不会把整个内容清空-->  <div>{{msg2}}</div>  <div v-text="msg2"></div>  <div v-html="msg2">111</div>  <!-- v-bind:是一个vue中提供的用于绑定属性的指令 -->  <!-- <input type="button" name="" value="按钮" v-bind:title="mytitle + '123'" > -->  <!-- 注意v-bind可以简写为：要绑定的属性 -->  <!-- v-bind中可以写合法的JS表达式 -->  <input type="button" name="" value="按钮" :title="mytitle + '123'" >  </div>  <script src="./lib/vue.js"></script>  <script>  var vm = new Vue({  el:"#app",  data:{  msg:"123",  msg2:"<h1>我是一个h1</h1>",  mytitle:"这是一个自己定义的title"  }  })  </script>  </body>  </html> |

第一天 10

v-on

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <style type="text/css">  [v-cloak}{  display: none}  </style>    </head>  <body>    <div id="app">  <!-- 使用v-cloak可以解决插值表达式闪烁的问题-->  <p v-cloak>+++++++{{msg}}----</p>  <h4 v-text="msg">+++++</h4>  <!--默认v-text是没有闪烁问题的-->  <!--v-text 会覆盖元素中与岸边的内容，但是插值表达式只会替换自己的占位符不会把整个内容清空-->  <div>{{msg2}}</div>  <div v-text="msg2"></div>  <div v-html="msg2">111</div>  <!-- v-bind:是一个vue中提供的用于绑定属性的指令 -->  <!-- <input type="button" name="" value="按钮" v-bind:title="mytitle + '123'" > -->  <!-- 注意v-bind可以简写为：要绑定的属性 -->  <!-- v-bind中可以写合法的JS表达式 -->  <!-- vue中提供了v-on事件绑定机制 -->  <!-- <input type="button" name="" value="按钮" :title="mytitle + '123'" v-on:click="alert()" > -->  <input type="button" value="按钮" v-on:click="show" name="">  </div>  <script src="./lib/vue.js"></script>  <script>  var vm = new Vue({  el:"#app",  data:{  msg:"123",  msg2:"<h1>我是一个h1</h1>",  mytitle:"这是一个自己定义的title",  },  methods:{//methods属性中定义了当前vue实例所有可用的方法  show:function(){  alert('hello')  }  }  })  </script>  </body>  </html> |

第一天 11

跑马等效果1

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <input type="button" value="跑起来" name="" @click="lang">  <input type="button" value="低调" name="">  <h4>{{msg}}</h4>  </div>  <script type="text/javascript">  //注意；在vm实例中如果想要获取data上的数据，或者想要调用methods中的方法，必须通过this.属性名 或 this.方法名来进行访问，这里的this就表示我们new处理的VM实例对象。  var vm=new Vue({  el:"#app",  data:{  msg:"勇往直前，别回头～～！"  },  methods:{  lang(){  //console.log(this.msg)  //获取到头的第一个字符  //()=>内部this指向外部this  setInterval(()=>{  var start = this.msg.substring(0,1)  //获取的到后面的所有字符  var end = this.msg.substring(1)  //更新拼接得到的新的字符串，并赋值给 this.msg  this.msg = end+start  //注意vm实例会监听自己身上的data中所有数据的改变，只要数据一发生变化就会自动把新的数据从data中同步到页面中去：好处：减少对dom的操作，只需要关心数据，不需要关心页面  },400)    }  }  })  //分析  //1.给【跑起来】按钮绑定一个点击事件 v-on  //2.在按钮的事件处理函数中写相关的业务逻辑代码：拿到msg字符串，然后调用字符串的substring来进行字符串的截取操作，把第一个字符串截取出来，放到最后一个位置即可  //3.为了实现点击一下按钮自动截取的功能需要把2放入一个定时器中去  </script>  </body>  </html> |

第一天 12

跑马灯效果2

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <input type="button" value="跑起来" name="" @click="lang">  <input type="button" value="低调" name="" @click="stop">  <h4>{{msg}}</h4>  </div>  <script type="text/javascript">  //注意；在vm实例中如果想要获取data上的数据，或者想要调用methods中的方法，必须通过this.属性名 或 this.方法名来进行访问，这里的this就表示我们new处理的VM实例对象。  var vm=new Vue({  el:"#app",  data:{  msg:"勇往直前，别回头～～！",  intervalInd:null//在data上定义定时器id  },  methods:{  lang(){  //console.log(this.msg)  //获取到头的第一个字符  //()=>内部this指向外部this  if(this.intervalInd!=null){  return  }  this.intervalInd=setInterval(()=>{  var start = this.msg.substring(0,1)  //获取的到后面的所有字符  var end = this.msg.substring(1)  //更新拼接得到的新的字符串，并赋值给 this.msg  this.msg = end+start  //注意vm实例会监听自己身上的data中所有数据的改变，只要数据一发生变化就会自动把新的数据从data中同步到页面中去：好处：减少对dom的操作，只需要关心数据，不需要关心页面  },400)    },  stop(){  clearInterval(this.intervalInd)  //每当清除了定时器需要把intervalId置为null  this.intervalInd=null;  }  }  })  //分析  //1.给【跑起来】按钮绑定一个点击事件 v-on  //2.在按钮的事件处理函数中写相关的业务逻辑代码：拿到msg字符串，然后调用字符串的substring来进行字符串的截取操作，把第一个字符串截取出来，放到最后一个位置即可  //3.为了实现点击一下按钮自动截取的功能需要把2放入一个定时器中去  </script>  </body>  </html> |

第一天 13

事件修饰符

Stop

Prevent

Capture

Self

Once

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <style type="text/css">  .inner{  height: 150px;  background-color:darkcyan;  }  .outer{  padding: 40px;  background-color: red  }  </style>  </head>  <body>  <div id="app">  <!-- 使用 .stop 阻止冒泡 -->  <!-- <div class="inner" @click="divhandler">  <input type="button" @click.stop="btnhandler" value="戳他" name="">  </div> -->  <!-- 使用 .prevent 阻止默认事件行为-->  <!-- <a href="http://www.baidu.com" @click.prevent="linkclick">有问题先百度</a> -->  <!-- 使用 .capture实现捕获触发事件的机制 -->  <!-- <div class="inner" @click.capture="divhandler">  <input type="button" @click.stop="btnhandler" value="戳他" name="">  </div> -->  <!-- 使用.self实现只有点击当前元素时候才会触发事件处理函数 -->  <!-- <div class="inner" @click.self="divhandler">  <input type="button" @click.stop="btnhandler" value="戳他" name="">  </div> -->  <!-- 使用once只触发一次事件处理函数 -->  <!-- <a href="http://www.baidu.com" @click.once.prevent="linkclick">有问题先百度</a> -->  <!-- 演示 .stop 和 .self 的区别 -->  <!-- <div class="outer" @click="div2handler">  <div class="inner" @click="divhandler">  <input type="button" @click.stop="btnhandler" value="戳他" name="">  </div>  </div> -->  <!-- .self 只会阻止自己身上的冒泡行为的触发，并不会阻止冒泡行为 -->  <!-- <div class="outer" @click="div2handler">  <div class="inner" @click.self="divhandler">  <input type="button" @click="btnhandler" value="戳他" name="">  </div>  </div> -->  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{},  methods:{  divhandler(){  console.log("触发了inner")  },  btnhandler(){  console.log("触发了btn")  },  linkclick(){  console.log("这是触发了链接")  },  div2handler(){  console.log(  "这是触发了outer")  }  }  })  </script>  </body>  </html>  <!-- 冒泡机制，从div最上层开始，一层一层触发 -->  <!-- 阻止事件冒泡，即只触发当前点击的这一个，它下面的不触发 -->  <!-- 捕获事件，从外往里触发事件，即从最下层触发 --> |

第一天 14

v-model

双向数据绑定

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <h4>{{msg}}</h4>  <!-- v-bind只能实现数据的单向绑定，只能从vm到v中去，无法实现数据的双向绑定 -->  <!-- <input type="text" v-bind:value="msg" name="" style="width: 100%"> -->  <!-- v-model 能够实现表单元素和model中数据的双向绑定-->  <!-- 注意v-model只能应用于表单元素中 -->  <!-- input (radio text address email) select checkbox textarea-->  <input type="text" v-model:value="msg" name="" style="width: 100%">  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  msg:"好学生",  },  methods:{}  })  </script>  </body>  </html> |

第一天 15

简易计算器案例

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <input type="text" v-model="n1" name="">  <select v-model="opt" >  <option value="+">+</option>  <option value="-">-</option>  <option value="\*">\*</option>  <option value="/">/</option>  </select>  <input type="text" name="" v-model="n2" >  <input type="button" value="=" name="" @click="calc">  <input type="text" v-model="result" name="">  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  n1:0,  n2:0,  result:0,  opt:'+',  },  methods:{  calc(){  /\*switch(this.opt){  case'+':  this.result=parseInt(this.n1) + parseInt(this.n2)  break;  case'-':  this.result=parseInt(this.n1) - parseInt(this.n2)  break;  case'\*':  this.result=parseInt(this.n1) \* parseInt(this.n2)  break;  case'/':  this.result=parseInt(this.n1) / parseInt(this.n2)  break;  }\*/  //正式开发尽量不要用  var codeStr = 'parseInt(this.n1)' +this.opt +'parseInt(this.n2)'  this.result= eval(codeStr)  }  }  })  </script>  </body>  </html> |

第一天 16

在vue中使用样式

使用class样式

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <style type="text/css">  .red{  color:red;  }  .thin{  font-weight: 200  }  .italic{  font-style: italic;  }  .active{  letter-spacing: 0.5em  }  </style>  </head>  <body>  <div id="app">  <!-- <h1 class="red thin">很大的h1</h1>  -->  <!-- 第一种使用方式，直接传递一个数组，注意：这里的class需要使用v-bind做数据绑定 -->  <!-- <h1 :class="['red','thin']">很大的h1</h1>  -->  <!-- 在数组中使用三目表达式-->  <!-- <h1 :class="['thin','italic',flag?'active':'']">很大的h1</h1> -->  <!-- 在数组中使用对象来代替三目表达式，提高代码的可读性 -->  <!-- <h1 :class="['thin','italic',{'active':flag}]">很大的h1</h1> -->  <!-- 在为class使用v-bingd绑定对象的时候对象的属性是类名，对象的属性可带引号也可不带引号所以这里没有带引号，属性的值是一个标识符 -->  <h1 :class="classObj">很大的h1</h1>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  flag:true,  classObj:{red:true,thin:true,italic:false,active:true}  },  methods:{}  })  </script>  </body>  </html> |

第一天 17

使用内联样式

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <!-- 对象就是无序键值对的集合 -->  <h1 :style="{color:'red','font-weight':'200'}">这是一个h1</h1>  <h1 :style="styleObj1">这是一个h1111</h1>  <h1 :style="[styleObj1,styleObj2]">这是一个h133</h1>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  styleObj1:{color:'red','font-weight':'200'},  styleObj2:{'font-style':'italic'}  },  methods:{}  })  </script>  </body>  </html> |

第一天 18

V-for

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <p>{{list[0]}}</p>  <p v-for="item in list">{{item}}</p>  <p v-for="(item,i) in list">{{i}}---------{{item}}</p>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  list:[1,2,3,4,5,6,7]  },  methods:{}  })  </script>  </body>  </html> |

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <p v-for="(user,i) in list">id:{{user.id}}----name:{{user.name}}----index:{{i}}</p>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  list:[  {id:1,name:'zs1'},  {id:2,name:'zs2'},  {id:3,name:'zs3'},  {id:4,name:'zs4'},  ]  },  methods:{}  })  </script>  </body>  </html> |

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <!-- 在遍历对象身上的键值对的时候，除了有val key 在第三个位置上还有一个索引 -->  <p v-for="(val,key,i) in user">  值是：{{val}}--键是：{{key}}--索引：{{i}}  </p>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  user:{  id:1,  name:'tuoni',  gender:'男'  }  },  methods:{}  })  </script>  </body>  </html> |

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <!-- in后面放过普通数组，对象数组，对象，还可以放数字 -->  <!-- 如果使用v-for 迭代数字，前面的count值从1开始 -->  <p v-for="count in 10">这是第{{count}}次循环</p>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{},  methods:{}  })  </script>  </body>  </html> |

第一天 19

2.2.0版本以后，当在组件中

|  |
| --- |
| <!DOCTYPE html>  <html>  <meta charset="utf-8">  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <label>  id:<input type="text" v-model="id" name="">  </label>  <label>  name:<input type="text" v-model="name" name="">  </label>  <input type="button" value="添加" @click="add" name="">  <!-- 注意v-for 循环时key属性只能使用number或者string -->  <!-- 注意：key在使用的时候，必须使用v-bind属性绑定的形式，指定key的值 -->  <!-- 在组件中使用v-for循环时，或者在一些特殊情况中，如果v-for有问题，必须在使用 v-for的同时，指定唯一的字符串或数字类型：key值-->  <p v-for="item in list" :key="item.id">  <input type="checkbox" name="">{{item.id}}---  {{item.name}}  </p>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  id:"",  name:"",  list:[  {id:1,name:"lishi"},  {id:2,name:"yinzheng"},  {id:3,name:"zhaogao"},  {id:4,name:"hangfei"}]  },  methods:{  add(){  this.list.unshift({id:this.id,name:this.name})  }  }  })  </script>  </body>  </html> |

第一天 20

v-if

v-show

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <!-- <input type="button" value="toggle" @click="toggle" name=""> -->  <!-- v-if的特点每次都会重新删除或者创建元素 -->  <!-- v-show每次不会重新dom的操作，只是切换了元素的display:none样式 -->  <!--v-if有较高的切换性能消耗 -->  <!--v-show有较高的初始渲染消耗 -->  <!-- 如果元素涉及到频繁的切换，最后不要使用v-if，而是推荐使用v-show-->  <!-- 如果元素可能永远可能不会被显示出来被用户看到，则推荐使用v-if-->  <input type="button" value="toggle" @click="flag=!flag" name="">  <h3 v-if="flag">v-id</h3>  <h3 v-show="flag">v-show</h3>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  flag:true,  },  methods:{  toggle(){  this.flag = ! this.flag  }  }  })  </script>  </body>  </html> |

第一天 21

总结

<!-- 1.mvc和mvvm的区别 -->

<!-- 2.vue基本代码的结构 注意不要给body加id="app" -->

<!-- 3.基本指令 插值表达式 v-cloak v-text v-html v-bind(:) v-on(@) v-model v-for v-if v-show -->

<!-- 4.事件修饰符 :stop :prevent :capture :self :once -->

<!-- 5.el 指定要控制的区域 data指定了控制区域内要用的数据 methods 虽然带s后缀，但是是个对象，这里可以自定义方法 -->

<!-- 6.在vm实例中，如果要访问data上的数据，或者要访问methods中的方法，必须带this -->

<!-- 7.v-for要会使用key属性，不接受对象，只接受number或string -->

<!-- 8.v-model只能应用于表单元素 -->

<!-- 9.在vue中绑定样式两种方式，v-bind:class v-bind:style-->

**第二天**

第二天1

反馈

React.ng.vue特点

React---移动app

第二天2

目录：

品牌管理案例

过滤器

自定义键盘修饰符

自定义指令

Vue生命周期

Vue数据请求

Vue动画

第二天3

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap.css">  </head>  <body>  <div id="app">  <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3>  </div>  <div class="panel-body form-inline">  <label>  id:  <input type="text"  class="form-control" v-model="id"  name="">  </label>  <label>  Name:  <input type="text" class="form-control"  v-model="name" name="">  </label>  <!-- 在vue中使用事件绑定机制为元素指定处理函数时如果加了（）,就可以传参了 -->  <input type="button" value="添加" class="btn btn-primary" @click="add()" name="">    </div>  </div>  <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <tr v-for="item in list" :key="item.id">  <td>{{item.id}}</td>  <td v-text="item.name"></td>  <td>{{item.ctime}}</td>  <td>删除</td>  </tr>  </tbody>  </table>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  id:"",  name:"",  list:[  {id:1,name:"奔驰",ctime:new Date()},  {id:2,name:"宝马",ctime:new Date()},  ]  },  methods:{  add(){  //console.log("ok")  //分析  //1.获取到id和name直接从data上面获取  //2.组织出一个对象  //3.把这个对象调用数组的相关发发添加到当前data上的list中  //4.注意：在vue中已经实现了数据的双向绑定，每当我们修改了data中的数据，vue会默认监听到数据的改动，自动把最新的数据应用到页面上  //5.当我们意识到上面的第四步时，已入门，更多的是进行vue中model数据的操作，同时，在操作model数据的时候，指定的业务逻辑操作  var car={id:this.id,name:this.name,ctime:new Date()}  this.list.push(car)  this.id=this.name=""  }  }  })  </script>  </body>  </html> |

第二天4

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap.css">  </head>  <body>  <div id="app">  <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3>  </div>  <div class="panel-body form-inline">  <label>  id:  <input type="text"  class="form-control" v-model="id"  name="">  </label>  <label>  Name:  <input type="text" class="form-control"  v-model="name" name="">  </label>  <!-- 在vue中使用事件绑定机制为元素指定处理函数时如果加了（）,就可以传参了 -->  <input type="button" value="添加" class="btn btn-primary" @click="add()" name="">    </div>  </div>  <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <tr v-for="item in list" :key="item.id">  <td>{{item.id}}</td>  <td v-text="item.name"></td>  <td>{{item.ctime}}</td>  <td @click.prevent="del(item.id)">删除</td>  </tr>  </tbody>  </table>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  id:"",  name:"",  list:[  {id:1,name:"奔驰",ctime:new Date()},  {id:2,name:"宝马",ctime:new Date()},  ]  },  methods:{  add(){  //console.log("ok")  //分析  //1.获取到id和name直接从data上面获取  //2.组织出一个对象  //3.把这个对象调用数组的相关发发添加到当前data上的list中  //4.注意：在vue中已经实现了数据的双向绑定，每当我们修改了data中的数据，vue会默认监听到数据的改动，自动把最新的数据应用到页面上  //5.当我们意识到上面的第四步时，已入门，更多的是进行vue中model数据的操作，同时，在操作model数据的时候，指定的业务逻辑操作  var car={id:this.id,name:this.name,ctime:new Date()}  this.list.push(car)  this.id=this.name=""  },  del(id){//根据id删除数据  //分析  //1.如何根据id找到要删除的索引  //2.直接调用数组的splice方法  //方法一  /\*this.list.some((item,i)=>{  if (item.id==id) {  this.list.splice(i,1)  //在数组的some方法中如果return true，就会立即终止这个数组的后续循环  return true;  }  })\*/  //方法二  var index=this.list.findIndex(item=>{  if (item.id==id) {  return true;  }  })  //console.log(index)  this.list.splice(index,1)  }  }  })  </script>  </body>  </html> |

