准备: 微信小程序: WEB问题速查 小恐龙图标

1. 历届师兄师姐在今天课程中曾经问过的高频的经典的问题清单和解答: 小程序->首页->VUE->day02、day03

自己可能没有发现问题，但是想知道别的同学都问过哪些问题

2. 今日课程对应的重点案例的视频以及鄙视面试所需的扩展知识视频：小程序->在线->VUE-> day02、03

(学有余力!!!的同学或自己认为自己基础好的同学必须看!!!!!!)：

今日扩展视频：

小程序->在线->VUE->day02

8. 绑定样式... ... 终极简写（基础一般的同学不用看！）

10. 计算属性...... 作业: vue购物车，仅包含客户端（基础一般的同学不用看！）

小程序->在线->VUE->day03

6. Vue组件化开发todo实现封装组件... ...删除（基础一般的同学不用看！）

7. 组件间传参 todo 父给子，子给父，兄弟间 ref $ref（基础一般的同学不用看！）

模块化开发（7个视频）（基础一般的同学不用看！）

3. 今日课程中包含的英语单词的翻译：小程序->单词->VUE->day02、day03

英文不太好的同学，可以提前看一下第二天需要用到哪些英文单词。

约定: 2件事:

1. 第三阶段禁止上课跟着敲笔记和敲代码！谁抄笔记抄的勤，谁抄代码抄的勤，谁学的越烂！

2. 每个完整功能的案例/视频，至少写三遍: (如果需要画图，必须亲手画图)

(1). 连代码+注释抄一遍 —— 明白这个例子是要做什么事儿

(2). 保留注释，自己试着看着注释，将代码填回来 —— 复习函数和对象

(3). 删除所有代码和注释，自己试着用自己的话，把注释写回来，再翻译为代码！—— 只有自己可以写出人话的注释，这个例子才算自己会！

如果没有问题的同学: （享受vip一对一答疑的时间还剩3+3天，端午节不休息）

1. 上课时那句话没听懂，笔记中那句话看不懂，案例中那句话看不懂，立刻问！

2. 小程序问题清单中哪个问题或解答看不懂，立刻问！

3. 小程序视频中哪个视频看不懂，立刻问

4. 个人项目中哪个功能不会做，立刻问

5. 平时刷笔试题时，遇到不会的笔试题，立刻问

正课:

1. 绑定样式:

2. 自定义指令

3. 计算属性

4. 过滤器

5. axios

6. \*\*\*\*\*vue生命周期

一. 绑定样式：

对应小程序视频: 小程序->在线->VUE->day02 8. 绑定样式 ...

1. 绑定内联样式style属性: (已讲过)

何时: 希望精确修改一个css属性的值时

2. 绑定class属性:

(1). 何时: 批量修改多个css属性值时

(2). 不好的方式: 将整个class看做一个普通的字符串属性绑定

缺点: 极其不便于修改其中某一个class

(3). 好的方式: 2种:

a. 对每个可能动态变化的class都执行一个变量

临时创建一个匿名对象的意思

绑定 ↓

1). <元素 :class="{ class名1: bool表达式或变量, class名2: bool表达式或变量 }"