第二天5

Vue调试工具

Vuedevtools

第二天6

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap.css">  </head>  <body>  <div id="app">  <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3>  </div>  <div class="panel-body form-inline">  <label>  id:  <input type="text"  class="form-control" v-model="id"  name="">  </label>  <label>  Name:  <input type="text" class="form-control"  v-model="name" name="">  </label>  <!-- 在vue中使用事件绑定机制为元素指定处理函数时如果加了（）,就可以传参了 -->  <input type="button" value="添加" class="btn btn-primary" @click="add()" name="">    <label>  搜索:  <input type="text" class="form-control"  v-model="keyword" name="">  </label>  </div>  </div>  <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <!-- 之前，v-for中的数据都是从data上的list中直接渲染过来的 -->  <!-- 现在，我们自定义一个search方法，同时我们把所有的关键字通过传参的形式，传递给了search方法 -->  <!-- 在search方法内部通过for循环，把所有符合搜索关键字的数据保存到一个新数组中 ，返回-->  <tr v-for="item in search(keyword)" :key="item.id">  <td>{{item.id}}</td>  <td v-text="item.name"></td>  <td>{{item.ctime}}</td>  <td @click.prevent="del(item.id)">删除</td>  </tr>  </tbody>  </table>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  id:"",  name:"",  keyword:"",//搜索关键字  list:[  {id:1,name:"奔驰",ctime:new Date()},  {id:2,name:"宝马",ctime:new Date()},  ]  },  methods:{  add(){  //console.log("ok")  //分析  //1.获取到id和name直接从data上面获取  //2.组织出一个对象  //3.把这个对象调用数组的相关发发添加到当前data上的list中  //4.注意：在vue中已经实现了数据的双向绑定，每当我们修改了data中的数据，vue会默认监听到数据的改动，自动把最新的数据应用到页面上  //5.当我们意识到上面的第四步时，已入门，更多的是进行vue中model数据的操作，同时，在操作model数据的时候，指定的业务逻辑操作  var car={id:this.id,name:this.name,ctime:new Date()}  this.list.push(car)  this.id=this.name=""  },  del(id){//根据id删除数据  //分析  //1.如何根据id找到要删除的索引  //2.直接调用数组的splice方法  //方法一  /\*this.list.some((item,i)=>{  if (item.id==id) {  this.list.splice(i,1)  //在数组的some方法中如果return true，就会立即终止这个数组的后续循环  return true;  }  })\*/  //方法二  var index=this.list.findIndex(item=>{  if (item.id==id) {  return true;  }  })  //console.log(index)  this.list.splice(index,1)  },  search(keyword){  //方法一  /\* var newList=[]  this.list.forEach(item=>{  if (item.name.indexOf(keyword)!=-1) {  newList.push(item)  }    })  return newList;\*/  //方法二  //注意：forEach some filter findIndex 数组的新方法  return this.list.filter(item=>{  //if (item.name.indexOf(keyword)!=-1)  //注意在es6中为字符串提供了一个新方法，叫string.prototype.includes('要包含的字符串')  //如果包含返回true ,否则为false  //  if (item.name.includes(keyword)) {  return item  }  })  }  }  })  </script>  </body>  </html> |

第二天7

过滤器

|  |
| --- |
| // 过滤器的定义语法  //Vue.filter('过滤器的名称',function(data){})  //过滤器中的function中的第一个参数已经被规定死了，永远都是过滤器管道符前面传递过来的数据  Vue.filter('nameope',function(data){}) |

多参数

多过滤器

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <P>{{msg | msgFormat('疯狂+1','123') | test}}</P>  </div>  <script type="text/javascript">  //定义一个全局的过滤器，名字msgFormat  //字符串的replace方法，第一个参数，除了可写一个字符串之外，还可以定义一个正则 g代表全局匹配  Vue.filter('msgFormat',function(msg,arg1,arg2){  return msg.replace(/单纯/g,arg1+arg2)  })  Vue.filter('test',function(msg){  return msg +"======="  })  var vm=new Vue({  el:"#app",  data:{  msg:'曾经，我也是一个单纯的少年,单纯的我，傻傻的问，谁是世界上最单纯的男人'  },  methods:{}  })    </script>  </body>  </html> |

第二天8

全局过滤器

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap.css">  </head>  <body>  <div id="app">  <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3>  </div>  <div class="panel-body form-inline">  <label>  id:  <input type="text"  class="form-control" v-model="id"  name="">  </label>  <label>  Name:  <input type="text" class="form-control"  v-model="name" name="">  </label>  <!-- 在vue中使用事件绑定机制为元素指定处理函数时如果加了（）,就可以传参了 -->  <input type="button" value="添加" class="btn btn-primary" @click="add()" name="">    <label>  搜索:  <input type="text" class="form-control"  v-model="keyword" name="">  </label>  </div>  </div>  <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <!-- 之前，v-for中的数据都是从data上的list中直接渲染过来的 -->  <!-- 现在，我们自定义一个search方法，同时我们把所有的关键字通过传参的形式，传递给了search方法 -->  <!-- 在search方法内部通过for循环，把所有符合搜索关键字的数据保存到一个新数组中 ，返回-->  <tr v-for="item in search(keyword)" :key="item.id">  <td>{{item.id}}</td>  <td v-text="item.name"></td>  <td>{{item.ctime | dateFormat }}</td>  <td @click.prevent="del(item.id)">删除</td>  </tr>  </tbody>  </table>  </div>  <script type="text/javascript">  //全局的过滤器，进行时间的格式化  Vue.filter('dateFormat',function(datestr,pattern=''){  //根据给定的字符串得到特定的时间  var dt=new Date(datestr)  //yyyy-mm-dd  var y=dt.getFullYear();  var m=dt.getMonth()  var d=dt.getDate();  //return y + '-' + m + '-' + d  //模板字符串  //tab上面的键点两下，中间写  //·yyyy-mm-dd·    //两种方式如果不传参 1 pattern为true 2.形参pattern=‘’  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss}`  }  })  var vm=new Vue({  el:"#app",  data:{  id:"",  name:"",  keyword:"",//搜索关键字  list:[  {id:1,name:"奔驰",ctime:new Date()},  {id:2,name:"宝马",ctime:new Date()},  ]  },  methods:{  add(){  //console.log("ok")  //分析  //1.获取到id和name直接从data上面获取  //2.组织出一个对象  //3.把这个对象调用数组的相关发发添加到当前data上的list中  //4.注意：在vue中已经实现了数据的双向绑定，每当我们修改了data中的数据，vue会默认监听到数据的改动，自动把最新的数据应用到页面上  //5.当我们意识到上面的第四步时，已入门，更多的是进行vue中model数据的操作，同时，在操作model数据的时候，指定的业务逻辑操作  var car={id:this.id,name:this.name,ctime:new Date()}  this.list.push(car)  this.id=this.name=""  },  del(id){//根据id删除数据  //分析  //1.如何根据id找到要删除的索引  //2.直接调用数组的splice方法  //方法一  /\*this.list.some((item,i)=>{  if (item.id==id) {  this.list.splice(i,1)  //在数组的some方法中如果return true，就会立即终止这个数组的后续循环  return true;  }  })\*/  //方法二  var index=this.list.findIndex(item=>{  if (item.id==id) {  return true;  }  })  //console.log(index)  this.list.splice(index,1)  },  search(keyword){  //方法一  /\* var newList=[]  this.list.forEach(item=>{  if (item.name.indexOf(keyword)!=-1) {  newList.push(item)  }  })  return newList;\*/  //方法二  //注意：forEach some filter findIndex 数组的新方法  return this.list.filter(item=>{  //if (item.name.indexOf(keyword)!=-1)  //注意在es6中为字符串提供了一个新方法，叫string.prototype.includes('要包含的字符串')  //如果包含返回true ,否则为false  //  if (item.name.includes(keyword)) {  return item  }  })  }  }  })  // 过滤器的定义语法  //Vue.filter('过滤器的名称',function(data){})  //过滤器中的function中的第一个参数已经被规定死了，永远都是过滤器管道符前面传递过来的数据  Vue.filter('nameope',function(data){})  </script>  </body>  </html>  <!-- 过滤器调用时的格式 {{ name | 过滤器的名称 }} --> |

第二天9

私有过滤器

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap.css">  </head>  <body>  <div id="app">  <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3>  </div>  <div class="panel-body form-inline">  <label>  id:  <input type="text"  class="form-control" v-model="id"  name="">  </label>  <label>  Name:  <input type="text" class="form-control"  v-model="name" name="">  </label>  <!-- 在vue中使用事件绑定机制为元素指定处理函数时如果加了（）,就可以传参了 -->  <input type="button" value="添加" class="btn btn-primary" @click="add()" name="">    <label>  搜索:  <input type="text" class="form-control"  v-model="keyword" name="">  </label>  </div>  </div>  <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <!-- 之前，v-for中的数据都是从data上的list中直接渲染过来的 -->  <!-- 现在，我们自定义一个search方法，同时我们把所有的关键字通过传参的形式，传递给了search方法 -->  <!-- 在search方法内部通过for循环，把所有符合搜索关键字的数据保存到一个新数组中 ，返回-->  <tr v-for="item in search(keyword)" :key="item.id">  <td>{{item.id}}</td>  <td v-text="item.name"></td>  <td>{{item.ctime | dateFormat }}</td>  <td @click.prevent="del(item.id)">删除</td>  </tr>  </tbody>  </table>  </div>  <div id="app2">  <h3>{{dt | dateFormat}}</h3>  </div>  <script type="text/javascript">  //全局的过滤器，进行时间的格式化  //全局过滤器，即所有的vue实例都共享  Vue.filter('dateFormat',function(datestr,pattern=''){  //根据给定的字符串得到特定的时间  var dt=new Date(datestr)    var y=dt.getFullYear();  var m=dt.getMonth()+1;  var d=dt.getDate();  //return y + '-' + m + '-' + d  //模板字符串  //tab上面的键点两下，中间写  //·yyyy-mm-dd·    //两种方式如果不传参 1 pattern为true 2.形参pattern=‘’  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss}`  }  })  //如何自定义私有过滤器  var vm2=new Vue({  el:"#app2",  data:{  dt:new Date(),  },  methods:{},  filters:{  //定义私有过滤器  //过滤器有两个条件【过滤器名称 和 处理函数】  //过滤器调用时采取就近原则，如果私有过滤器和全局过滤器名称都一致，这时候优先调用私有过滤器    dateFormat:function(datestr,pattern=''){  var dt=new Date(datestr)  var y=dt.getFullYear();  var m=dt.getMonth()+1;  var d=dt.getDate();    if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss} ~~~`  }  },  }  })  var vm=new Vue({  el:"#app",  data:{  id:"",  name:"",  keyword:"",//搜索关键字  list:[  {id:1,name:"奔驰",ctime:new Date()},  {id:2,name:"宝马",ctime:new Date()},  ]  },  methods:{  add(){  //console.log("ok")  //分析  //1.获取到id和name直接从data上面获取  //2.组织出一个对象  //3.把这个对象调用数组的相关发发添加到当前data上的list中  //4.注意：在vue中已经实现了数据的双向绑定，每当我们修改了data中的数据，vue会默认监听到数据的改动，自动把最新的数据应用到页面上  //5.当我们意识到上面的第四步时，已入门，更多的是进行vue中model数据的操作，同时，在操作model数据的时候，指定的业务逻辑操作  var car={id:this.id,name:this.name,ctime:new Date()}  this.list.push(car)  this.id=this.name=""  },  del(id){//根据id删除数据  //分析  //1.如何根据id找到要删除的索引  //2.直接调用数组的splice方法  //方法一  /\*this.list.some((item,i)=>{  if (item.id==id) {  this.list.splice(i,1)  //在数组的some方法中如果return true，就会立即终止这个数组的后续循环  return true;  }  })\*/  //方法二  var index=this.list.findIndex(item=>{  if (item.id==id) {  return true;  }  })  //console.log(index)  this.list.splice(index,1)  },  search(keyword){  //方法一  /\* var newList=[]  this.list.forEach(item=>{  if (item.name.indexOf(keyword)!=-1) {  newList.push(item)  }    })  return newList;\*/  //方法二  //注意：forEach some filter findIndex 数组的新方法  return this.list.filter(item=>{  //if (item.name.indexOf(keyword)!=-1)  //注意在es6中为字符串提供了一个新方法，叫string.prototype.includes('要包含的字符串')  //如果包含返回true ,否则为false  //  if (item.name.includes(keyword)) {  return item  }  })  }  }  })  // 过滤器的定义语法  //Vue.filter('过滤器的名称',function(data){})  //过滤器中的function中的第一个参数已经被规定死了，永远都是过滤器管道符前面传递过来的数据  Vue.filter('nameope',function(data){})  </script>  </body>  </html>  <!-- 过滤器调用时的格式 {{ name | 过滤器的名称 }} --> |

第二天10

时间填充

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap.css">  </head>  <body>  <div id="app">  <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3>  </div>  <div class="panel-body form-inline">  <label>  id:  <input type="text"  class="form-control" v-model="id"  name="">  </label>  <label>  Name:  <input type="text" class="form-control"  v-model="name" name="">  </label>  <!-- 在vue中使用事件绑定机制为元素指定处理函数时如果加了（）,就可以传参了 -->  <input type="button" value="添加" class="btn btn-primary" @click="add()" name="">    <label>  搜索:  <input type="text" class="form-control"  v-model="keyword" name="">  </label>  </div>  </div>  <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <!-- 之前，v-for中的数据都是从data上的list中直接渲染过来的 -->  <!-- 现在，我们自定义一个search方法，同时我们把所有的关键字通过传参的形式，传递给了search方法 -->  <!-- 在search方法内部通过for循环，把所有符合搜索关键字的数据保存到一个新数组中 ，返回-->  <tr v-for="item in search(keyword)" :key="item.id">  <td>{{item.id}}</td>  <td v-text="item.name"></td>  <td>{{item.ctime | dateFormat }}</td>  <td @click.prevent="del(item.id)">删除</td>  </tr>  </tbody>  </table>  </div>  <div id="app2">  <h3>{{dt | dateFormat}}</h3>  </div>  <script type="text/javascript">  //全局的过滤器，进行时间的格式化  //全局过滤器，即所有的vue实例都共享  Vue.filter('dateFormat',function(datestr,pattern=''){  //根据给定的字符串得到特定的时间  var dt=new Date(datestr)    var y=dt.getFullYear();  var m=dt.getMonth()+1;  var d=dt.getDate();  //return y + '-' + m + '-' + d  //模板字符串  //tab上面的键点两下，中间写  //·yyyy-mm-dd·    //两种方式如果不传参 1 pattern为true 2.形参pattern=‘’  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss}`  }  })  //如何自定义私有过滤器  var vm2=new Vue({  el:"#app2",  data:{  dt:new Date(),  },  methods:{},  filters:{  //定义私有过滤器  //过滤器有两个条件【过滤器名称 和 处理函数】  //过滤器调用时采取就近原则，如果私有过滤器和全局过滤器名称都一致，这时候优先调用私有过滤器  dateFormat:function(datestr,pattern=''){  var dt=new Date(datestr)  var y=dt.getFullYear();  var m=(dt.getMonth()+1).toString().padStart(2,'0');  var d=(dt.getDate()).toString().padStart(2,'0')  // 时间补充长度  //前面加0　String.padStart(n,'0')\  //后面加0 String.padEnd(n,'0')  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss} ~~~`  }  },  }  })  var vm=new Vue({  el:"#app",  data:{  id:"",  name:"",  keyword:"",//搜索关键字  list:[  {id:1,name:"奔驰",ctime:new Date()},  {id:2,name:"宝马",ctime:new Date()},  ]  },  methods:{  add(){  //console.log("ok")  //分析  //1.获取到id和name直接从data上面获取  //2.组织出一个对象  //3.把这个对象调用数组的相关发发添加到当前data上的list中  //4.注意：在vue中已经实现了数据的双向绑定，每当我们修改了data中的数据，vue会默认监听到数据的改动，自动把最新的数据应用到页面上  //5.当我们意识到上面的第四步时，已入门，更多的是进行vue中model数据的操作，同时，在操作model数据的时候，指定的业务逻辑操作  var car={id:this.id,name:this.name,ctime:new Date()}  this.list.push(car)  this.id=this.name=""  },  del(id){//根据id删除数据  //分析  //1.如何根据id找到要删除的索引  //2.直接调用数组的splice方法  //方法一  /\*this.list.some((item,i)=>{  if (item.id==id) {  this.list.splice(i,1)  //在数组的some方法中如果return true，就会立即终止这个数组的后续循环  return true;  }  })\*/  //方法二  var index=this.list.findIndex(item=>{  if (item.id==id) {  return true;  }  })  //console.log(index)  this.list.splice(index,1)  },  search(keyword){  //方法一  /\* var newList=[]  this.list.forEach(item=>{  if (item.name.indexOf(keyword)!=-1) {  newList.push(item)  }    })  return newList;\*/  //方法二  //注意：forEach some filter findIndex 数组的新方法  return this.list.filter(item=>{  //if (item.name.indexOf(keyword)!=-1)  //注意在es6中为字符串提供了一个新方法，叫string.prototype.includes('要包含的字符串')  //如果包含返回true ,否则为false  //  if (item.name.includes(keyword)) {  return item  }  })  }  }  })  // 过滤器的定义语法  //Vue.filter('过滤器的名称',function(data){})  //过滤器中的function中的第一个参数已经被规定死了，永远都是过滤器管道符前面传递过来的数据  Vue.filter('nameope',function(data){})  </script>  </body>  </html>  <!-- 过滤器调用时的格式 {{ name | 过滤器的名称 }} --> |

第二天11

按键修饰符

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap.css">  </head>  <body>  <div id="app">  <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3>  </div>  <div class="panel-body form-inline">  <label>  id:  <input type="text"  class="form-control" v-model="id"  name="">  </label>  <label>  Name:  <!-- <input type="text" class="form-control"  v-model="name" @keyup.enter="add" name=""> -->  <!-- <input type="text" class="form-control"  v-model="name" @keyup.13="add" name=""> -->  <!-- <input type="text" class="form-control"  v-model="name" @keyup.113="add" name=""> -->  <input type="text" class="form-control"  v-model="name" @keyup.f2="add" name="">  </label>  <!-- 在vue中使用事件绑定机制为元素指定处理函数时如果加了（）,就可以传参了 -->  <input type="button" value="添加" class="btn btn-primary" @click="add()" name="">    <label>  搜索:  <input type="text" class="form-control"  v-model="keyword" name="">  </label>  </div>  </div>  <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <!-- 之前，v-for中的数据都是从data上的list中直接渲染过来的 -->  <!-- 现在，我们自定义一个search方法，同时我们把所有的关键字通过传参的形式，传递给了search方法 -->  <!-- 在search方法内部通过for循环，把所有符合搜索关键字的数据保存到一个新数组中 ，返回-->  <tr v-for="item in search(keyword)" :key="item.id">  <td>{{item.id}}</td>  <td v-text="item.name"></td>  <td>{{item.ctime | dateFormat }}</td>  <td @click.prevent="del(item.id)">删除</td>  </tr>  </tbody>  </table>  </div>  <div id="app2">  <h3>{{dt | dateFormat}}</h3>  </div>  <script type="text/javascript">  //全局的过滤器，进行时间的格式化  //全局过滤器，即所有的vue实例都共享  Vue.filter('dateFormat',function(datestr,pattern=''){  //根据给定的字符串得到特定的时间  var dt=new Date(datestr)    var y=dt.getFullYear();  var m=dt.getMonth()+1;  var d=dt.getDate();  //return y + '-' + m + '-' + d  //模板字符串  //tab上面的键点两下，中间写  //·yyyy-mm-dd·    //两种方式如果不传参 1 pattern为true 2.形参pattern=‘’  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss}`  }  })  //自定义全局按键修饰符  Vue.config.keyCodes.f2 = 113  //如何自定义私有过滤器  var vm2=new Vue({  el:"#app2",  data:{  dt:new Date(),  },  methods:{},  filters:{  //定义私有过滤器  //过滤器有两个条件【过滤器名称 和 处理函数】  //过滤器调用时采取就近原则，如果私有过滤器和全局过滤器名称都一致，这时候优先调用私有过滤器  dateFormat:function(datestr,pattern=''){  var dt=new Date(datestr)  var y=dt.getFullYear();  var m=(dt.getMonth()+1).toString().padStart(2,'0');  var d=(dt.getDate()).toString().padStart(2,'0')  // 时间补充长度  //前面加0　String.padStart(n,'0')\  //后面加0 String.padEnd(n,'0')  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss} ~~~`  }  },  }  })  var vm=new Vue({  el:"#app",  data:{  id:"",  name:"",  keyword:"",//搜索关键字  list:[  {id:1,name:"奔驰",ctime:new Date()},  {id:2,name:"宝马",ctime:new Date()},  ]  },  methods:{  add(){  //console.log("ok")  //分析  //1.获取到id和name直接从data上面获取  //2.组织出一个对象  //3.把这个对象调用数组的相关发发添加到当前data上的list中  //4.注意：在vue中已经实现了数据的双向绑定，每当我们修改了data中的数据，vue会默认监听到数据的改动，自动把最新的数据应用到页面上  //5.当我们意识到上面的第四步时，已入门，更多的是进行vue中model数据的操作，同时，在操作model数据的时候，指定的业务逻辑操作  var car={id:this.id,name:this.name,ctime:new Date()}  this.list.push(car)  this.id=this.name=""  },  del(id){//根据id删除数据  //分析  //1.如何根据id找到要删除的索引  //2.直接调用数组的splice方法  //方法一  /\*this.list.some((item,i)=>{  if (item.id==id) {  this.list.splice(i,1)  //在数组的some方法中如果return true，就会立即终止这个数组的后续循环  return true;  }  })\*/  //方法二  var index=this.list.findIndex(item=>{  if (item.id==id) {  return true;  }  })  //console.log(index)  this.list.splice(index,1)  },  search(keyword){  //方法一  /\* var newList=[]  this.list.forEach(item=>{  if (item.name.indexOf(keyword)!=-1) {  newList.push(item)  }    })  return newList;\*/  //方法二  //注意：forEach some filter findIndex 数组的新方法  return this.list.filter(item=>{  //if (item.name.indexOf(keyword)!=-1)  //注意在es6中为字符串提供了一个新方法，叫string.prototype.includes('要包含的字符串')  //如果包含返回true ,否则为false  //  if (item.name.includes(keyword)) {  return item  }  })  }  }  })  // 过滤器的定义语法  //Vue.filter('过滤器的名称',function(data){})  //过滤器中的function中的第一个参数已经被规定死了，永远都是过滤器管道符前面传递过来的数据  Vue.filter('nameope',function(data){})  </script>  </body>  </html>  <!-- 过滤器调用时的格式 {{ name | 过滤器的名称 }} -->  <!-- 按键修饰符 enter tab delete esc space up down left right -->  <!-- 键盘的码值也是可以 F2=113 enter=13 --> |

第二天12

自定义指令

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap.css">  </head>  <body>  <div id="app">  <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3>  </div>  <div class="panel-body form-inline">  <label>  id:  <input type="text"  class="form-control" v-model="id"  name="">  </label>  <label>  Name:  <!-- <input type="text" class="form-control"  v-model="name" @keyup.enter="add" name=""> -->  <!-- <input type="text" class="form-control"  v-model="name" @keyup.13="add" name=""> -->  <!-- <input type="text" class="form-control"  v-model="name" @keyup.113="add" name=""> -->  <input type="text" class="form-control"  v-model="name" @keyup.f2="add" name="">  </label>  <!-- 在vue中使用事件绑定机制为元素指定处理函数时如果加了（）,就可以传参了 -->  <input type="button" value="添加" class="btn btn-primary" @click="add()" name="">    <label>  搜索:  <!-- 注意vue中所有的指令在调用的时候，都以 v- 开头 -->  <input type="text" class="form-control"  v-model="keyword" name="" id="search" v-focus>  </label>  </div>  </div>  <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <!-- 之前，v-for中的数据都是从data上的list中直接渲染过来的 -->  <!-- 现在，我们自定义一个search方法，同时我们把所有的关键字通过传参的形式，传递给了search方法 -->  <!-- 在search方法内部通过for循环，把所有符合搜索关键字的数据保存到一个新数组中 ，返回-->  <tr v-for="item in search(keyword)" :key="item.id">  <td>{{item.id}}</td>  <td v-text="item.name"></td>  <td>{{item.ctime | dateFormat }}</td>  <td @click.prevent="del(item.id)">删除</td>  </tr>  </tbody>  </table>  </div>  <div id="app2">  <h3>{{dt | dateFormat}}</h3>  </div>  <script type="text/javascript">  //全局的过滤器，进行时间的格式化  //全局过滤器，即所有的vue实例都共享  Vue.filter('dateFormat',function(datestr,pattern=''){  //根据给定的字符串得到特定的时间  var dt=new Date(datestr)    var y=dt.getFullYear();  var m=dt.getMonth()+1;  var d=dt.getDate();  //return y + '-' + m + '-' + d  //模板字符串  //tab上面的键点两下，中间写  //·yyyy-mm-dd·    //两种方式如果不传参 1 pattern为true 2.形参pattern=‘’  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss}`  }  })  //自定义全局按键修饰符  Vue.config.keyCodes.f2 = 113  //使用 Vue.directive() 定义全局的指令，其中参数一是指令的名称，在定义的时候指令的名称前面不需要加 v- 前缀，在调用的时候必须加上 v- 前缀来进行调用，参数二十一个对象，在这个对象身上有一些指令相关的函数(钩子函数),这些函数可以在特定的阶段执行相关的操作  Vue.directive('focus',{  //每当指令绑定到元素上的时候会执行这个bind函数，这个指令只执行一次  bind:function(el){  //注意；在每个函数中，第一个参数永远是el，表示绑定了指令的那个元素，这个el参数，十一个原生的js对象  //在元素刚绑定了指令的时候，还没有插入到Dom中去，这时候调用focus方法没有作用，因为一个元素只有插入到DOM中之后，才能获取焦点  //el.focus()  },  //元素插入到dom中的时候会执行inserted函数，只执行一次  inserted:function(el){  el.focus()  },  //当vNode更新的时候会执行updated，可能会触发多次  updated:function(el){},    })  //如何自定义私有过滤器  var vm2=new Vue({  el:"#app2",  data:{  dt:new Date(),  },  methods:{},  filters:{  //定义私有过滤器  //过滤器有两个条件【过滤器名称 和 处理函数】  //过滤器调用时采取就近原则，如果私有过滤器和全局过滤器名称都一致，这时候优先调用私有过滤器  dateFormat:function(datestr,pattern=''){  var dt=new Date(datestr)  var y=dt.getFullYear();  var m=(dt.getMonth()+1).toString().padStart(2,'0');  var d=(dt.getDate()).toString().padStart(2,'0')  // 时间补充长度  //前面加0　String.padStart(n,'0')\  //后面加0 String.padEnd(n,'0')  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss} ~~~`  }  },  }  })  var vm=new Vue({  el:"#app",  data:{  id:"",  name:"",  keyword:"",//搜索关键字  list:[  {id:1,name:"奔驰",ctime:new Date()},  {id:2,name:"宝马",ctime:new Date()},  ]  },  methods:{  add(){  //console.log("ok")  //分析  //1.获取到id和name直接从data上面获取  //2.组织出一个对象  //3.把这个对象调用数组的相关发发添加到当前data上的list中  //4.注意：在vue中已经实现了数据的双向绑定，每当我们修改了data中的数据，vue会默认监听到数据的改动，自动把最新的数据应用到页面上  //5.当我们意识到上面的第四步时，已入门，更多的是进行vue中model数据的操作，同时，在操作model数据的时候，指定的业务逻辑操作  var car={id:this.id,name:this.name,ctime:new Date()}  this.list.push(car)  this.id=this.name=""  },  del(id){//根据id删除数据  //分析  //1.如何根据id找到要删除的索引  //2.直接调用数组的splice方法  //方法一  /\*this.list.some((item,i)=>{  if (item.id==id) {  this.list.splice(i,1)  //在数组的some方法中如果return true，就会立即终止这个数组的后续循环  return true;  }  })\*/  //方法二  var index=this.list.findIndex(item=>{  if (item.id==id) {  return true;  }  })  //console.log(index)  this.list.splice(index,1)  },  search(keyword){  //方法一  /\* var newList=[]  this.list.forEach(item=>{  if (item.name.indexOf(keyword)!=-1) {  newList.push(item)  }    })  return newList;\*/  //方法二  //注意：forEach some filter findIndex 数组的新方法  return this.list.filter(item=>{  //if (item.name.indexOf(keyword)!=-1)  //注意在es6中为字符串提供了一个新方法，叫string.prototype.includes('要包含的字符串')  //如果包含返回true ,否则为false  //  if (item.name.includes(keyword)) {  return item  }  })  }  }  })  // 过滤器的定义语法  //Vue.filter('过滤器的名称',function(data){})  //过滤器中的function中的第一个参数已经被规定死了，永远都是过滤器管道符前面传递过来的数据  Vue.filter('nameope',function(data){})  //搜索获取焦点  /\*document.getElementById('search').focus()\*/  </script>  </body>  </html>  <!-- 过滤器调用时的格式 {{ name | 过滤器的名称 }} -->  <!-- 按键修饰符 enter tab delete esc space up down left right -->  <!-- 键盘的码值也是可以 F2=113 enter=13 --> |

第二天13

自定义指令传参

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap.css">  </head>  <body>  <div id="app">  <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3>  </div>  <div class="panel-body form-inline">  <label>  id:  <input type="text"  class="form-control" v-model="id"  name="">  </label>  <label>  Name:  <!-- <input type="text" class="form-control"  v-model="name" @keyup.enter="add" name=""> -->  <!-- <input type="text" class="form-control"  v-model="name" @keyup.13="add" name=""> -->  <!-- <input type="text" class="form-control"  v-model="name" @keyup.113="add" name=""> -->  <input type="text" class="form-control"  v-model="name" @keyup.f2="add" name="">  </label>  <!-- 在vue中使用事件绑定机制为元素指定处理函数时如果加了（）,就可以传参了 -->  <input type="button" value="添加" class="btn btn-primary" @click="add()" name="">    <label>  搜索:  <!-- 注意vue中所有的指令在调用的时候，都以 v- 开头 -->  <input type="text" class="form-control"  v-model="keyword" name="" id="search" v-focus v-color="'blue'">  </label>  </div>  </div>  <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <!-- 之前，v-for中的数据都是从data上的list中直接渲染过来的 -->  <!-- 现在，我们自定义一个search方法，同时我们把所有的关键字通过传参的形式，传递给了search方法 -->  <!-- 在search方法内部通过for循环，把所有符合搜索关键字的数据保存到一个新数组中 ，返回-->  <tr v-for="item in search(keyword)" :key="item.id">  <td>{{item.id}}</td>  <td v-text="item.name"></td>  <td>{{item.ctime | dateFormat }}</td>  <td @click.prevent="del(item.id)">删除</td>  </tr>  </tbody>  </table>  </div>  <div id="app2">  <h3>{{dt | dateFormat}}</h3>  </div>  <script type="text/javascript">  //全局的过滤器，进行时间的格式化  //全局过滤器，即所有的vue实例都共享  Vue.filter('dateFormat',function(datestr,pattern=''){  //根据给定的字符串得到特定的时间  var dt=new Date(datestr)    var y=dt.getFullYear();  var m=dt.getMonth()+1;  var d=dt.getDate();  //return y + '-' + m + '-' + d  //模板字符串  //tab上面的键点两下，中间写  //·yyyy-mm-dd·    //两种方式如果不传参 1 pattern为true 2.形参pattern=‘’  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss}`  }  })  //自定义全局按键修饰符  Vue.config.keyCodes.f2 = 113  //使用 Vue.directive() 定义全局的指令，其中参数一是指令的名称，在定义的时候指令的名称前面不需要加 v- 前缀，在调用的时候必须加上 v- 前缀来进行调用，参数二十一个对象，在这个对象身上有一些指令相关的函数(钩子函数),这些函数可以在特定的阶段执行相关的操作  Vue.directive('focus',{  //每当指令绑定到元素上的时候会执行这个bind函数，这个指令只执行一次  bind:function(el){  //注意；在每个函数中，第一个参数永远是el，表示绑定了指令的那个元素，这个el参数，十一个原生的js对象  //在元素刚绑定了指令的时候，还没有插入到Dom中去，这时候调用focus方法没有作用，因为一个元素只有插入到DOM中之后，才能获取焦点  //el.focus()  },  //元素插入到dom中的时候会执行inserted函数，只执行一次  inserted:function(el){  el.focus()  //和JS行为相关的的操作，最好在inserted中执行，防止这个行为不生效  },  //当vNode更新的时候会执行updated，可能会触发多次  updated:function(el){},    })  //自定义一个颜色设置指令  Vue.directive('color',{  //样式，只要通过指令绑定给了元素，不管这个元素有没有插入到页面中去，这个元素肯定有了一个内联的样式  //将来元素肯定会显示到页面中去，这时候浏览器的渲染引擎比如会解析样式，应用这个元素  bind:function(el,binding){  el.style.color='red'  //和样式相关的操作，一般可以在bind中执行  //console.log(binding.name)  //console.log(binding.value)  //console.log(binding.expression)  el.style.color=binding.value;  }  })  //如何自定义私有过滤器  var vm2=new Vue({  el:"#app2",  data:{  dt:new Date(),  },  methods:{},  filters:{  //定义私有过滤器  //过滤器有两个条件【过滤器名称 和 处理函数】  //过滤器调用时采取就近原则，如果私有过滤器和全局过滤器名称都一致，这时候优先调用私有过滤器  dateFormat:function(datestr,pattern=''){  var dt=new Date(datestr)  var y=dt.getFullYear();  var m=(dt.getMonth()+1).toString().padStart(2,'0');  var d=(dt.getDate()).toString().padStart(2,'0')  // 时间补充长度  //前面加0　String.padStart(n,'0')\  //后面加0 String.padEnd(n,'0')  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss} ~~~`  }  },  }  })  var vm=new Vue({  el:"#app",  data:{  id:"",  name:"",  keyword:"",//搜索关键字  list:[  {id:1,name:"奔驰",ctime:new Date()},  {id:2,name:"宝马",ctime:new Date()},  ]  },  methods:{  add(){  //console.log("ok")  //分析  //1.获取到id和name直接从data上面获取  //2.组织出一个对象  //3.把这个对象调用数组的相关发发添加到当前data上的list中  //4.注意：在vue中已经实现了数据的双向绑定，每当我们修改了data中的数据，vue会默认监听到数据的改动，自动把最新的数据应用到页面上  //5.当我们意识到上面的第四步时，已入门，更多的是进行vue中model数据的操作，同时，在操作model数据的时候，指定的业务逻辑操作  var car={id:this.id,name:this.name,ctime:new Date()}  this.list.push(car)  this.id=this.name=""  },  del(id){//根据id删除数据  //分析  //1.如何根据id找到要删除的索引  //2.直接调用数组的splice方法  //方法一  /\*this.list.some((item,i)=>{  if (item.id==id) {  this.list.splice(i,1)  //在数组的some方法中如果return true，就会立即终止这个数组的后续循环  return true;  }  })\*/  //方法二  var index=this.list.findIndex(item=>{  if (item.id==id) {  return true;  }  })  //console.log(index)  this.list.splice(index,1)  },  search(keyword){  //方法一  /\* var newList=[]  this.list.forEach(item=>{  if (item.name.indexOf(keyword)!=-1) {  newList.push(item)  }    })  return newList;\*/  //方法二  //注意：forEach some filter findIndex 数组的新方法  return this.list.filter(item=>{  //if (item.name.indexOf(keyword)!=-1)  //注意在es6中为字符串提供了一个新方法，叫string.prototype.includes('要包含的字符串')  //如果包含返回true ,否则为false  //  if (item.name.includes(keyword)) {  return item  }  })  }  }  })  // 过滤器的定义语法  //Vue.filter('过滤器的名称',function(data){})  //过滤器中的function中的第一个参数已经被规定死了，永远都是过滤器管道符前面传递过来的数据  Vue.filter('nameope',function(data){})  //搜索获取焦点  /\*document.getElementById('search').focus()\*/  </script>  </body>  </html>  <!-- 过滤器调用时的格式 {{ name | 过滤器的名称 }} -->  <!-- 按键修饰符 enter tab delete esc space up down left right -->  <!-- 键盘的码值也是可以 F2=113 enter=13 --> |