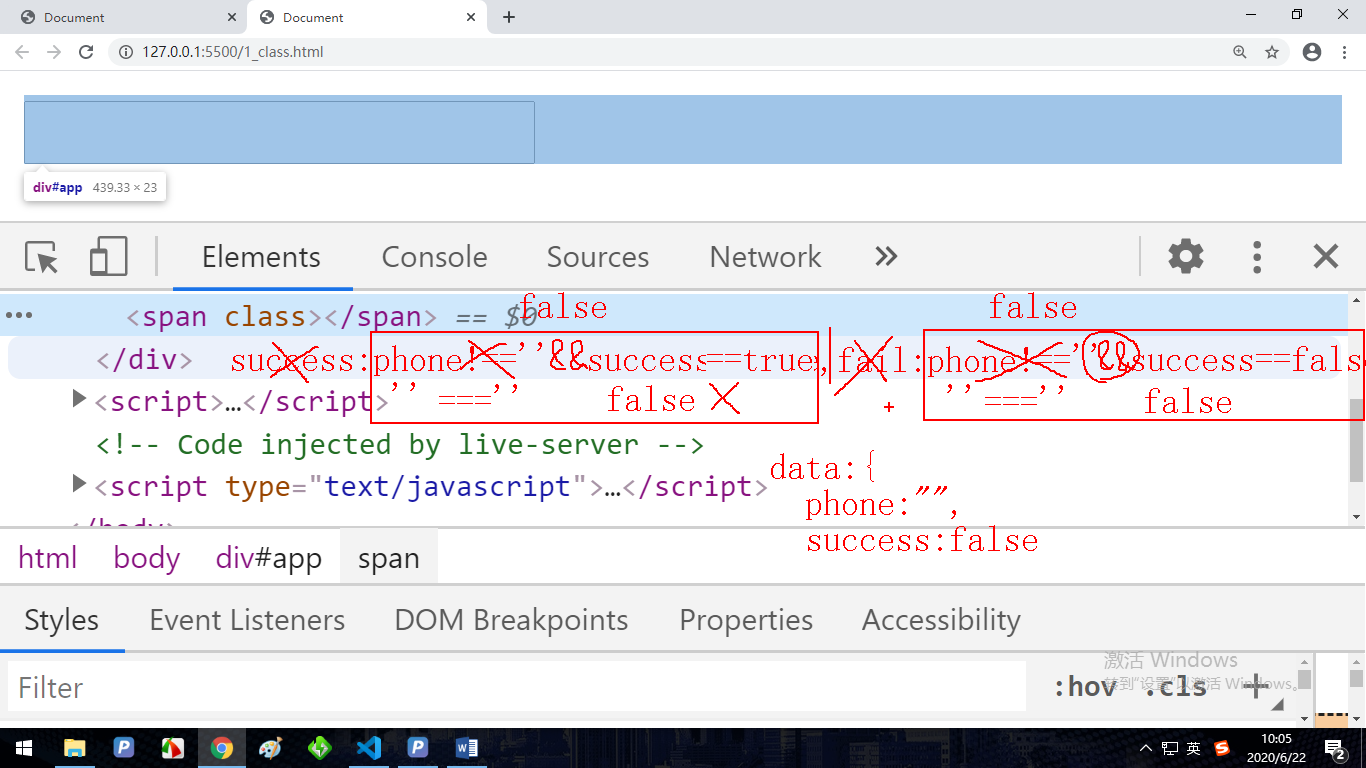
2). 原理: new Vue()首次扫描到这里时，或依赖的变量发生变化时

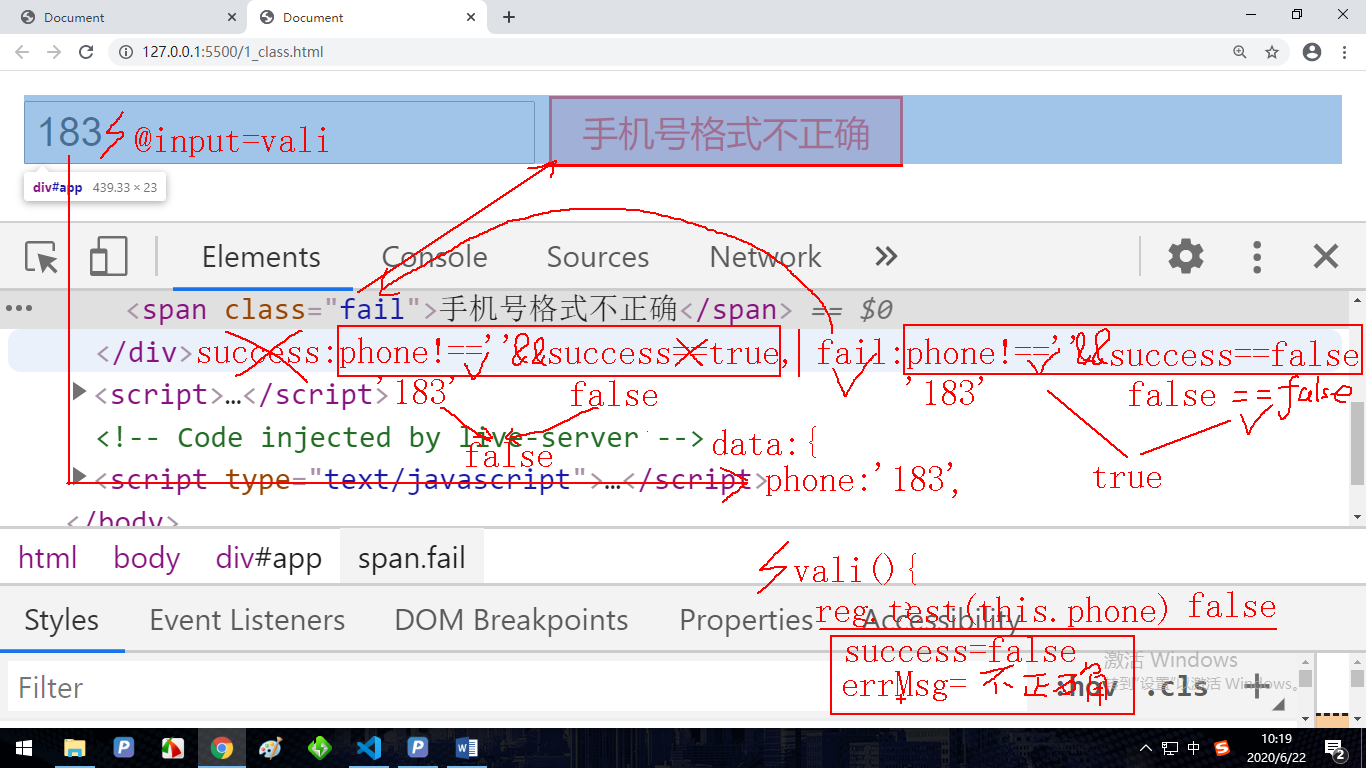