第二天14

私有指令

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap.css">  </head>  <body>  <div id="app">  <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3>  </div>  <div class="panel-body form-inline">  <label>  id:  <input type="text"  class="form-control" v-model="id"  name="">  </label>  <label>  Name:  <!-- <input type="text" class="form-control"  v-model="name" @keyup.enter="add" name=""> -->  <!-- <input type="text" class="form-control"  v-model="name" @keyup.13="add" name=""> -->  <!-- <input type="text" class="form-control"  v-model="name" @keyup.113="add" name=""> -->  <input type="text" class="form-control"  v-model="name" @keyup.f2="add" name="">  </label>  <!-- 在vue中使用事件绑定机制为元素指定处理函数时如果加了（）,就可以传参了 -->  <input type="button" value="添加" class="btn btn-primary" @click="add()" name="">    <label>  搜索:  <!-- 注意vue中所有的指令在调用的时候，都以 v- 开头 -->  <input type="text" class="form-control"  v-model="keyword" name="" id="search" v-focus v-color="'blue'">  </label>  </div>  </div>  <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <!-- 之前，v-for中的数据都是从data上的list中直接渲染过来的 -->  <!-- 现在，我们自定义一个search方法，同时我们把所有的关键字通过传参的形式，传递给了search方法 -->  <!-- 在search方法内部通过for循环，把所有符合搜索关键字的数据保存到一个新数组中 ，返回-->  <tr v-for="item in search(keyword)" :key="item.id">  <td>{{item.id}}</td>  <td v-text="item.name"></td>  <td>{{item.ctime | dateFormat }}</td>  <td @click.prevent="del(item.id)">删除</td>  </tr>  </tbody>  </table>  </div>  <div id="app2">  <h3 v-color="'blue'" v-fontweight="800">{{dt | dateFormat}}</h3>  </div>  <script type="text/javascript">  //全局的过滤器，进行时间的格式化  //全局过滤器，即所有的vue实例都共享  Vue.filter('dateFormat',function(datestr,pattern=''){  //根据给定的字符串得到特定的时间  var dt=new Date(datestr)    var y=dt.getFullYear();  var m=dt.getMonth()+1;  var d=dt.getDate();  //return y + '-' + m + '-' + d  //模板字符串  //tab上面的键点两下，中间写  //·yyyy-mm-dd·    //两种方式如果不传参 1 pattern为true 2.形参pattern=‘’  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss}`  }  })  //自定义全局按键修饰符  Vue.config.keyCodes.f2 = 113  //使用 Vue.directive() 定义全局的指令，其中参数一是指令的名称，在定义的时候指令的名称前面不需要加 v- 前缀，在调用的时候必须加上 v- 前缀来进行调用，参数二十一个对象，在这个对象身上有一些指令相关的函数(钩子函数),这些函数可以在特定的阶段执行相关的操作  Vue.directive('focus',{  //每当指令绑定到元素上的时候会执行这个bind函数，这个指令只执行一次  bind:function(el){  //注意；在每个函数中，第一个参数永远是el，表示绑定了指令的那个元素，这个el参数，十一个原生的js对象  //在元素刚绑定了指令的时候，还没有插入到Dom中去，这时候调用focus方法没有作用，因为一个元素只有插入到DOM中之后，才能获取焦点  //el.focus()  },  //元素插入到dom中的时候会执行inserted函数，只执行一次  inserted:function(el){  el.focus()  //和JS行为相关的的操作，最好在inserted中执行，防止这个行为不生效  },  //当vNode更新的时候会执行updated，可能会触发多次  updated:function(el){},    })  //自定义一个颜色设置指令  Vue.directive('color',{  //样式，只要通过指令绑定给了元素，不管这个元素有没有插入到页面中去，这个元素肯定有了一个内联的样式  //将来元素肯定会显示到页面中去，这时候浏览器的渲染引擎比如会解析样式，应用这个元素  bind:function(el,binding){  el.style.color='red'  //binding为形参，可以为别的名字  //和样式相关的操作，一般可以在bind中执行  //console.log(binding.name)  //console.log(binding.value)  //console.log(binding.expression)  el.style.color=binding.value;  }  })  //如何自定义私有过滤器  var vm2=new Vue({  el:"#app2",  data:{  dt:new Date(),  },  methods:{},  filters:{  //定义私有过滤器  //过滤器有两个条件【过滤器名称 和 处理函数】  //过滤器调用时采取就近原则，如果私有过滤器和全局过滤器名称都一致，这时候优先调用私有过滤器  dateFormat:function(datestr,pattern=''){  var dt=new Date(datestr)  var y=dt.getFullYear();  var m=(dt.getMonth()+1).toString().padStart(2,'0');  var d=(dt.getDate()).toString().padStart(2,'0')  // 时间补充长度  //前面加0　String.padStart(n,'0')\  //后面加0 String.padEnd(n,'0')  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss} ~~~`  }  },  },  directives:{//自定义私有指令  'fontweight':{  bind:function(el,binding){  el.style.fontWeight=binding.value  }  }  }  })  var vm=new Vue({  el:"#app",  data:{  id:"",  name:"",  keyword:"",//搜索关键字  list:[  {id:1,name:"奔驰",ctime:new Date()},  {id:2,name:"宝马",ctime:new Date()},  ]  },  methods:{  add(){  //console.log("ok")  //分析  //1.获取到id和name直接从data上面获取  //2.组织出一个对象  //3.把这个对象调用数组的相关发发添加到当前data上的list中  //4.注意：在vue中已经实现了数据的双向绑定，每当我们修改了data中的数据，vue会默认监听到数据的改动，自动把最新的数据应用到页面上  //5.当我们意识到上面的第四步时，已入门，更多的是进行vue中model数据的操作，同时，在操作model数据的时候，指定的业务逻辑操作  var car={id:this.id,name:this.name,ctime:new Date()}  this.list.push(car)  this.id=this.name=""  },  del(id){//根据id删除数据  //分析  //1.如何根据id找到要删除的索引  //2.直接调用数组的splice方法  //方法一  /\*this.list.some((item,i)=>{  if (item.id==id) {  this.list.splice(i,1)  //在数组的some方法中如果return true，就会立即终止这个数组的后续循环  return true;  }  })\*/  //方法二  var index=this.list.findIndex(item=>{  if (item.id==id) {  return true;  }  })  //console.log(index)  this.list.splice(index,1)  },  search(keyword){  //方法一  /\* var newList=[]  this.list.forEach(item=>{  if (item.name.indexOf(keyword)!=-1) {  newList.push(item)  }    })  return newList;\*/  //方法二  //注意：forEach some filter findIndex 数组的新方法  return this.list.filter(item=>{  //if (item.name.indexOf(keyword)!=-1)  //注意在es6中为字符串提供了一个新方法，叫string.prototype.includes('要包含的字符串')  //如果包含返回true ,否则为false  //  if (item.name.includes(keyword)) {  return item  }  })  }  }  })  // 过滤器的定义语法  //Vue.filter('过滤器的名称',function(data){})  //过滤器中的function中的第一个参数已经被规定死了，永远都是过滤器管道符前面传递过来的数据  Vue.filter('nameope',function(data){})  //搜索获取焦点  /\*document.getElementById('search').focus()\*/  </script>  </body>  </html>  <!-- 过滤器调用时的格式 {{ name | 过滤器的名称 }} -->  <!-- 按键修饰符 enter tab delete esc space up down left right -->  <!-- 键盘的码值也是可以 F2=113 enter=13 --> |

第二天15

自定义指令函数简写

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap.css">  </head>  <body>  <div id="app">  <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3>  </div>  <div class="panel-body form-inline">  <label>  id:  <input type="text"  class="form-control" v-model="id"  name="">  </label>  <label>  Name:  <!-- <input type="text" class="form-control"  v-model="name" @keyup.enter="add" name=""> -->  <!-- <input type="text" class="form-control"  v-model="name" @keyup.13="add" name=""> -->  <!-- <input type="text" class="form-control"  v-model="name" @keyup.113="add" name=""> -->  <input type="text" class="form-control"  v-model="name" @keyup.f2="add" name="">  </label>  <!-- 在vue中使用事件绑定机制为元素指定处理函数时如果加了（）,就可以传参了 -->  <input type="button" value="添加" class="btn btn-primary" @click="add()" name="">    <label>  搜索:  <!-- 注意vue中所有的指令在调用的时候，都以 v- 开头 -->  <input type="text" class="form-control"  v-model="keyword" name="" id="search" v-focus v-color="'blue'">  </label>  </div>  </div>  <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <!-- 之前，v-for中的数据都是从data上的list中直接渲染过来的 -->  <!-- 现在，我们自定义一个search方法，同时我们把所有的关键字通过传参的形式，传递给了search方法 -->  <!-- 在search方法内部通过for循环，把所有符合搜索关键字的数据保存到一个新数组中 ，返回-->  <tr v-for="item in search(keyword)" :key="item.id">  <td>{{item.id}}</td>  <td v-text="item.name"></td>  <td>{{item.ctime | dateFormat }}</td>  <td @click.prevent="del(item.id)">删除</td>  </tr>  </tbody>  </table>  </div>  <div id="app2">  <h3 v-color="'blue'" v-fontweight="800" v-fontsize="40">{{dt | dateFormat}}</h3>  </div>  <script type="text/javascript">  //全局的过滤器，进行时间的格式化  //全局过滤器，即所有的vue实例都共享  Vue.filter('dateFormat',function(datestr,pattern=''){  //根据给定的字符串得到特定的时间  var dt=new Date(datestr)    var y=dt.getFullYear();  var m=dt.getMonth()+1;  var d=dt.getDate();  //return y + '-' + m + '-' + d  //模板字符串  //tab上面的键点两下，中间写  //·yyyy-mm-dd·    //两种方式如果不传参 1 pattern为true 2.形参pattern=‘’  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss}`  }  })  //自定义全局按键修饰符  Vue.config.keyCodes.f2 = 113  //使用 Vue.directive() 定义全局的指令，其中参数一是指令的名称，在定义的时候指令的名称前面不需要加 v- 前缀，在调用的时候必须加上 v- 前缀来进行调用，参数二十一个对象，在这个对象身上有一些指令相关的函数(钩子函数),这些函数可以在特定的阶段执行相关的操作  Vue.directive('focus',{  //每当指令绑定到元素上的时候会执行这个bind函数，这个指令只执行一次  bind:function(el){  //注意；在每个函数中，第一个参数永远是el，表示绑定了指令的那个元素，这个el参数，十一个原生的js对象  //在元素刚绑定了指令的时候，还没有插入到Dom中去，这时候调用focus方法没有作用，因为一个元素只有插入到DOM中之后，才能获取焦点  //el.focus()  },  //元素插入到dom中的时候会执行inserted函数，只执行一次  inserted:function(el){  el.focus()  //和JS行为相关的的操作，最好在inserted中执行，防止这个行为不生效  },  //当vNode更新的时候会执行updated，可能会触发多次  updated:function(el){},    })  //自定义一个颜色设置指令  Vue.directive('color',{  //样式，只要通过指令绑定给了元素，不管这个元素有没有插入到页面中去，这个元素肯定有了一个内联的样式  //将来元素肯定会显示到页面中去，这时候浏览器的渲染引擎比如会解析样式，应用这个元素  bind:function(el,binding){  el.style.color='red'  //binding为形参，可以为别的名字  //和样式相关的操作，一般可以在bind中执行  //console.log(binding.name)  //console.log(binding.value)  //console.log(binding.expression)  el.style.color=binding.value;  }  })  //如何自定义私有过滤器  var vm2=new Vue({  el:"#app2",  data:{  dt:new Date(),  },  methods:{},  filters:{  //定义私有过滤器  //过滤器有两个条件【过滤器名称 和 处理函数】  //过滤器调用时采取就近原则，如果私有过滤器和全局过滤器名称都一致，这时候优先调用私有过滤器  dateFormat:function(datestr,pattern=''){  var dt=new Date(datestr)  var y=dt.getFullYear();  var m=(dt.getMonth()+1).toString().padStart(2,'0');  var d=(dt.getDate()).toString().padStart(2,'0')  // 时间补充长度  //前面加0　String.padStart(n,'0')\  //后面加0 String.padEnd(n,'0')  if (pattern.toLowerCase()=='yyyy-mm-dd') {  return `${y}-${m}-${d}`  }else{  var hh=dt.getHours();  var mm=dt.getMinutes();  var ss=dt.getSeconds();  return `${y}-${m}-${d} ${hh}:${mm}:${ss} ~~~`  }  },  },  directives:{//自定义私有指令  //设置字体粗细  'fontweight':{  bind:function(el,binding){  el.style.fontWeight=binding.value  }  },  'fontsize':function(el,binding){  //注意这个function等同于把白马写到了bind和update中去了  el.style.fontSize=parseInt(binding.value) + 'px'  }  }  })  var vm=new Vue({  el:"#app",  data:{  id:"",  name:"",  keyword:"",//搜索关键字  list:[  {id:1,name:"奔驰",ctime:new Date()},  {id:2,name:"宝马",ctime:new Date()},  ]  },  methods:{  add(){  //console.log("ok")  //分析  //1.获取到id和name直接从data上面获取  //2.组织出一个对象  //3.把这个对象调用数组的相关发发添加到当前data上的list中  //4.注意：在vue中已经实现了数据的双向绑定，每当我们修改了data中的数据，vue会默认监听到数据的改动，自动把最新的数据应用到页面上  //5.当我们意识到上面的第四步时，已入门，更多的是进行vue中model数据的操作，同时，在操作model数据的时候，指定的业务逻辑操作  var car={id:this.id,name:this.name,ctime:new Date()}  this.list.push(car)  this.id=this.name=""  },  del(id){//根据id删除数据  //分析  //1.如何根据id找到要删除的索引  //2.直接调用数组的splice方法  //方法一  /\*this.list.some((item,i)=>{  if (item.id==id) {  this.list.splice(i,1)  //在数组的some方法中如果return true，就会立即终止这个数组的后续循环  return true;  }  })\*/  //方法二  var index=this.list.findIndex(item=>{  if (item.id==id) {  return true;  }  })  //console.log(index)  this.list.splice(index,1)  },  search(keyword){  //方法一  /\* var newList=[]  this.list.forEach(item=>{  if (item.name.indexOf(keyword)!=-1) {  newList.push(item)  }    })  return newList;\*/  //方法二  //注意：forEach some filter findIndex 数组的新方法  return this.list.filter(item=>{  //if (item.name.indexOf(keyword)!=-1)  //注意在es6中为字符串提供了一个新方法，叫string.prototype.includes('要包含的字符串')  //如果包含返回true ,否则为false  //  if (item.name.includes(keyword)) {  return item  }  })  }  }  })  // 过滤器的定义语法  //Vue.filter('过滤器的名称',function(data){})  //过滤器中的function中的第一个参数已经被规定死了，永远都是过滤器管道符前面传递过来的数据  Vue.filter('nameope',function(data){})  //搜索获取焦点  /\*document.getElementById('search').focus()\*/  </script>  </body>  </html>  <!-- 过滤器调用时的格式 {{ name | 过滤器的名称 }} -->  <!-- 按键修饰符 enter tab delete esc space up down left right -->  <!-- 键盘的码值也是可以 F2=113 enter=13 --> |

第二天16

VUE的生命周期

从VUE实例创建、运行、到销毁期间，总是伴随着各种各样的事件，这些事件被称为生命周期

生命周期钩子：就是生命周期时间的别名

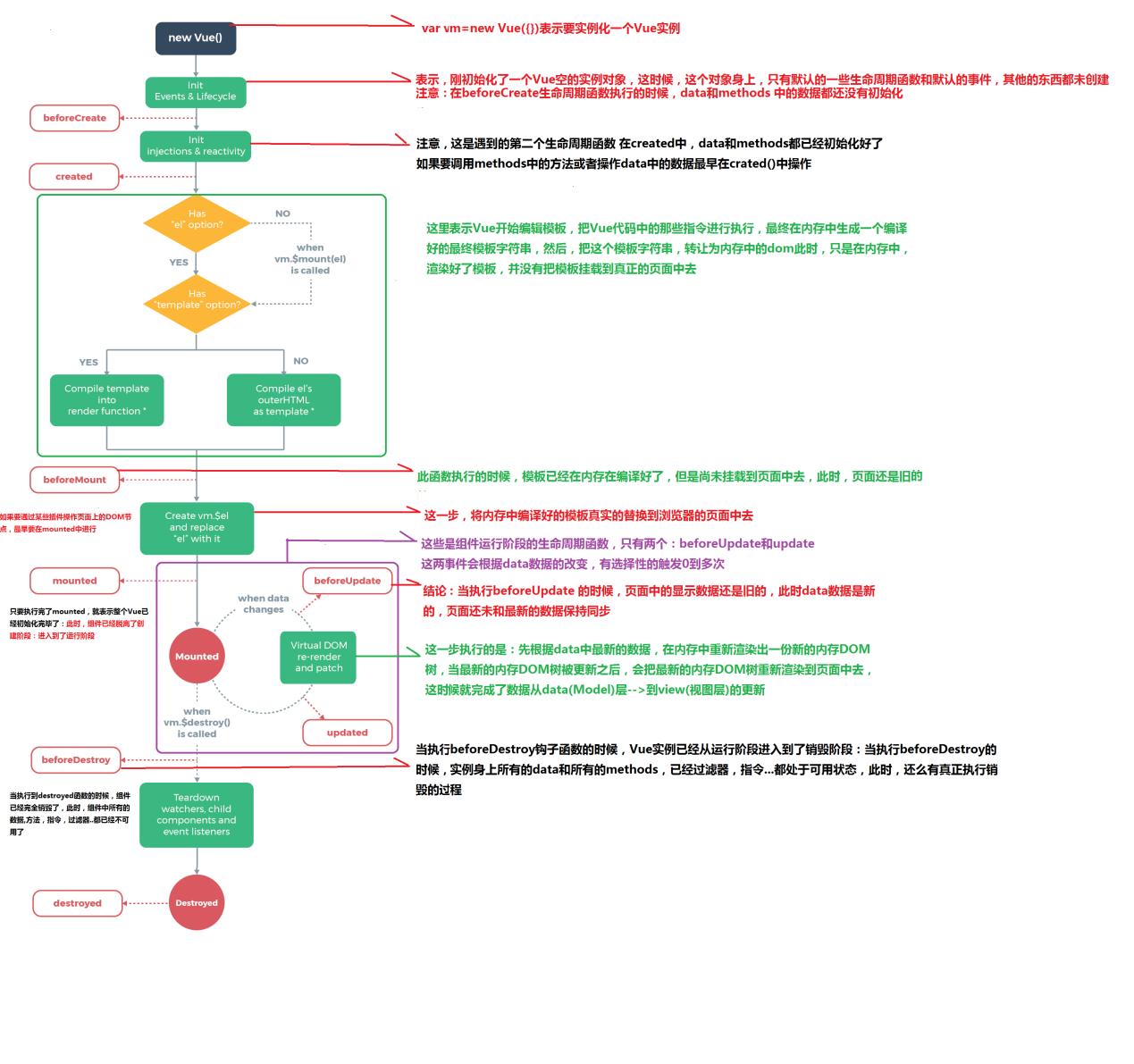
生命周期钩子 = 生命周期函数 =生命周期事件

分类

创建期间生命周期函数

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <h3 id="h3">{{msg}}</h3>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  msg:'ok',  },  methods:{  show(){  console.log("show")  }  },  beforeCreate(){  //这是我们遇到的第一个生命周期函数，表示实例完全被创建出来之前会执行它  //console.log(this.msg)  //console.log(this.show)  //注意：在beforeCreate生命周期函数执行的时候，data和methods 中的数据都还没有初始化  },  created(){  //注意，这是遇到的第二个生命周期函数  /\*console.log(this.msg)  console.log(this.show)\*/  //在created中，data和methods都已经初始化好了  //如果要调用methods中的方法或者操作data中的数据最早在crated()中操作  },  beforeMount(){  //这是遇到的第三个生命周期函数，表示模板已经值啊内存中编译完成了，但是尚未把模板渲染到页面中去  /\*console.log(document.getElementById('h3').innerText)\*/  //在beforeMount执行的时候，页面中的元素还没有被真正替换过来，只是之前的一些模板字符串  },  mounted(){  //这是遇到的第四个生命周期函数，表示，内存中的模板，已经真实的挂载到了页面中，用户已经可以看到渲染好的页面  console.log(document.getElementById('h3').innerText)  //注意，mounted是实例创建期间的最后一个生命周期函数，当执行完mounted就表示，实例已经被完全创建好了，此时，实例已经静静的放在内存中了  }  })  </script>  </body>  </html> |

第二天17



运行期间生命周期函数

销毁期间生命周期函数

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <input type="button" value="修改" @click="msg='no'" name="">  <h3 id="h3">{{msg}}</h3>  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{  msg:'ok',  },  methods:{  show(){  console.log("show")  }  },  beforeCreate(){  //这是我们遇到的第一个生命周期函数，表示实例完全被创建出来之前会执行它  //console.log(this.msg)  //console.log(this.show)  //注意：在beforeCreate生命周期函数执行的时候，data和methods 中的数据都还没有初始化  },  created(){  //注意，这是遇到的第二个生命周期函数  /\*console.log(this.msg)  console.log(this.show)\*/  //在created中，data和methods都已经初始化好了  //如果要调用methods中的方法或者操作data中的数据最早在crated()中操作  },  beforeMount(){  //这是遇到的第三个生命周期函数，表示模板已经值啊内存中编译完成了，但是尚未把模板渲染到页面中去  /\*console.log(document.getElementById('h3').innerText)\*/  //在beforeMount执行的时候，页面中的元素还没有被真正替换过来，只是之前的一些模板字符串  },  mounted(){  //这是遇到的第四个生命周期函数，表示，内存中的模板，已经真实的挂载到了页面中，用户已经可以看到渲染好的页面  /\*console.log(document.getElementById('h3').innerText)\*/  //注意，mounted是实例创建期间的最后一个生命周期函数，当执行完mounted就表示，实例已经被完全创建好了，此时，实例已经静静的放在内存中了  },  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //接下来的是运行中的两个事件  beforeUpdate(){  //这时候表示我们的界面还没有被更新，数据肯定被更新了  /\*console.log(document.getElementById('h3').innerText)  console.log(this.msg)\*/  //结论：当执行beforeUpdate 的时候，页面中的显示数据还是旧的，此时data数据是新的，页面还未和最新的数据保持同步  },  updated(){  console.log(document.getElementById('h3').innerText)  console.log(this.msg)  //updated事件执行的时候，页面和data数据已经保持同步了，都是最新的  }  })  </script>  </body>  </html> |

第二天18

Vue-resource实现get post jsonp请求

除了vue-resource之外，还可以使用axios第三方包实现数据的请求

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

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  <script type="text/javascript" src="./lib/vue.js"></script>  <!-- 注意，vue-resource依赖于Vue所有先后顺序要注意-->  <!-- this.$http.get this.$http.post -->  <script type="text/javascript"  src="./lib/vue-resource.js"></script>  </head>  <body>  <div id="app">  <input type="button" value="get请求" @click="getInfo" name="">  <input type="button" value="post请求" @click="postInfo" name="">  <input type="button" value="jsonp请求" @click="jsonpInfo" name="">  </div>  <script type="text/javascript">  var vm=new Vue({  el:"#app",  data:{},  methods:{  getInfo(){  //发起get请求  //当发起get请求之后，通过.then来设置成功的回调函数  this.$http.get('http://vue.studyit.io/api/getlunbo').then(function(result){  //通过result。body拿到服务器返回的成功的数据  console.log(result.body)  })  },  postInfo(){  //发起post请求 application/x-www-form-urlencoded  //手动发起的post请求默认没有表单格式，所以，有的服务器处理不了  //通过post方法的第三个参数的，{emulateJson:true}设置，提交的内容类型为普通表单数据格式  this.$http.post('http://vue.studyit.io/api/post',{},{emulateJSON:true}).then(result=>{  console.log(result.body)  })  },  jsonpInfo(){  //发起jsonp请求  this.$http.jsonp('http://vue.studyit.io/api/jsonp').then(result=>{  console.log(result.body)  })  }  }  })  </script>  </body>  </html> |

第二天19

浏览器不允许跨域请求

使用了node.js

客户端代码

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title></title>  </head>  <body>  <script type="text/javascript">  function show(data){  console.log(data)  }  </script>  <script type="text/javascript" src="http://127.0.0.1:3000/getscript?callback=showInfo">    </script>  </body>  </html> |

服务器端代码

|  |
| --- |
| //導入http內置模塊  const http= require('http')  //这个核心模块能帮我们解析URL地址从而拿到pathname query  const urlModule = require("url")  //創建一個http服務器  const server = http.createServer()  //監聽http服務器的request請求  server.on('request',function(req,res){  //const url=req.url  const {pathname:url,query}=urlModule.parse(req.url,true)  if (url==='/getscript') {  //var scriptStr= 'show()'  var data ={  name:'xjj',  age:18,  gender:'nv'  }  var scriptStr =`${query.callback}(${JSON.stringify(data)})`  res.end(scriptStr)  }else{  res.end('404')  }  })  //指定端口號並啟動服務器監聽  server.listen(3000,function(){  console.log("server listen at http:127.0.0.1:3000")  }) |

是由服务器主动调用前台接口返回数据

第三天1 2 3增删

|  |
| --- |
| 页面  <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.min.js"></script>  <script src="./lib/vue-resource.js"></script>  <link href="./lib/bootstrap/bootstrap.css" rel="stylesheet" type="text/css" media="screen">  </head>  <body>  <div id="app">    <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3> </div>  <div class="panel-body form-inline">  <label >  Name:  <input type="text" v-model="name" class="form-control">  </label>  <input type="button" value="添加" @click="add()" class="btn btn-primary">  </div>  </div>    <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <tr v-for="item in list" key="item.id">  <td>{{item.id}}</td>  <td>{{item.name}}</td>  <td>{{item.ctime}}</td>  <td><a href="#" @click.prevent="del(item.id)">删除</a></td>  </tr>    </tbody>  </table>    </div>  <script>  //如果我们通过全局配置了请求的数据接口 根域名，则，在每次单独发起http请求的时候，请求的url,应该以相对路径开头，前面不能带/，否则不会启用根路径做拼接  Vue.http.options.root = 'http:127.0.0.1:9527';  //全局启用emuemulateJSON选项  Vue.http.options.emulateJSON=true  var vm=new Vue({  el:"#app",  data:{  name:'',  list:[//存放所有品牌列表的地方  {id:1,name:"AAA",ctime:new Date()},  {id:2,name:"BBB",ctime:new Date()}  ]  },    methods:{    getAllList(){  // 1.分析：由于已经导入了vue-resource这个包，所以可以直接通过 this.$http来发起数据请求  // 2.根据接口API 文档，知道获取列表的时候应该发起GET请求  // 3.通过this.$http.get("url").then(function(result){})  // 4.当通过then 指定回调函数之后，在回调函数中，可以拿到数据服务器返回的result  // 5.先判断result.status是否等于0，如果等于0就成功了，可以把result.message赋值给this.list；如果不等于0，可以弹框提醒获取数据失败  // this.$http.get("http:127.0.0.1:9527/api/getprodlist").then((result)=>{    // var result = result.body;  // console.log(result)  // if (result.status===0) {  // this.list=result.message;  // } else {  // alert("get data err")  // }  // },function(){})  this.$http.jsonp('api/getprodlist', {},  {headers: {},emulateJSON: true }).then((result)=>{    var result = JSON.parse(result.body);  console.log(result)  if (result.status===0) {  this.list=result.message;  } else {  alert("get data err")  }  },function(){})  },  add(){    //分析  // 1.经查看数据api 要发送一个post请求 this.$http.post  // 2.this.$http.post中接受三个参数：  // 2.1第一个参数：要请求的URL地址  // 2.2第二个参数：要提交给服务器的数据，要以对象形式提交给服务器  // 2.3第三个参数：是一个page对象，要以哪种表单数据提交过去 emulateJSON:true  // 以普通表单格式将数据提交给服务器  // application/x-www-form-urlencoded  // 3在post方法中，使用.then来设置成功的回调函数，如果想要拿到成功的结果，需要result.body  this.$http.post('api/addproduct',{name:this.name}).then(result=>{  //console.log(result)  if (result.status===200) {  console.log("ok")  this.getAllList();  this.name=""  } else {  alert("添加失败")  }  })  },  del(id){  this.$http.jsonp('api/delproduct/?id='+id, {},  {headers: {},emulateJSON: true }).then((result)=>{    var result = JSON.parse(result.body);  console.log(result)  if (result.status===0) {  this.getAllList();  } else {  alert("get data err")  }  },function(){})  }  },  created(){  //当vm 实例的data和method 初始化完毕后，vm实例会自动执行created 这个生命周期函数  this.getAllList();  },    })  </script>  </body>  </html> |
| 服务器  const http=require("http")  const querystring=require("querystring")  const server=http.createServer()  const mysql=require("mysql")  const db=mysql.createConnection({host:"localhost",user:"root",password:"root",database:"product"})  server.listen(9527,()=>{  console.log("http:127.0.0.1:9527")  })  const urlModule=require("url")  server.on("request",(req,res)=>{  let origin=req.headers.origin;  res.setHeader('Access-Control-Allow-Origin',"\*");  res.setHeader('Access-Control-Allow-Headers','Content-Type');  const {pathname:url,query} = urlModule.parse(req.url,true)  console.log(url)  if (url=="/api/getprodlist") {  var relust={};  db.query(`select \* from product`,(err,data)=>{  if (err) {  relust.status=-1;  relust.message="database err";  } else {  relust.status=0;  relust.message=data;  }  var scriptStr=`${query.callback}(${JSON.stringify(relust)})`  res.end(scriptStr)  })  } else if(url==='/api/addproduct'){  var str="";  var relusts={};  req.on("data",function(data){  str+=data  })  req.on("end",function(){  const Info=querystring.parse(str)    if (Info.name!='') {    db.query(`insert into product(name,ctime) values('${Info.name}','${new Date().toLocaleString()}')`,(err,data)=>{  if (err) {  console.log(err)  relusts.status=-1  relusts.message="err"  } else {    relusts.status=0  relusts.message="success"  }  })    } else {  relusts.status=-1  relusts.message="err"  }  console.log(relusts.message);  var scriptStr=`${query.callback}(${JSON.stringify(relusts)})`  console.log(scriptStr)  res.end(scriptStr)  })      }else if(url=="/api/delproduct/"){  const {query:param,query}=urlModule.parse(req.url, true);  var relust={};  db.query(`delete from product where id='${param.id}'`,(err,data)=>{  res.setHeader('Content-Type','text/plain;charset=utf-8')  if (err) {  relust.status=-1;  relust.message="database err";  } else {  relust.status=0;  relust.message='ok';  }  var scriptStr=`${query.callback}(${JSON.stringify(relust)})`  res.end(scriptStr)  })  }else{  console.log("404")  }    }) |

第三天4

//如果我们通过全局配置了请求的数据接口 根域名，则，在每次单独发起http请求的时候，请求的url,应该以相对路径开头，前面不能带/，否则不会启用根路径做拼接

Vue.http.options.root = 'http:127.0.0.1:9527';

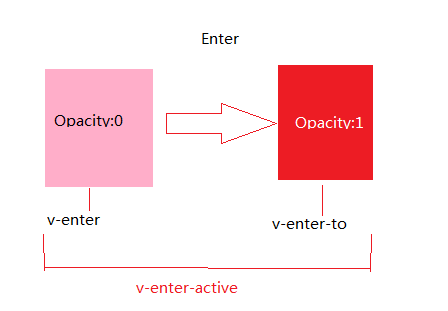
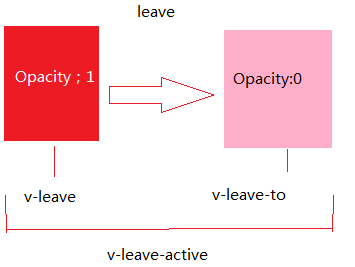
|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.min.js"></script>  <script src="./lib/vue-resource.js"></script>  <link href="./lib/bootstrap/bootstrap.css" rel="stylesheet" type="text/css" media="screen">  </head>  <body>  <div id="app">    <div class="panel panel-primary">  <div class="panel-heading">  <h3 class="panel-title">添加品牌</h3> </div>  <div class="panel-body form-inline">  <label >  Name:  <input type="text" v-model="name" class="form-control">  </label>  <input type="button" value="添加" @click="add()" class="btn btn-primary">  </div>  </div>    <table class="table table-bordered table-hover table-striped">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Ctime</th>  <th>Operation</th>  </tr>  </thead>  <tbody>  <tr v-for="item in list" key="item.id">  <td>{{item.id}}</td>  <td>{{item.name}}</td>  <td>{{item.ctime}}</td>  <td><a href="#" @click.prevent="del(item.id)">删除</a></td>  </tr>    </tbody>  </table>    </div>  <script>  //如果我们通过全局配置了请求的数据接口 根域名，则，在每次单独发起http请求的时候，请求的url,应该以相对路径开头，前面不能带/，否则不会启用根路径做拼接  Vue.http.options.root = 'http:127.0.0.1:9527';  var vm=new Vue({  el:"#app",  data:{  name:'',  list:[//存放所有品牌列表的地方  {id:1,name:"AAA",ctime:new Date()},  {id:2,name:"BBB",ctime:new Date()}  ]  },    methods:{    getAllList(){  // 1.分析：由于已经导入了vue-resource这个包，所以可以直接通过 this.$http来发起数据请求  // 2.根据接口API 文档，知道获取列表的时候应该发起GET请求  // 3.通过this.$http.get("url").then(function(result){})  // 4.当通过then 指定回调函数之后，在回调函数中，可以拿到数据服务器返回的result  // 5.先判断result.status是否等于0，如果等于0就成功了，可以把result.message赋值给this.list；如果不等于0，可以弹框提醒获取数据失败  // this.$http.get("http:127.0.0.1:9527/api/getprodlist").then((result)=>{    // var result = result.body;  // console.log(result)  // if (result.status===0) {  // this.list=result.message;  // } else {  // alert("get data err")  // }  // },function(){})  this.$http.jsonp('api/getprodlist', {},  {headers: {},emulateJSON: true }).then((result)=>{    var result = JSON.parse(result.body);  console.log(result)  if (result.status===0) {  this.list=result.message;  } else {  alert("get data err")  }  },function(){})  },  add(){    //分析  // 1.经查看数据api 要发送一个post请求 this.$http.post  // 2.this.$http.post中接受三个参数：  // 2.1第一个参数：要请求的URL地址  // 2.2第二个参数：要提交给服务器的数据，要以对象形式提交给服务器  // 2.3第三个参数：是一个page对象，要以哪种表单数据提交过去 emulateJSON:true  // 以普通表单格式将数据提交给服务器  // application/x-www-form-urlencoded  // 3在post方法中，使用.then来设置成功的回调函数，如果想要拿到成功的结果，需要result.body  this.$http.post('api/addproduct',{name:this.name},{headers: {"Content-Type":"application/x-www-form-urlencoded"},emulateJSON:true}).then(result=>{  //console.log(result)  if (result.status===200) {  console.log("ok")  this.getAllList();  } else {  alert("添加失败")  }  })  },  del(id){  this.$http.jsonp('api/delproduct/?id='+id, {},  {headers: {},emulateJSON: true }).then((result)=>{    var result = JSON.parse(result.body);  console.log(result)  if (result.status===0) {  this.getAllList();  } else {  alert("get data err")  }  },function(){})  }  },  created(){  //当vm 实例的data和method 初始化完毕后，vm实例会自动执行created 这个生命周期函数  this.getAllList();  },  })  </script>  </body>  </html> |