i. 计算每个class名之后的bool表达式或变量值

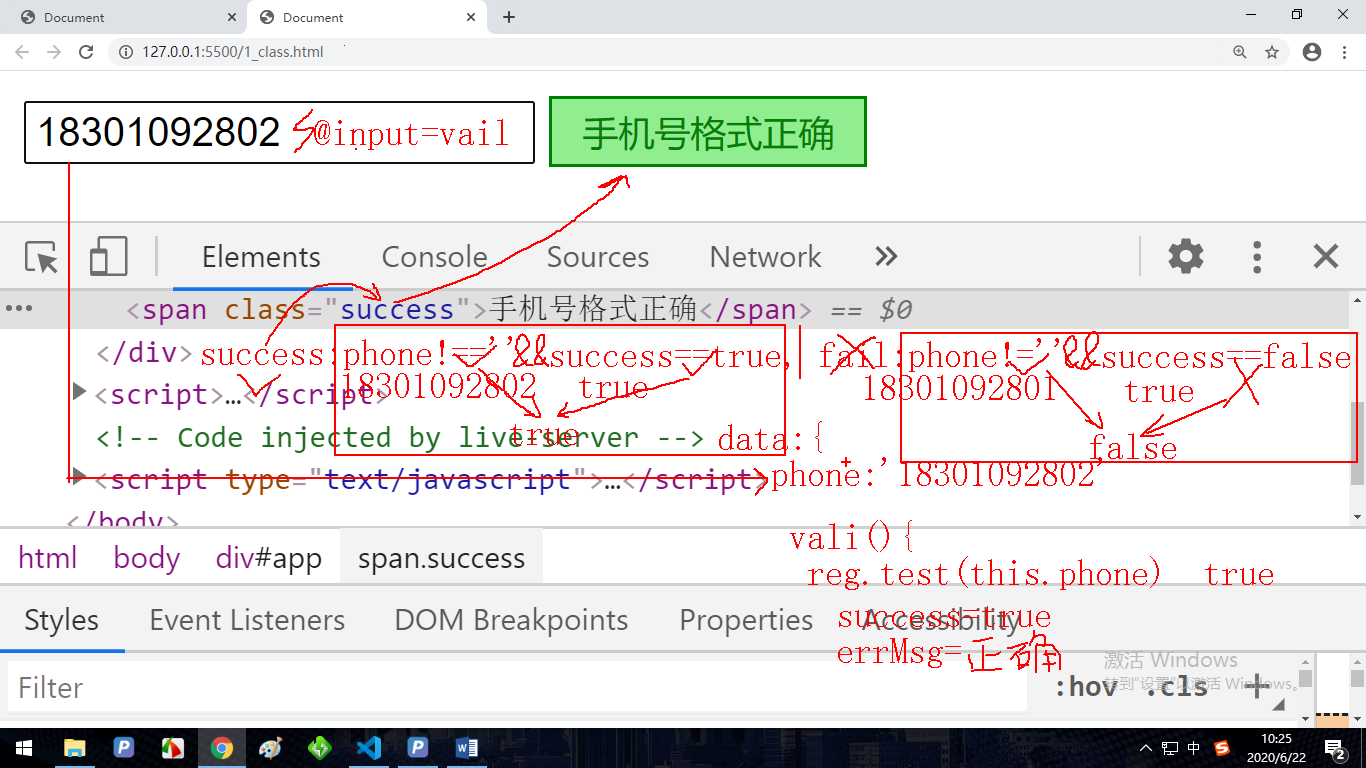
ii. 如果一个class名之后的bool表达式或变量值为true，则该class会出现在最终的元素上发挥作用——亮灯