第三天5

|  |
| --- |
| //全局启用emuemulateJSON选项  Vue.http.options.emulateJSON=true  表单保存时，设置全局变量，防止意外发生  this.$http.post('api/addproduct',{name:this.name}).then(result=>{  //console.log(result)  if (result.status===200) {  console.log("ok")  this.getAllList();  this.name=""  } else {  alert("添加失败")  }  }) |

第三天6 vue 动画

使用过度类名

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <input type="button" value="toggle" @click="flag=!flag">  <!--需求，点击按钮让h3显示，再点击，让h3隐藏 -->  <h3 v-if="flag">这是一个h3</h3>  </div>  <script>  var vm=new Vue({  el:'#app',  data:{  flag:false,  },  methods:{}  })  </script>  </body>  </html> |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  <!-- 2.自定义两组样式，来控制transition内部的元素实现动画 -->  <style>  /\*v-enter 【这是一个时间点】是进入之前，元素的起始状态，此时还没有开始进入\*/  /\*v-leave-to【这是一个时间点】是动画离开之后，离开的终止状态，此时，动画已经结束\*/  .v-enter,  .v-leave-to{  opacity: 0;  transform: translateX(150px)  }  /\*v-enter-active【入场动画的时间段】\*/  /\*v-leave-active【离场动画的时间段】\*/  .v-enter-active,  .v-leave-active{  transition: all 0.4s ease;  }  </style>  </head>  <body>  <div id="app">  <input type="button" value="toggle" @click="flag=!flag">  <!--需求，点击按钮让h3显示，再点击，让h3隐藏 -->  <!-- 1.使用transition元素把需要被动画控制的元素包裹起来 -->  <!-- transition 元素是vue官方提供的 -->  <transition>  <h3 v-if="flag">这是一个h3</h3>  </transition>  </div>  <script>  var vm=new Vue({  el:'#app',  data:{  flag:false,  },  methods:{}  })  </script>  </body>  </html> |

第三天7动画-自定义v-前缀

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  <!-- 2.自定义两组样式，来控制transition内部的元素实现动画 -->  <style>  /\*v-enter 【这是一个时间点】是进入之前，元素的起始状态，此时还没有开始进入\*/  /\*v-leave-to【这是一个时间点】是动画离开之后，离开的终止状态，此时，动画已经结束\*/  .v-enter,  .v-leave-to{  opacity: 0;  transform: translateX(150px)  }  /\*v-enter-active【入场动画的时间段】\*/  /\*v-leave-active【离场动画的时间段】\*/  .v-enter-active,  .v-leave-active{  transition: all 0.4s ease;  }  .my-enter,  .my-leave-to{  opacity: 0;  transform: translateY(70px)  }  .my-enter-active,  .my-leave-active{  transition: all 0.8s ease;  }  </style>  </head>  <body>  <div id="app">  <input type="button" value="toggle" @click="flag=!flag">  <!--需求，点击按钮让h3显示，再点击，让h3隐藏 -->  <!-- 1.使用transition元素把需要被动画控制的元素包裹起来 -->  <!-- transition 元素是vue官方提供的 -->  <transition>  <h3 v-if="flag">这是一个h3</h3>  </transition>  <input type="button" value="toggle2" @click="flag2=!flag2">  <transition name="my">  <h6 v-if="flag2">这是一个h3</h6>  </transition>  </div>  <script>  var vm=new Vue({  el:'#app',  data:{  flag:false,  flag2:false,  },  methods:{}  })  </script>  </body>  </html> |

第三天8使用第三方animate.css类库实现动画

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link href="./lib/animate.css" rel="stylesheet">  <script src="./lib/vue.js"></script>  <!-- 需要基本类animated 入场bounceIn 离场bounceOut -->  </head>  <body>  <div id="app">  <input type="button" value="toggle" @click="flag=!flag">  <!--需求，点击按钮让h3显示，再点击，让h3隐藏 -->  <!-- <transition enter-active-class="animated bounceIn" leave-active-class="animated bounceOut">  <h3 v-if="flag">这是一个h3</h3>  </transition> -->  <!-- 使用：duration 来统一设置入场和离场的动画时长 -->  <!-- 入场离场时间 :duration="400" -->  <!-- <transition enter-active-class=" bounceIn" leave-active-class=" bounceOut" :duration="200">  <h3 v-if="flag" class="animated">这是一个h3</h3>  </transition> -->  <!-- 使用 :duration="{enter:200,leave:400}" 分别设置入场和离场的时长 -->  <transition  enter-active-class=" bounceIn"  leave-active-class=" bounceOut"  :duration="{enter:200,leave:400}">  <h3 v-if="flag" class="animated">这是一个h3</h3>  </transition>  </div>  <script>  var vm=new Vue({  el:'#app',  data:{  flag:false,  },  methods:{}  })  </script>  </body>  </html> |

第三天9 动画-钩子函数实现半动画的介绍

动画的生命周期函数

|  |
| --- |
| <transition  v-on:before-enter="beforeEnter"  v-on:enter="enter"  v-on:after-enter="afterEnter"  v-on:enter-cancelled="enterCancelled"   v-on:before-leave="beforeLeave"  v-on:leave="leave"  v-on:after-leave="afterLeave"  v-on:leave-cancelled="leaveCancelled" >  <!-- ... --> </transition> |

|  |
| --- |
| // ... methods: {  // --------  // 进入中  // --------   beforeEnter: function (el) {  // ...  },  // 当与 CSS 结合使用时  // 回调函数 done 是可选的  enter: function (el, done) {  // ...  done()  },  afterEnter: function (el) {  // ...  },  enterCancelled: function (el) {  // ...  },   // --------  // 离开时  // --------   beforeLeave: function (el) {  // ...  },  // 当与 CSS 结合使用时  // 回调函数 done 是可选的  leave: function (el, done) {  // ...  done()  },  afterLeave: function (el) {  // ...  },  // leaveCancelled 只用于 v-show 中  leaveCancelled: function (el) {  // ...  } } |

第三天10 动画-钩子函数实现小球半场动画

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  <style>  .ball{  width: 15px;  height: 15px;  border-radius: 50%;  background: red;  }  </style>  </head>  <body>  <div id="app">  <input type="button" value="快到我的碗里来" @click="flag=!flag">  <!-- 1.使用transition元素把小球包裹起来 -->  <!-- 2. -->  <transition  @before-enter="beforeEnter"  @enter="enter"  @after-enter="afterEnter"  >  <div class="ball" v-show="flag"></div>  </transition>    </div>  <script>  var vm=new Vue({  el:'#app',  data:{  flag:false,  },  methods:{  //注意：动画钩子函数的第一参数：el,表示要执行动画的那个dom元素是那个原生的js dom对象  // 可以以为，el是通过document.getElementById("")方式获取到的原生js dom对象  beforeEnter(el){  //beforeEnter 表示动画入场开始之前，此时，动画尚未开始，可以在beforeEnter中设置元素开始动画的起始样式  el.style.transform="translate(0,0)"  },  enter(el,done){  //这句话 没有实际的作用，但是如果不写出不来动画效果  //可以认为，el.offsetWidth会强制刷新  //el.offsetTop el.offsetLeft el.offsetHeigh都可以  el.offsetWidth  //表示动画开始之后的样式，这里可以设置小球完成后的 结束样式  el.style.transform="translate(150px,450px)"  el.style.transition="all 1s ease"  done()  //动画结束之后立即结束  //这里的done ，其实就是afterEnter这个函数，也就是说done是afterEnter函数的引用  },  afterEnter(el){  //动画完成之后会调用afterEnter  this.flag=!this.flag;  // done();  },  }  })  </script>  </body>  </html> |

第三天11动画-小球动画每次重新开始的位置说明

每次渲染动画重新执行一遍动画的生命周期，从而设置小球的位置为原始位置

第三天12动画-使用transition-group原始实现列表动画

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  <style>  li{  border: 1px dashed #999;  margin: 5px;  line-height: 35px;  padding-left: 12px;  }  li:hover{  background-color: hotpink;  transition: all 0.8s ease;  }  .v-enter,  .v-leave-to{  opacity: 0;  transform :translateY(80px)  }  .v-enter-active,  .v-leave-active{  transition: all 0.6s ease;  }  </style>  </head>  <body>  <div id="app">  <div>  <label for="">  Id:  <input type="text" v-model="id">  </label>  <label for="">  Name:  <input type="text" v-model="name">  </label>  <input type="button" value="添加" @click="add">  </div>  <ul>  <!-- 在实现列表过度的时候，如果需要过渡的元素，是通过v-for 循环渲染出来的，不能使用transition包裹，需要使用transitionGroup -->  <!-- 如果要为v-for 循环创建的元素设置动画，必须为每一个元素设置：key属性 第二定义两组类 -->  <transition-group>  <li v-for="item in list" :key="item.id">  {{item.id}}---{{item.name}}  </li>  </transition-group>  </ul>  </div>  <script>  var vm=new Vue({  el:'#app',  data:{  id:"",  name:"",  list:[  {id:1,name:"AAA"},  {id:2,name:"BBB"},  {id:3,name:"CCC"},  {id:4,name:"DDD"}  ]  },  methods:{  add(){  this.list.push({id:this.id,name:this.name})  this.id=this.name="";  },  }  })  </script>  </body>  </html> |

第三天13动画-实现列表删除和删除时候的动画效果

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  <style>  li{  border: 1px dashed #999;  margin: 5px;  line-height: 35px;  padding-left: 12px;  width: 100%;  }  li:hover{  background-color: hotpink;  transition: all 0.8s ease;  }  .v-enter,  .v-leave-to{  opacity: 0;  transform :translateY(80px)  }  .v-enter-active,  .v-leave-active{  transition: all 0.6s ease;  }  /\* 下面的 .v-move 和 .v-leave-active配合使用，能够使用列表后续元素,渐渐地飘上来的效果 \*/  .v-move{  transition: all 0.6s ease;  }  .v-leave-active{  position: absolute;  }  </style>  </head>  <body>  <div id="app">  <div>  <label for="">  Id:  <input type="text" v-model="id">  </label>  <label for="">  Name:  <input type="text" v-model="name">  </label>  <input type="button" value="添加" @click="add">  </div>  <ul>  <!-- 在实现列表过度的时候，如果需要过渡的元素，是通过v-for 循环渲染出来的，不能使用transition包裹，需要使用transitionGroup -->  <!-- 如果要为v-for 循环创建的元素设置动画，必须为每一个元素设置：key属性 第二定义两组类 -->  <transition-group>  <li v-for="(item , i) in list" :key="item.id" @click="del(i)">  {{item.id}}---{{item.name}}  </li>  </transition-group>  </ul>  </div>  <script>  var vm=new Vue({  el:'#app',  data:{  id:"",  name:"",  list:[  {id:1,name:"AAA"},  {id:2,name:"BBB"},  {id:3,name:"CCC"},  {id:4,name:"DDD"}  ]  },  methods:{  add(){  this.list.push({id:this.id,name:this.name})  this.id=this.name="";  },  del(i){  this.list.splice(i,1)  }  }  })  </script>  </body>  </html> |

第三天14动画-transition-group中apperar和tag属性的作用

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  <style>  li{  border: 1px dashed #999;  margin: 5px;  line-height: 35px;  padding-left: 12px;  width: 100%;  }  li:hover{  background-color: hotpink;  transition: all 0.8s ease;  }  .v-enter,  .v-leave-to{  opacity: 0;  transform :translateY(80px)  }  .v-enter-active,  .v-leave-active{  transition: all 0.6s ease;  }  /\* 下面的 .v-move 和 .v-leave-active配合使用，能够使用列表后续元素,渐渐地飘上来的效果 \*/  .v-move{  transition: all 0.6s ease;  }  .v-leave-active{  position: absolute;  }  </style>  </head>  <body>  <div id="app">  <div>  <label for="">  Id:  <input type="text" v-model="id">  </label>  <label for="">  Name:  <input type="text" v-model="name">  </label>  <input type="button" value="添加" @click="add">  </div>  <!-- <ul> -->  <!-- 在实现列表过度的时候，如果需要过渡的元素，是通过v-for 循环渲染出来的，不能使用transition包裹，需要使用transitionGroup -->  <!-- 如果要为v-for 循环创建的元素设置动画，必须为每一个元素设置：key属性 第二定义两组类 -->  <!-- 给transition-group 加上apperar属性实现页面刚展示出来入场时候的效果 -->  <!-- 通过为transition-group元素，设置tag属性，指定transition-group渲染为指定的元素，如果不指定tag属性，默认渲染为span标签 -->  <transition-group appear tag="ul">  <li v-for="(item , i) in list" :key="item.id" @click="del(i)">  {{item.id}}---{{item.name}}  </li>  </transition-group>  <!-- </ul> -->  </div>  <script>  var vm=new Vue({  el:'#app',  data:{  id:"",  name:"",  list:[  {id:1,name:"AAA"},  {id:2,name:"BBB"},  {id:3,name:"CCC"},  {id:4,name:"DDD"}  ]  },  methods:{  add(){  this.list.push({id:this.id,name:this.name})  this.id=this.name="";  },  del(i){  this.list.splice(i,1)  }  }  })  </script>  </body>  </html> |

15组件化和模块化的区别

定义vue组件

什么是组件？

组件的出现就是为了拆分vue实例的代码量，能够让我们以不同的组件，来划分不同的功能模块，将来我们需要什么样的功能，就可以调用对应的组件即可；

组件化和模块化的区别

----模块化：是从代码逻辑的角度进行划分的；方便代码分层开发，保证每个功能模块的职能单一；

----组件化：是从UI界面的角度进行划分的，前端的组件化，方便UI组件的重用

第三天16组件-创建组件的方式1

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <!-- 如果要使用组件，直接把组件的名称以HTML标签的形式引入进来 -->  <mycom1></mycom1>  </div>  <script>  //1.1使用Vue.extend来创建全局的vue组件  // var com1=Vue.extend({  // template:"<h3>这是使用Vue.extend创建的组件</h3>"//通过template属性，指定了组件要展示的HTML  // })  //1.2使用Vue.component("组件的名称"，创建出来的组件模板对象)  // Vue.component("myCom1",com1);  //如果使用Vue.component定义全局组件的时候，组件名称使用了驼峰命名，则在引用组件的时候，需要把大写的驼峰改为小写的字母，同时，两个单词之间使用 - 链接；  //如果不使用驼峰，则直接那名称来引用  // Vue.component("mycom1",com1);  // Vue.component 第一个参数：组件的名称，将来在引用组件的时候就是一个标签形式来引入它的  // 第二个参数：Vue.extend创建的组件，其中template就是组件将来要展示HTML内容  Vue.component("mycom1",Vue.extend({  template:"<h3>这是使用Vue.extend创建的组件</h3>"//通过template属性，指定了组件要展示的HTML  }));  var vm=new Vue({  el:'#app',  data:{    },  methods:{}  })  </script>  </body>  </html> |

第三天17组件-创建组件的方式2

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <!-- 还是使用标签形式 引入自己的组件 -->  <mycom2></mycom2>  </div>  <script>  Vue.component("mycom2",{  //注意不论是哪种方式创建出来的组件，组件的template属性指向的模板内容必须有且 只能有一个唯一的一个根元素  template:"<div><h3>这是直接使用Vue.component创建出来的组件</h3><span>123</span></div>"  })  var vm=new Vue({  el:'#app',  data:{    },  methods:{}  })  </script>  </body>  </html> |

第三天17组件-创建组件的方式3

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <mycom3></mycom3>  </div>  <!-- 在 被控制的#app使用template 元素，定义组件的HTML模板结构 -->  <template id="tmpl">  <div>  <h1>这是通过template元素，在外部定义的组件结构，这个方式，有代码的智能提示和高亮</h1>  <h4>好用，不错！ </h4>  </div>  </template>  <script>  Vue.component('mycom3',{  template:"#tmpl"  })  var vm=new Vue({  el:'#app',  data:{    },  methods:{}  })  </script>  </body>  </html> |

第三天19组件-使用components定义私有组件

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <mycom3></mycom3>  </div>  <div id="app2">  <mycom3></mycom3>  <login></login>  </div>  <!-- 在 被控制的#app使用template 元素，定义组件的HTML模板结构 -->  <template id="tmpl">  <div>  <h1>这是通过template元素，在外部定义的组件结构，这个方式，有代码的智能提示和高亮</h1>  <h4>好用，不错！ </h4>  </div>  </template>  <template id="templ2">  <h1>这是私有的login组件</h1>  </template>  <script>  Vue.component('mycom3',{  template:"#tmpl"  })  var vm=new Vue({  el:'#app',  data:{    },  methods:{}  })  var vm2=new Vue({  el:'#app2',  data:{    },  methods:{},  filters:{},  directives:{},  components:{//定义实例内部私有组件的  login:{  template:"#templ2"}  },  beforeCreate(){},  created(){},  beforeMount(){},  mounted(){},  beforeUpdate(){},  updated(){},  beforeDestrory(){},  destroyed(){},  })  </script>  </body>  </html> |

第三天20组件-组件中的data

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <mycom1></mycom1>  </div>  <script>  //1.组件可以有自己的data数据  //2.组件的data和实例的data有点不一样，实例中的data可以为一个对象，组件中的data必须是一个方法  //3.组件中的data除了必须 是一个方法之外，这个方法必须返回一个对象才行  //4.组件中的data数据使用方式和实例中的data使用方式一样  Vue.component("mycom1",{  template:"<h1>这是全局组件------{{msg}}</h1>",  data:function(){  return {  msg:"这是组件中data定义的数据"  }  }  })  var vm=new Vue({  el:'#app',  data:{    },  methods:{}  })  </script>  </body>  </html> |

第三天21组件-为什么组件中的data必须是一个function

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <counter></counter>  <hr>  <counter></counter>  <hr>  <counter></counter>  <hr>  </div>  <template id="templ">  <div>  <input type="button" value="+1" @click="increment">  <h3>{{count}}</h3>  </div>  </template>  <script>  //这是一个计数器的组件，身上有个按钮，每当点击按钮，让data中的count +1；  //var dataObj={count:0}  Vue.component('counter',{  template:'#templ',  data:function(){  return {count:0};  },  methods:{  increment(){  this.count++;  }  }  })  var vm=new Vue({  el:'#app',  data:{    },  methods:{}  })  </script>  </body>  </html> |

第三天22组件切换—使用v-if和v-else结合flag进行切换

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <a href="" @click.prevent="flag=true">登录</a>  <a href="" @click.prevent="flag=false">注册</a>  <login v-if="flag"></login>  <register v-else="flag"></register>  </div>  <script>  Vue.component('login',{  template:'<h3>登录组件</h3>'  })  Vue.component('register',{  template:"<h3>注册组件</h3>"  })  var vm=new Vue({  el:'#app',  data:{  flag:true,  },  methods:{}  })  </script>  </body>  </html> |

第三天23组件切换-使用Vue提供的component元素实现组件切换

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <a href="" @click.prevent="comName='login'">登录</a>  <a href="" @click.prevent="comName='register'">注册</a>  <!-- Vue提供了component ，来展示对应名称的组件 -->  <!-- component 是一个占位符 ，is属性，可以用来指定要展示的组件名称 -->  <component :is="comName"></component>  <!-- 总结：当前学习了几个vue提供的标签了？？？ -->  <!-- component template transition transition-group -->  </div>  <script>  //组件名称是字符串  Vue.component('login',{  template:'<h3>登录组件</h3>'  })  Vue.component('register',{  template:"<h3>注册组件</h3>"  })  var vm=new Vue({  el:'#app',  data:{  comName:'login'//当前component中的：is 绑定的组件名称  },  methods:{}  })  </script>  </body>  </html> |

第三天24组件切换-应用切换动画和mode方式

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <script src="./lib/vue.js"></script>  <style>  .v-enter,  .v-leave-to{  opacity: 0;  transform: translateX(150px)  }  .v-enter-active,  .v-leave-active{  transition: all 0.6s ease;  }  </style>  </head>  <body>  <div id="app">  <a href="" @click.prevent="comName='login'">登录</a>  <a href="" @click.prevent="comName='register'">注册</a>  <!-- Vue提供了component ，来展示对应名称的组件 -->  <!-- component 是一个占位符 ，is属性，可以用来指定要展示的组件名称 -->  <!-- 通过mode属性 设置组件切换时候的方式 -->  <transition mode="out-in">  <component :is="comName"></component>  </transition>  <!-- 总结：当前学习了几个vue提供的标签了？？？ -->  <!-- component template transition transition-group -->  </div>  <script>  //组件名称是字符串  Vue.component('login',{  template:'<h3>登录组件</h3>'  })  Vue.component('register',{  template:"<h3>注册组件</h3>"  })  var vm=new Vue({  el:'#app',  data:{  comName:'login'//当前component中的：is 绑定的组件名称  },  methods:{}  })  </script>  </body>  </html> |

第四天 01复习---实现小球动画

|  |
| --- |
| <!DOCTYPE html>  <html lang="">  <head>  <meta charset="utf-8">  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta name="viewport" content="width=device-width, initial-scale=1">  <title>Title Page</title>  <link rel="stylesheet" href="./lib/animate.css">  <script src="./lib/vue.js"></script>  <style>  .ball{  width: 15px;  height: 15px;  background-color:red;  border-radius: 50%;  }    </style>  </head>  <body>  <div id="app">  <input type="button" value="加入购物车" @click="flag=!flag">  <transition  @before-enter="beforeEnter"  @enter="enter"  @after-enter="afterEnter">  <div class="ball" v-show="flag">  </transition>  </div>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  flag:false,  },  methods:{  beforeEnter(el){  el.style.transform="translate(0,0)"  },  enter(el,done){  el.offsetWidth;  el.style.transform="translate(150px,450px)"  el.style.transition="all 1s ease"  done();  },  afterEnter(el){  //这句话，第一个功能，是控制小球的显示与隐藏  //第二个功能，直接跳过后半场动画，让flag标识符直接变为false  //当第二次再点击按钮的时候，flag false-->true  this.flag=!this.flag;  //el.style.opacity=0;  //vue 把一个完整的函数，使用钩子函数拆分为了两部分  //我们使用flag标识符，来表示动画的切换，  //刚开始，flag=false -->true true-->false  //  },  },  })  </script>  </body>  </html> |

第四天02组件---组件定义方式复习

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="./lib/animate.css">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <login></login>  </div>  <script>  //定义组件的时候，如果要定义全局组件，Vue.component('组件名称',{})  //通过对象字面量的形式，定义了一个组件模板对象  var login={  template:'<h1>1234</h1>'  }  //通过Vue.component把组件模板对象注册为一个全局Vue对象，同时，为这个组件起了一个名称，可以让我们通过标签形式，在页面中直接引入这个组件  //Vue.component('mylogin',login)  var vm=new Vue({  el:"#app",  data:{  flag:false,  },  methods:{},  components:{  //'组件的名称':"组件模板对象"  //'mylogin':login  login,  }})  </script>  </body>  </html> |

第四天03组件传值---父组件向子组件传值和data与props的区别

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="./lib/animate.css">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <!-- 父组件可以在引用子组件的时候通过属性绑定的形式(v-bind),把需要传递给子组件的数据，以属性绑定的形式传递到子组件内部，供子组件使用 -->  <com1 v-bind:parentmsg="msg"></com1 >  </div>  <script>  //定义组件的时候，如果要定义全局组件，Vue.component('组件名称',{})  //通过对象字面量的形式，定义了一个组件模板对象    //通过Vue.component把组件模板对象注册为一个全局Vue对象，同时，为这个组件起了一个名称，可以让我们通过标签形式，在页面中直接引入这个组件  //Vue.component('mylogin',login)  var vm=new Vue({  el:"#app",  data:{  msg:'123--父组件中的数据',  },  methods:{},  components:{  //结论：经过演示，发现，子组件中，默认无法访问到父组件上data上的数据和methods中的方法  com1:{  //注意：子组件中的data数据，并不是通过父组件传递过来的而是子组件自身私有的，比如，子组件通过ajax请求回来的数据都可以放到data身上  //data上的数据都是可读可写的  data(){  return{  title:'123',  content:'aaa',  }  },  //注意；组件中的所有props中的数据，都是父组件传递给子组件的  template:'<h1 @click="change">这是子组件---{{parentmsg}}</h1>',  //把父组件传递过来的属性，先在props数组中定义，这样才能使用这个数据  //props中的数据都是只读的，无法重现赋值  props:['parentmsg',],  methods:{  change(){  this.parentmsg="被修改了"  }  }  }  }})  </script>  </body>  </html> |

第四天04组件传值---子组件通过事件向父组件传值

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <!-- 父组件向子组件传递方法 使用的是事件绑定机制 v-on @ 当我们自定义了一个事件属性之后，那么，子组件就能够，通过某种方式，来调用传递进去的这个方法-->  <com2 @func123="show"></com2>  </div>  <template id="templ">  <div>    <h1>这是子组件</h1>  <input type="button" value="这是子组件中的按钮，点击它，触发父组件传递过来的fun方法" @click="myclick">  </div>  </template>  <script>  //定义了一个字面量类型的组件模板对象  var com2={  template:"#templ",//通过指定了一个Id,表示要加载这个指定Id的template元素中的内容，当作组件的HTML结构  data(){  return {  sonmes:{  name:"小头儿子",  age:6  }  }  },  methods:{  myclick(){  //当点击子组件的按钮的时候，如何拿到父组件传递过来的fun方法，并且调用它呢？？？  //emit 英文原意是触发调用的意思  this.$emit('func123',this.sonmes)  }  }  }  var vm=new Vue({  el:"#app",  data:{  datamstFromSon:null,  },  methods:{  show(data){  console.log(data)  this.datamstFromSon=data;  }  },  components:{  com2,    }  })  </script>  </body>  </html> |

第四天05组件案例---发表评论功能的实现

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <link rel="stylesheet" href="./lib/bootstrap/bootstrap.css">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <cmt-box></cmt-box>  <ul class="list-group">  <li class="list-group-item" v-for="item in list" :key="item.id">  <span class="badge">评论人：{{item.user}}</span>  {{item.content}}  </li>  </ul>    </div>  <template id="templ">  <div>  <div class="form-group">  <label for="">评论人</label>  <input type="text" class="form-control" v-model="user">  </div>  <div class="form-group">  <label for="">评论内容</label>  <textarea class="form-control" name="" id="" cols="30" rows="10" v-model="content"></textarea>  </div>  <div class="form-group">  <input type="button" @click="postComment" value="发表评论" class="btn btn-primary">  </div>  </div>  </template>  <script>  var commentBox={  template:'#templ',  data(){  return{  user:"",  content:""  }  },  methods:{  postComment(){//发表评论的方法  //分析：发表评论的业务逻辑  //1.评论数据存到哪里去？？？存放到了localstorage中 localstorage.setItem('','')  //2.先组织输入一个最新的评论数据对象  //3.想办法，把第二步中得到的评论对象保存到localstorage中  //3.1loclstorage只支持存放字符串数据，要先调用JSON.stringify  //3.2在保存最新的评论数据之前，要先localstorage中获取到之前的数据 string ,转换为一个数组对象，然后把最新的评论push到这个数组中去 JSON.parse  //3.3如果获取到的localstorage中的评论字符串为空或不存在则可以返回一个[],让JSON.parse去转换  //3.4把最新的评论列表数组，再次调用JSON.stringify,转为字符串，然后调用localstorage.setItem()  var comment={id:Date.now(),user:this.user,content:this.content}  var list=JSON.parse(localStorage.getItem('cmts')||'[]');  //从localstorage 中获取所有的评论  list.unshift(comment)  //重新保存最新的评论数据  localStorage.setItem('cmts',JSON.stringify(list))  this.user=this.content=""  }  }    }  var vm=new Vue({  el:"#app",  data:{  list:[  {id:Date.now(),user:'李白',content:"天生我才必有用"},  {id:Date.now(),user:'江小白',content:"劝君更尽一杯酒"},  {id:Date.now(),user:'小马',content:"我姓马，风吹草地见牛羊的马"},  ]  },  methods:{},  components:{  'cmt-box':commentBox,  }  })  </script>  </body>  </html> |

第四天06组件案例---实现评论的发表和自动刷新

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <link rel="stylesheet" href="./lib/bootstrap/bootstrap.css">  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <cmt-box @func="loadComments"></cmt-box>  <ul class="list-group">  <li class="list-group-item" v-for="item in list" :key="item.id">  <span class="badge">评论人：{{item.user}}</span>  {{item.content}}  </li>  </ul>    </div>  <template id="templ">  <div>  <div class="form-group">  <label for="">评论人</label>  <input type="text" class="form-control" v-model="user">  </div>  <div class="form-group">  <label for="">评论内容</label>  <textarea class="form-control" name="" id="" v-model="content"></textarea>  </div>  <div class="form-group">  <input type="button" @click="postComment" value="发表评论" class="btn btn-primary">  </div>  </div>  </template>  <script>  var commentBox={  template:'#templ',  data(){  return{  user:"",  content:""  }  },  methods:{  postComment(){//发表评论的方法  //分析：发表评论的业务逻辑  //1.评论数据存到哪里去？？？存放到了localstorage中 localstorage.setItem('','')  //2.先组织输入一个最新的评论数据对象  //3.想办法，把第二步中得到的评论对象保存到localstorage中  //3.1loclstorage只支持存放字符串数据，要先调用JSON.stringify  //3.2在保存最新的评论数据之前，要先localstorage中获取到之前的数据 string ,转换为一个数组对象，然后把最新的评论push到这个数组中去 JSON.parse  //3.3如果获取到的localstorage中的评论字符串为空或不存在则可以返回一个[],让JSON.parse去转换  //3.4把最新的评论列表数组，再次调用JSON.stringify,转为字符串，然后调用localstorage.setItem()  var comment={id:Date.now(),user:this.user,content:this.content}  var list=JSON.parse(localStorage.getItem('cmts')||'[]');  //从localstorage 中获取所有的评论  list.unshift(comment)  //重新保存最新的评论数据  localStorage.setItem('cmts',JSON.stringify(list))  this.user=this.content=""  this.$emit('func')    }  }    }  var vm=new Vue({  el:"#app",  data:{  list:[  {id:Date.now(),user:'李白',content:"天生我才必有用"},  {id:Date.now(),user:'江小白',content:"劝君更尽一杯酒"},  {id:Date.now(),user:'小马',content:"我姓马，风吹草地见牛羊的马"},  ]  },  beforeCreate(){  //注意这里不能 调用loadComments方法，因为在执行这个钩子函数的时候data和methods都还没有被初始化好  },  created(){  this.loadComments();  },  methods:{  loadComments(){//从本地localstorage中 ，加载评论列表  var list=JSON.parse(localStorage.getItem('cmts')||'[]');  this.list=list;  }  },  components:{  'cmt-box':commentBox,  }  })  </script>  </body>  </html> |

第四天07使用ref获取dom元素和组件引用

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <input ref="mybtn" type="button" value="获取元素" @click="getElement">  <h3 ref="myh3">哈哈哈，今天天气太好了</h3>  <hr>  <login ref="mylogin"></login>  </div>  <script>  var login={  template:"<h1>登录组件</h1>"  ,data(){  return {  msg:"son msg"  }  },  methods:{  show(){  console.log("调用了子组件的方法")  }  }  }  var vm=new Vue({  el:"#app",  data:{},  methods:{  getElement(){  //ref 是英文单词reference 的简写 【值类型 引用类型】【reference err】  //  // console.log(this.$refs.myh3.innerText)  // console.log(this.$refs.mylogin.msg)  this.$refs.mylogin.show();  }  },  components:{  login,  }  })  </script>  </body>  </html> |

第四天08路由---前端路由和后端路由的概念

什么是路由

后端路由 对于普通的网站，所有的超链接都是url地址，所有的URL地址都对应服务器上对应的资源

前端路由 对于单页面应用程序来说，主要通过URL中的hash（#）来实现的不同页面之间的切换同时hash还有一个特点：http请求中不会包含hash相关的内容，所有，单页面程序中的页面跳转主要用hash实现；

在单页面应用程序中，这种通过hash改变来实现切换页面的方式，称作前端路由；

第四天09路由---安装vue-router两种方式

第四天10路由---vue-router的基本使用

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  <!-- 1.安装vue-router 路由模块 -->  <script src="./lib/vue-router.js"></script>  </head>  <body>  <div id="app">  <a href="#/login">登录</a>  <a href="#/register">注册</a>  <!-- 这是vue-router提供的元素专门用来当做占位符的将来路由规则匹配到的组件就会展示到这个router-view中去  所以我们可以把router-view当做一个占位符 -->  <router-view></router-view>  </div>  <script>  //组件的模板对象  var login={  template:"<h1>登录组件</h1>"  }  var register={  template:"<h1>注册组件</h1>"  }  // Vue.component('login',{  // template:"<h1>登录组件</h1>"  // })  //2.创建一个路由对象，当导入vue-router包之后，在Windows全局对象中，就有了一个路由的构造函数，叫做vueRouter，  //在new 路由对象的时候，可以为构造函数传递一个配置对象  var routerObj=new VueRouter({  //route//这个配置对象表示【路由配置规则】的意思  routes:[//路由匹配规则  //每个路由规则，都是一个对象，这个规则对象，身上都有两个必须的属性  //属性1是path 表示监听那个路由连接地址  //属性2是component 表示如果路由是前面匹配到的path，则展示component属性对应的那个组件  //注意：component 的属性值，必须是一个组件的模板对象，不能是组件的引用名称  {path:'/login',component:login},  {path:'/register',component:register}  ]  });  var vm=new Vue({  el:"#app",  data:{},  methods:{},  router:routerObj,//将路由规则对象注册到vm实例上，用来监听URL地址的变化，然后展示对应的组件  })  </script>  </body>  </html> |

第四天11路由---router-link的使用

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  <!-- 1.安装vue-router 路由模块 -->  <script src="./lib/vue-router.js"></script>  </head>  <body>  <div id="app">  <!-- <a href="#/login">登录</a>  <a href="#/register">注册</a> -->  <!-- router-link 默认渲染a标签 -->  <router-link tag="span" to="/login">登录</router-link>  <router-link to="/register" >注册</router-link>  <!-- 这是vue-router提供的元素专门用来当做占位符的将来路由规则匹配到的组件就会展示到这个router-view中去  所以我们可以把router-view当做一个占位符 -->  <router-view></router-view>  </div>  <script>  //组件的模板对象  var login={  template:"<h1>登录组件</h1>"  }  var register={  template:"<h1>注册组件</h1>"  }  // Vue.component('login',{  // template:"<h1>登录组件</h1>"  // })  //2.创建一个路由对象，当导入vue-router包之后，在Windows全局对象中，就有了一个路由的构造函数，叫做vueRouter，  //在new 路由对象的时候，可以为构造函数传递一个配置对象  var routerObj=new VueRouter({  //route//这个配置对象表示【路由配置规则】的意思  routes:[//路由匹配规则  //每个路由规则，都是一个对象，这个规则对象，身上都有两个必须的属性  //属性1是path 表示监听那个路由连接地址  //属性2是component 表示如果路由是前面匹配到的path，则展示component属性对应的那个组件  //注意：component 的属性值，必须是一个组件的模板对象，不能是组件的引用名称  {path:'/login',component:login},  {path:'/register',component:register}  ]  });  var vm=new Vue({  el:"#app",  data:{},  methods:{},  router:routerObj,//将路由规则对象注册到vm实例上，用来监听URL地址的变化，然后展示对应的组件  })  </script>  </body>  </html> |