iii. 如果一个class名之后的bool表达式或变量值为false，则该class不会出现在最终的元素上，不会发挥作用！——灭灯

3). 示例: 验证手机号格式是否正确——带css样式







1\_class.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <script src="js/vue.js"></script>  <style>  /\* 验证通过时的样式 \*/  .success{  border:1px solid green;  background-color:lightGreen;  color:green;  padding:3px 10px;  font-size:12px;  }  /\* 验证失败的样式 \*/  .fail{  border:1px solid red;  background-color:pink;  color:red;  padding:3px 10px;  font-size:12px;  }  </style>  </head>  <body>  <div id="app">  <!--验证前后界面上哪里可能发生变化: 3处  1. input的内容 - phone 存储用户输入的手机号  开局: phone:""  2. span的class - success 存储一个bool值表示是否验证通过  开局: success:false  3. span的内容 - errMsg 存储span中要显示的提示信息  开局: errMsg:""-->  <!--只要用户在input中输入内容时，触发验证，所以  绑定oninput事件，执行vali函数，验证文本框中新内容-->  <input v-model="phone" @input="vali">  <!--如果用户没有在文本框中输入内容，则根本不用应用任何class-->  <!--只有用户在文本框中输入了内容且验证结果通过，就启用success class，禁用fail  否则如果用户在文本框中输入了内容且验证结果未通过，就启用fail class，禁用success-->  <span :class="{  success:phone.trim()!==''&&success==true,  fail:phone.trim()!==''&&success==false  }">{{errMsg}}</span>  </div>  <script>  new Vue({  el:"#app",  //因为界面上需要三个变量所以  data:{  phone:"",  success:false,  errMsg:""  },  //因为界面上需要一个函数  methods:{  vali(){  console.log(this.phone);  //用正则验证phone变量中手机号的格式是否正确  //先定义手机号正则  var reg=/^1[3-9]\d{9}$/;  //在用正则验证phone的内容是否符合格式要求  var result=reg.test(this.phone.trim());  //如果phone的内容为空  if(this.phone.trim()===""){  //就清除errMsg的内容，等于清除span的内容,什么也不显示  this.errMsg="";  }else if(result==true){//否则如果验证通过:  //就修改success变量为true，表示验证通过  this.success=true;  //就修改errMsg变量为手机号格式正确  this.errMsg="手机号格式正确";  }else{//否则如果验证未通过:  //就修改success变量为false，表示验证未通过  this.success=false;  //就修改errMsg变量为手机号格式不正确  this.errMsg="手机号格式不正确";  }  }  }  })  </script>  </body>  </html>  运行结果: |

b. 对整个class指定一个变量对象，一个变量对象中包含多个动态变化的class

1). <元素 :class="变量名">

data:{

变量: {

class1:true或false,

class2:true或false

}

}

2). 示例: 使用一个class变量修改上例:

2\_class\_obj.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <script src="js/vue.js"></script>  <style>  .msg{  padding:3px 10px;  font-size:12px;  }  /\* 验证通过时的样式 \*/  .success{  border:1px solid green;  background-color:lightGreen;  color:green;  }  /\* 验证失败的样式 \*/  .fail{  border:1px solid red;  background-color:pink;  color:red;    }  </style>  </head>  <body>  <div id="app">  <!--验证前后界面上哪里可能发生变化: 3处  1. input的内容 - phone 存储用户输入的手机号  开局: phone:""  2. span的class - spanClass 用对象结构存储两个class success fail的bool值  开局: { success:false, fail:false }  3. span的内容 - errMsg 存储span中要显示的提示信息  开局: errMsg:""-->  <!--只要用户在input中输入内容时，触发验证，所以  绑定oninput事件，执行vali函数，验证文本框中新内容-->  <input v-model="phone" @input="vali">  <!--新方法，不要在HTML中做任何判断条件-->  <span class="msg" :class="spanClass">{{errMsg}}</span>  </div>  <script>  new Vue({  el:"#app",  //因为界面上需要三个变量所以  data:{  phone:"",  spanClass:{  success:false, //都灭灯  fail:false, //都灭灯  },  errMsg:""  },  //因为界面上需要一个函数  methods:{  vali(){  //用正则验证phone变量中手机号的格式是否正确  //先定义手机号正则  var reg=/^1[3-9]\d{9}$/;  //在用正则验证phone的内容是否符合格式要求  var result=reg.test(this.phone.trim());  //如果phone的内容为空  if(this.phone.trim()===""){  //就清除errMsg的内容，等于清除span的内容,什么也不显示  this.errMsg="";  //清除success和fail两个class的残留  this.spanClass={ success:false, fail:false }  }else if(result==true){//否则如果验证通过:  //就修改spanClass内的success值为true，同时修改fail值为false  this.spanClass={ success:true, fail:false };  //就修改errMsg变量为手机号格式正确  this.errMsg="手机号格式正确";  }else{//否则如果验证未通过:  //就修改spanClass内的success值为false，同时修改fail值为true  this.spanClass={ success:false, fail:true };  //就修改errMsg变量为手机号格式不正确  this.errMsg="手机号格式不正确";  }  }  }  })  </script>  </body>  </html>  运行结果:  C:\xampp\htdocs\dd\VUE\day03\2_class_obj1.pngC:\xampp\htdocs\dd\VUE\day03\2_class_obj2.png |

(4). 如果一个元素上既有不变的class，又有动态改变的class:

<元素 class="固定不变的class们" :class="可能动态改变的class们">

最终两个class会合并为一个class，作用到元素上。

二. 自定义指令:

对应小程序视频: 小程序->在线->VUE->day02 9. 自定义指令vue directive。。。

1. 何时: 希望在页面加载时就对元素做一些初始化的DOM操作:

比如: 自动获得焦点

2. 如何: 2步

(1). 向Vue家里添加一个自定义指令

Vue.directive("指令名",{ //强调: 指令名千万不要带v-前缀！

//回调函数，当首次加载页面时，代用当前指令的元素被渲染到页面之后

插入后

inserted(domElem){

//domElem会自动接住带有当前指令的dom元素对象

对当前DOM元素对象domElem执行一些DOM相关的初始化操作，比如获得焦点

}

})

(2). 使用自定义指令:

<元素 v-自定义指令名> //强调: 使用自定义指令时必须加v-前缀！

3. 结果:

(1). new Vue()首次扫描到这里时，会去Vue家里找有没有同名的自定义属性。

(2). 如果找到，就会自动调用自定义指令中的inserted()函数，并将当前带有自定义指令的DOM元素对象，传给inserted()第一个形参。

(3). 在inserted()回调函数内，对当前dom元素执行的操作，就会反应到页面上。

4. 示例: 使用自定义指令，让文本框自动获得焦点:

3\_directive.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <script src="js/vue.js"></script>  <script>  //向vue大家庭中添加一个新指令my-focus  Vue.directive("my-focus",{ //因为HTML语法不区分大小写，所以如果名字由多个单词组成，强烈推荐用-分割，而不用驼峰命名  //当当前元素被添加到页面上之后，当前元素自定获得焦点  inserted(domElem){  domElem.focus(); //复习DOM中四天上午  }  })  </script>  </head>  <body>  <div id="app">  <!--希望文本框input在页面加载时就自动获得焦点  1. vue官方没有定义让元素自动获得焦点的指令  2. 的确可以写文本框.focus()，但是vue中没地方写DOM代  唯一可以名正言顺写DOM代码的地方——自定义指令中-->  <input v-my-focus><button>百度一下</button>  </div>  <script>  new Vue({  el:"#app"  })  </script>  </body>  </html>  运行结果: |