第四天12路由---redirect重定向的使用

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  <!-- 1.安装vue-router 路由模块 -->  <script src="./lib/vue-router.js"></script>  </head>  <body>  <div id="app">  <!-- <a href="#/login">登录</a>  <a href="#/register">注册</a> -->  <!-- router-link 默认渲染a标签 -->  <router-link tag="span" to="/login">登录</router-link>  <router-link to="/register" >注册</router-link>  <!-- 这是vue-router提供的元素专门用来当做占位符的将来路由规则匹配到的组件就会展示到这个router-view中去  所以我们可以把router-view当做一个占位符 -->  <router-view></router-view>  </div>  <script>  //组件的模板对象  var login={  template:"<h1>登录组件</h1>"  }  var register={  template:"<h1>注册组件</h1>"  }  // Vue.component('login',{  // template:"<h1>登录组件</h1>"  // })  //2.创建一个路由对象，当导入vue-router包之后，在Windows全局对象中，就有了一个路由的构造函数，叫做vueRouter，  //在new 路由对象的时候，可以为构造函数传递一个配置对象  var routerObj=new VueRouter({  //route//这个配置对象表示【路由配置规则】的意思  routes:[//路由匹配规则  //每个路由规则，都是一个对象，这个规则对象，身上都有两个必须的属性  //属性1是path 表示监听那个路由连接地址  //属性2是component 表示如果路由是前面匹配到的path，则展示component属性对应的那个组件  //注意：component 的属性值，必须是一个组件的模板对象，不能是组件的引用名称  // {path:'/',component:login},  {path:'/',redirect:'/login'},//这里的redirect 和node 里面的redirect完全是两码事  {path:'/login',component:login},  {path:'/register',component:register}  ]  });  var vm=new Vue({  el:"#app",  data:{},  methods:{},  router:routerObj,//将路由规则对象注册到vm实例上，用来监听URL地址的变化，然后展示对应的组件  })  </script>  </body>  </html> |

第四天13路由---设置选中路由高亮的两种方式

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  <!-- 1.安装vue-router 路由模块 -->  <script src="./lib/vue-router.js"></script>  <style>    .router-link-active,.myactive{  color: red;  font-weight: 800;  font-style: italic;  font-size: 80px;  text-decoration: underline;  background-color: green;  }  </style>  </head>  <body>  <div id="app">  <!-- <a href="#/login">登录</a>  <a href="#/register">注册</a> -->  <!-- router-link 默认渲染a标签 -->  <router-link tag="span" to="/login">登录</router-link>  <router-link to="/register" >注册</router-link>  <!-- 这是vue-router提供的元素专门用来当做占位符的将来路由规则匹配到的组件就会展示到这个router-view中去  所以我们可以把router-view当做一个占位符 -->  <router-view></router-view>  </div>  <script>  //组件的模板对象  var login={  template:"<h1>登录组件</h1>"  }  var register={  template:"<h1>注册组件</h1>"  }  // Vue.component('login',{  // template:"<h1>登录组件</h1>"  // })  //2.创建一个路由对象，当导入vue-router包之后，在Windows全局对象中，就有了一个路由的构造函数，叫做vueRouter，  //在new 路由对象的时候，可以为构造函数传递一个配置对象  var routerObj=new VueRouter({  //route//这个配置对象表示【路由配置规则】的意思  routes:[//路由匹配规则  //每个路由规则，都是一个对象，这个规则对象，身上都有两个必须的属性  //属性1是path 表示监听那个路由连接地址  //属性2是component 表示如果路由是前面匹配到的path，则展示component属性对应的那个组件  //注意：component 的属性值，必须是一个组件的模板对象，不能是组件的引用名称  // {path:'/',component:login},  {path:'/',redirect:'/login'},//这里的redirect 和node 里面的redirect完全是两码事  {path:'/login',component:login},  {path:'/register',component:register}  ],  linkActiveClass:'myactive',  });  var vm=new Vue({  el:"#app",  data:{},  methods:{},  router:routerObj,//将路由规则对象注册到vm实例上，用来监听URL地址的变化，然后展示对应的组件  })  </script>  </body>  </html> |

第四天14路由---为路由切换启动动画

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  <!-- 1.安装vue-router 路由模块 -->  <script src="./lib/vue-router.js"></script>  <style>    .router-link-active,.myactive{  color: red;  font-weight: 800;  font-style: italic;  font-size: 80px;  text-decoration: underline;  background-color: green;  }  .v-enter,  .v-leave-to{  opacity: 0;  transform: translateX(140px)  }  .v-enter-active,  .v-leave-active{  transition: all 0.5s ease;  }  </style>  </head>  <body>  <div id="app">  <!-- <a href="#/login">登录</a>  <a href="#/register">注册</a> -->  <!-- router-link 默认渲染a标签 -->  <router-link tag="span" to="/login">登录</router-link>  <router-link to="/register" >注册</router-link>  <!-- 这是vue-router提供的元素专门用来当做占位符的将来路由规则匹配到的组件就会展示到这个router-view中去  所以我们可以把router-view当做一个占位符 -->  <transition mode="out-in">  <router-view></router-view>  </transition>  </div>  <script>  //组件的模板对象  var login={  template:"<h1>登录组件</h1>"  }  var register={  template:"<h1>注册组件</h1>"  }  // Vue.component('login',{  // template:"<h1>登录组件</h1>"  // })  //2.创建一个路由对象，当导入vue-router包之后，在Windows全局对象中，就有了一个路由的构造函数，叫做vueRouter，  //在new 路由对象的时候，可以为构造函数传递一个配置对象  var routerObj=new VueRouter({  //route//这个配置对象表示【路由配置规则】的意思  routes:[//路由匹配规则  //每个路由规则，都是一个对象，这个规则对象，身上都有两个必须的属性  //属性1是path 表示监听那个路由连接地址  //属性2是component 表示如果路由是前面匹配到的path，则展示component属性对应的那个组件  //注意：component 的属性值，必须是一个组件的模板对象，不能是组件的引用名称  // {path:'/',component:login},  {path:'/',redirect:'/login'},//这里的redirect 和node 里面的redirect完全是两码事  {path:'/login',component:login},  {path:'/register',component:register}  ],  linkActiveClass:'myactive',  });  var vm=new Vue({  el:"#app",  data:{},  methods:{},  router:routerObj,//将路由规则对象注册到vm实例上，用来监听URL地址的变化，然后展示对应的组件  })  </script>  </body>  </html> |

第四天15路由参数---使用query方式传递参数

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  <script src="./lib/vue-router.js"></script>  </head>  <body>  <div id="app">  <!-- 如果在路由中使用 查询字符串，给路由传递参数，则不需要修改路由规则 的path属性 -->  <router-link to="/login?id=10&name=zs">登录</router-link>  <router-link to="/register">注册</router-link>  <router-view></router-view>  </div>  <script>  var login={  template:"<h1>登录---{{$route.query.id}}---{{$route.query.name}}</h1>",  data(){  return{  msg:'123'  }  },  created(){  //组件的生命周期钩子函数  console.log(this.$route.query.id)  }  }  var register={  template:"<h1>注册</h1>",  }  var router=new VueRouter({  routes:[  {path:'/login',component:login},  {path:'/register',component:register}  ]  })  var vm=new Vue({  el:"#app",  data:{},  methods:{},  router,  })  </script>  </body>  </html> |

第四天16路由传参---使用params方式传递路由参数

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  <script src="./lib/vue-router.js"></script>  </head>  <body>  <div id="app">  <!-- 如果在路由中使用 查询字符串，给路由传递参数，则不需要修改路由规则 的path属性 -->  <router-link to="/login/10/l4">登录</router-link>  <router-link to="/register">注册</router-link>  <router-view></router-view>  </div>  <script>  var login={  template:"<h1>登录---{{$route.params.id}}---{{$route.params.name}}</h1>",  data(){  return{  msg:'123'  }  },  created(){  //组件的生命周期钩子函数  console.log(this.$route.params.id)  }  }  var register={  template:"<h1>注册</h1>",  }  var router=new VueRouter({  routes:[  {path:'/login/:id/:name',component:login},  {path:'/register',component:register}  ]  })  var vm=new Vue({  el:"#app",  data:{},  methods:{},  router,  })  </script>  </body>  </html> |

第四天17路由---使用children属性实现路由嵌套

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  <script src="./lib/vue-router.js"></script>  </head>  <body>  <div id="app">  <router-link to="/account">Account</router-link>  <router-view></router-view>  </div>  <template id="templ">  <div>  <h1>这是account组件</h1>  <router-link to="/account/login">登录</router-link>  <router-link to="/account/register">注册</router-link>  <router-view></router-view>  </div>  </template>  <script>  var account={  template:'#templ',  }  var login={  template:'<h3>登录</h3>'  }  var register={  template:"<h3>注册</h3>"  }  var router=new VueRouter({  routes:[  {path:'/account',  component:account,  // 使用children属性，实现了子路由，子路由的path前面，不加 带/否则永远以根路径开始请求，这样不方便我们用户理解URL理解请求地址  children:[  {path:'login',component:login},  {path:'register',component:register}  ]},  // {path:'/account/login',component:login},  // {path:'/account/register',component:register}  ]  })  var vm=new Vue({  el:"#app",  data:{},  methods:{},  router,  })  </script>  </body>  </html> |

第四天18 路由—使用命名视图实现经典布局

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  <script src="./lib/vue-router.js"></script>  <style>  html,body{  margin: 0;  padding:0;  }  .header{  background-color: #8eb2fc;  height: 80px;  }  .container{  display: flex;  height: 400px;  }  h1{  margin: 0;  padding: 0;  font-size:16px;  }  .left{  background-color: lightgreen;  flex: 2  }  .main{  background-color: lightpink;  flex: 8  }  </style>  </head>  <body>  <div id="app">  <router-view></router-view>    <div class="container">  <router-view name="left"></router-view>  <router-view name="main"></router-view>  </div>    </div>  <script>  var header={  template:"<h1 class='header'>Header头部区域</h1>"  }  var leftBox={  template:"<h1 class='left'>Left侧边栏区域</h1>"  }  var mainBox={  template:"<h1 class='main'>mainBox主体区域</h1>"  }  var router=new VueRouter({  routes:[  // {path:"/",component:header},  // {path:'/left',component:leftBox},  // {path:'/main',component:mainBox}  {path:'/',components:{  'default':header,  'left':leftBox,  'main':mainBox,  }}  ]  })  var vm=new Vue({  el:"#app",  data:{},  methods:{},  router  })  </script>  </body>  </html> |

第四天19路由---关于命名视图的名称到底是变量还是字符串的说明

字符串

第五天01复习---父子组件之间传值

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">    <com1 @func="getMsgFromSon" v-bind:parentmsg="msg"></com1>  </div>  <template id="templ">  <div>  <input type="button" value="向父组件传递消息" @click="sendMsg">  <h1>这是子元素---{{parentmsg}}</h1>  </div>  </template>  <script>  var com1={  template:"#templ",  data(){  return {  msg:"做一个孝顺的孩子，给爸爸一些钱去挥霍吧"  }  },  props:['parentmsg'],  methods:{  sendMsg(){  this.$emit('func',this.msg)  }  }    }  var vm=new Vue({  el:"#app",  data:{  msg:'这是父组件中的数据，爸爸有100块钱，my son ,你要不？',  msgFromSon:"",  },  methods:{  getMsgFromSon(data){  this.msgFromSon=data;  console.log(this.msgFromSon)  }  },  components:{  com1  }  })  </script>  </body>  </html> |

第五天02路由---基本的使用

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  <!-- 1.导入包 -->  <script src="./lib/vue-router.js"></script>  </head>  <body>  <div id="app">  <router-link to="/login">登录</router-link>  <router-link to="/register">注册</router-link>  <router-view></router-view>  </div>  <script>  // 2.创建子组件  var login={  template:"<h3>这是登录组件，这个组件是奔波霸开发的</h3>"  }  var register={  template:"<h3>这是注册组件，这个组件是巴博泵开发的</h3>"  }  //3.创建路由对象  var router=new VueRouter({  routes:[//路由规则数组  {path:'/',redirect:"/login"},  {path:'/login',component:login},  {path:'/register',component:register}  ],  linkActiveClass:"myactive"//和激活相关的类  })  var vm=new Vue({  el:"#app",  data:{},  methods:{},  router,  })  </script>  </body>  </html> |

第五天03名称案例---使用keyup事件实现

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <!-- 1.我们要监听到文本框数据的改变，这样才能知道什么去拼接出一个fullname -->  <!-- 2.如何监听到文本框的数据改变呢？？？ -->  <!-- 3. -->  <input type="text" v-model="firstname" @keyup="getFullname">  <input type="text" v-model="lastname" @keyup="getFullname">  <input type="text" v-model="fullname">  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  firstname:"",  lastname:"",  fullname:"",  },  methods:{  getFullname(){  this.fullname=this.firstname+"-"+this.lastname;  }  },  })  </script>  </body>  </html> |

第五天04名称案例---使用watch监听文本框数据的变化

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">    <input type="text" v-model="firstname" >  <input type="text" v-model="lastname" >  <input type="text" v-model="fullname">  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  firstname:"",  lastname:"",  fullname:"",  },  methods:{    },  watch:{  //使用这个属性，可以监听data中指定的数据变化，然后出发这个watch中对应的function处理函数  firstname:function(newVal,oldVal){  // this.fullname=this.firstname+"---"+this.lastname;  this.fullname=newVal+"---"+this.lastname  },  lastname:function(newVal){  this.fullname=this.firstname+"---"+newVal;  },  }  })  </script>  </body>  </html> |

第五天05watch---监视路由地址的改变

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  <!-- 1.导入包 -->  <script src="./lib/vue-router.js"></script>  </head>  <body>  <div id="app">  <router-link to="/login">登录</router-link>  <router-link to="/register">注册</router-link>  <router-view></router-view>  </div>  <script>  // 2.创建子组件  var login={  template:"<h3>这是登录组件，这个组件是奔波霸开发的</h3>"  }  var register={  template:"<h3>这是注册组件，这个组件是巴博泵开发的</h3>"  }  //3.创建路由对象  var router=new VueRouter({  routes:[//路由规则数组  {path:'/',redirect:"/login"},  {path:'/login',component:login},  {path:'/register',component:register}  ],  linkActiveClass:"myactive"//和激活相关的类  })  var vm=new Vue({  el:"#app",  data:{},  methods:{},  router,  watch:{  //this.$route.path  '$route.path':function(newVal,oleVal){  // console.log(newVal+"---"+oleVal)  if (newVal=="/login") {  console.log("AAAA")  } else {  console.log("BBBB")  }  }  }  })  </script>  </body>  </html> |

第五天06computed---计算属性的使用和3个特点

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Page Title</title>  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" type="text/css" media="screen" href="./lib/animate.css" />  <script src="./lib/vue.js"></script>  </head>  <body>  <div id="app">  <input type="text" v-model="firstname" > +  <input type="text" v-model="middlename" >+  <input type="text" v-model="lastname" > =  <input type="text" v-model="fullname">  <p>{{fullname}}</p>  <p>{{fullname}}</p>  <p>{{fullname}}</p>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  firstname:"",  lastname:"",  middlename:"",    },  methods:{},  computed:{  //在computed中，可以定义一些属性，  //这些属性叫做计算属性，本质就是一个方法，  //只不过，我们在使用这些计算属性的时候 ，  //是把他们的名称直接当做属性来使用的，并  //不会把计算属性当做方法来调用  //注意1；计算属性在引用的时候一定不要加()去调用，直接把它当做普通属性去使用就好了  //注意2：只要计算属性这个function内部所用到的任何data中的数据发生了变化，就会立即重新计算这个计算属性的值  //注意3：计算属性的求值结果会被缓存起来，方便下次使用，如果计算属性方法中所依赖的任何数据都没有发生过变化，则不会重新对计算属性进行求值  'fullname':function(){  console.log("ok")  return this.firstname+"-"+this.middlename+"-"+this.lastname  }  },    })  </script>  </body>  </html> |

第五天07watch、computed、methods的对比

Watch.computed 都是函数

Computed 一定要return

Watch 不需要return

Methods 适合写大量业务逻辑

总结

1.’computed’属性的结果会被缓存，除非依赖的响应式属性变化才会重新计算。注意当做属性来使用。

2.’methods’方法表示一个具体的操作，注意书写业务逻辑

3.’watch’一个对象，键需要观察的表达式，值是对应回调函数，主要用来监听某些特定数据的变化，从而进行某些具体的业务逻辑操作，可以看做是’computed’和’methods’的结合体

第五天08nrm的使用

1运行npm I nrm -g全局安装‘nrm’包

2.使用nrm list 查看当前所有可用的镜像源地址以及当前所使用的镜像源地址

3.使用nrm use npm或nrm use taobao切换不同的镜像源地址

注意：nrm只是单纯的提供了几个常用的下载包的URL地址，并能够让我们在这几个地址之

间很方便的进行切换，但是，我们每次转包的时候使用的转包工具都是npm

npm i cnpm –g

镜像地址，

第五天09webpack概念的引入

在网页中会引用哪些常见的静态资源？

+Js

-.js .jsx .coffee .ts(typescript 类c#语言)

Css

-.css .less .sass .scss(sass已更新为scss)用法基本和less一样

Images

-jpg .png .gif .bmp .svg

字体文件

-.svg .ttf .ect .woff .woff2

模板文件

-.ejs .jade .vue【这是在webpack中定义组件的方式，推荐这么用】

在网页中引入的静态资源多了以后有什么问题？

1. 网页加载速度慢，因为我们要发起很多二次请求；
2. 要处理错综复杂的依赖关系

如何解决上述两个问题？

1. 合并、压缩、精灵图、图片的base64编码【小图片转为base64】
2. 可以使用之前学过的requireJS 也可以使用webpack 可以解决各个包之间的复杂依赖关系

什么是webpack？

Webpack 是前端的一个项目构建工具，他是基于node.js开发出来的一个前端工具

如何玩么实现上述的2种解决方案？

1. 使用gulp，是基于task任务的
2. 使用webpack 是基于整个项目进行构建的

+借助于webpack 这个前端自主化构建工具，可以完美实现资源的合并、打包、压缩混淆等诸多功能

+根据官网的图片介绍webpack 打包的过程

-webpack 官网：http:webpack.github.io

第五天10webpack---基本的使用方式

安装

1. 运行npm I webpack –g 全局安装webpack 这样就能在全局使用webpack 的命令

2. 在项目根目录中运行npm I webpack –save-dev安装到项目依赖中

初步使用

Dist

Src

Ul>li\*10{这是第$个里}

Npm init –y

Npm I juery -S

Npm I webpack-cli

第五天1

第五天1

第五天1

第五天1

第五天1

第五天1

第五天1

第五天1

第五天1

第五天1

第五天1