三. 计算属性:

对应小程序视频: 小程序->在线->VUE->day02 10 计算属性

1. 什么是: 自己不实际保存属性值，每次使用属性时，都临时根据所依赖的变量动态计算出结果应用到页面上。

2. 何时: 如果一个属性值不是现成的，需要经过复杂的计算过程才能获得，都要用计算属性

3. 如何:

(1). 在new Vue()中添加新成员:

new Vue({

el:"#app",

data:{ ... },

methods:{ 事件处理函数 },

computed:{ 专门保存计算属性

属性名(){

return 根据其他变量动态计算出新属性值

}

}

})

(2). 使用计算属性:

<元素>{{ 属性名 }}</元素> //计算属性使用时一定不要加()！

4. 问题: 既然是一个函数，为什么不放在methods中？

答: 因为放在methods中的普通函数，每调用一次，都会重新执行一次复杂计算，计算结果不会被vue缓存并反复使用。

5. 计算属性的优点: 计算属性首次计算的结果会被vue缓存起来，即使反复使用计算属性也不会重复计算。除非依赖的变量发生了变化，才被迫重新计算。但是新值依然会被缓存，并重复使用！

6. 总结:

a. 今后如果侧重于做一件事儿，而不太关心返回值，就用methods中的普通函数！

b. 今后如果更侧重于使用函数返回的结果值，就用computed计算属性

7. 示例: 使用计算属性实现购物车总价:

4\_computed.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <script src="js/vue.js"></script>  </head>  <body>  <div id="app">  <!--希望显示购物车商品列表和购物车总价-->  <h3>总价: ¥{{total.toFixed(2)}}</h3>  <ul>  <li v-for="(p,i) of cart" :key="i">  {{p.pname}} | ¥{{p.price.toFixed(2)}} | {{p.count}} | ¥{{(p.price\*p.count).toFixed(2)}}  </li>  </ul>  <h3>总价: ¥{{total.toFixed(2)}}</h3>  </div>  <script>  new Vue({  el:"#app",  data:{  cart:[  {pname:"华为", price:5588, count:2},  {pname:"小米", price:3588, count:3},  {pname:"苹果", price:8588, count:1},  ]  },  methods:{    },  computed:{  //因为购物车总价不是现成的，需要经过复杂的遍历过程才能求出  //所以，用计算属性。  total(){  console.log(`调用了一次total()`)  var result=0;  for(var p of this.cart){  result+=p.price\*p.count;  }  return result;  }  }  })  </script>  </body>  </html>  运行结果: |

四. 过滤器:

对象小程序视频: 小程序->在线->VUE->day03 3. 过滤器

1. 什么是: 专门将变量的原始值经过加工后再显示 的一种特殊的函数

2. 为什么: 因为有些从服务器端拿到的变量值，不能直接给人看

比如: 日期和时间 性别

服务器端存的是: 毫秒数 1和0

人希望看到: 年月日时分秒 男和女

3. 何时: 只要变量的原始值不能直接给人看，需要加工后才能给人看时，都用过滤器

4. 如何: 2步

(1). 向vue大家庭中添加过滤器

进

↓

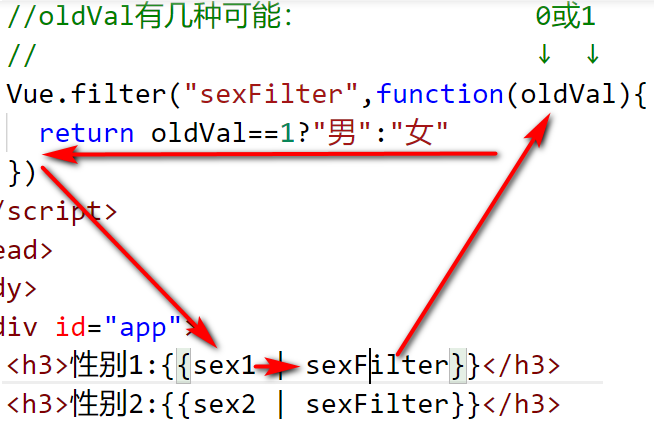
Vue.filter("过滤器名", function( oldVal ){

←出 return 根据oldVal加工后获得的新值

})

(2). 使用过滤器: <元素>{{ 变量 | 过滤器 }}</元素>

5. 示例: 使用过滤器加工性别0和1后再显示



5\_filter.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <script src="js/vue.js"></script>  <script>  //因为项目中所有性别都需要先加工再显示  //将来 {{ sex | sexFilter }}  //oldVal有几种可能： 0或1  // ↓ ↓  Vue.filter("sexFilter",function(oldVal){  return oldVal==1?"男":"女"  })  </script>  </head>  <body>  <div id="app">  <h3>性别1:{{sex1}}</h3>  <h3>性别2:{{sex2}}</h3>  <h3>性别1:{{sex1==1?"男":"女"}}</h3>  <h3>性别2:{{sex2==1?"男":"女"}}</h3>  <h3>性别1:{{sex1 | sexFilter}}</h3>  <h3>性别2:{{sex2 | sexFilter}}</h3>  </div>  <script>  new Vue({  el:"#app",  data:{  sex1:1,  sex2:0  }  })  </script>  </body>  </html>  运行结果: |

6. 过滤器可以带参数，根据不同的参数值，过滤出不同的结果:

(1). 定义过滤器时:

Vue.filter("过滤器名", function(oldVal, 自定义形参){

//接住的是过滤器之前的变量的原始值

//自定义形参可接住调用过滤器时传入的实参值

})

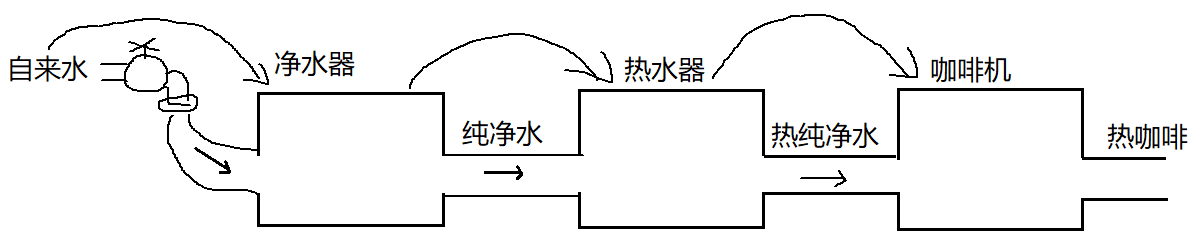
(2). 使用过滤器时:

<元素>{{ 变量 | 过滤器(自定义实参值) }}</元素>

7. 多个过滤器还可以连用: <元素>{{ 变量 | 过滤器1 | 过滤器2 | ... }}</元素>

强调: 只有第一个过滤器才能收到变量的原始值

之后其它过滤器收到的都不是原始值，而是上一个相邻的过滤器加工后的中间产物



8. 示例: 定义带参数过滤器过滤出不同语言种类的男和女，并且为性别添加图标

6\_filter.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <script src="js/vue.js"></script>  <script>  //希望根据输入的语言种类不同，加工出不同语言的男和女  // 预留 自定义  Vue.filter("sexFilter",function(oldVal, 语言种类){  //如果语言种类传入"英文",则加工出Male Female  if(语言种类=="英文"){  return oldVal==1?"Male":"Female"  }else{//否则如果传入的语言种类不是"英文"，则默认加工出男和女  return oldVal==1?"男":"女"  }  });  //为了能够给性别末尾追加图标:  //可能: {{sex | sexIcon }} 0和1  //也可能: {{ sex | sexFilter | sexIcon}} 男和女  //也可能: {{ sex | sexFilter("英文") | sexIcon}} Male和Female  // 预留  Vue.filter("sexIcon",function(oldVal){  //oldVal可能6种: 1和0 男和女 Male和Female  //如果oldVal是1或0，则直接返回♂和♀  if(oldVal==1){  return "♂"  }else if(oldVal==0){  return "♀"  }else if(oldVal=='男'||oldVal=="Male"){//否则如果oldVal不是0或1则需要将♂和♀拼接到现有oldVal结尾  return oldVal+"♂"  }else{  return oldVal+"♀"  }  })  </script>  </head>  <body>  <div id="app">  <h3>性别1:{{sex1 | sexIcon}}</h3>  <h3>性别2:{{sex2 | sexIcon}}</h3>  <h3>性别1:{{sex1 | sexFilter | sexIcon}}</h3>  <h3>性别2:{{sex2 | sexFilter | sexIcon}}</h3>  <!-- 自定义 -->  <h3>性别1:{{sex1 | sexFilter("英文") | sexIcon}}</h3>  <h3>性别2:{{sex2 | sexFilter("英文") | sexIcon}}</h3>  </div>  <script>  new Vue({  el:"#app",  data:{  sex1:1,  sex2:0  }  })  </script>  </body>  </html>  运行结果： |

五. axios:

对应小程序视频: 小程序->在线->VUE->day03 4. axios

1. 什么是: 专门在各种平台都能发送ajax的基于promise的函数库

2. 为什么: jQuery不是已经又$.ajax()

今后框架开发，几乎不用jquery的。仅仅为了使用一个函数$.ajax()发送ajax请求，就把包含几十个上百个函数的jQuery函数库引入进来，极其不划算！

今后其它非jquery的前端框架，急需要一种专门发送ajax请求的函数库。

3. 何时: 只要在vue中发送ajax请求，则都用axios。

但是其实axios绝不仅局限于vue中，普通网页中甚至nodejs中都可以使用

4. 如何:

(1). 发送get请求:

axios.get("服务器端接口地址", {

params:{ 变量名:值, ...: ..., }

}).then(function(result){ //.then会在成功接收到服务器端返回的结果时自动触发

//坑: result不是服务器端返回的数据

//result.data才是服务器端返回的数据！

})

(2). 发送post请求：

axios.post("服务器端接口地址","变量1=值1&变量2=值2&...")

.then(function(result){ //.then会在成功接收到服务器端返回的结果时自动触发

//坑: result不是服务器端返回的数据

//result.data才是服务器端返回的数据！

})

5. 问题: 项目中很多地方都要写ajax请求，如果每次都从http://开始写服务器端接口的完整路径，太麻烦了！

6. 解决: 其实一个项目中的绝大多数接口的域名部分是完全一样的。

(1). axios提供了一个位置，专门保存所有接口共用的相同部分的域名。

axios.defaults.baseURL="http://基础域名"

(2). 用axios发送请求时，只写接口的相对路径即可，运行时axios自动将保存的基础域名部分和相对接口名称拼接为完整的接口地址，再发送请求。

axios.get("/接口名",...).then(...)

7. 示例: 使用axios向东哥新浪云学子商城服务端接口发送三种请求:

7\_axios.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <script src="js/axios.min.js"></script>  </head>  <body>  <script>  //配置所有接口相同的基础域名路径部分  axios.defaults.baseURL="http://xzserver.applinzi.com";  axios.get("/index")  .then(function(result){  console.log(result.data);  });  axios.get("/details",{  params:{ lid:5 } //自动翻译为?lid=5，添加到url结尾  }).then(function(result){  console.log(result.data);  });  axios.post(  "/users/signin",  "uname=dingding&upwd=123456"  ).then(function(result){  console.log(result.data);  });  </script>  </body>  </html>  运行结果： |

8. 问题: axios.get().then(function(){ ... })this->window，但是今后我们希望new Vue()中所有this都要指向当前new Vue()对象

9. 解决: 今后axios.get().then(function(){ ... })

↓

axios.get("/接口名",...).then(reuslt=>{ ... })

结果: .then()回调函数中的this和外部的this保持一致了！

六. vue对象生命周期:

对应小程序视频: 小程序->在线->VUE->day03 5. 生命周期

1. 什么是: 一个VUe对象从创建到所有页面内容加载完成所经历的过程

2. 包括: 4个阶段:

(1). 创建 create 必经

a. 创建new Vue()对象，在new Vue()对象中创建data对象并请保镖

b. 暂时不扫描DOM树，暂时没有虚拟DOM树

(2). 挂载 mount 必经

a. 扫描DOM树，生成虚拟DOM树，并首次挂载数据到页面元素上

-------------------------------------------

(3). 更新 update 当data中的变量被修改时才触发

(4). 销毁 destroy 还有调用$destroy函数销毁当前组件时，才触发 —— 很少用

3. 为什么要学习生命周期:

(1). jquery中，在页面首次加载时有很多初始化操作，比如自动发送ajax请求，比如自动绑定首屏数据。这些操作都写在$(function(){ ... })中。凡是写在$(function(){ ... })中的代码，会在DOM内容加载完成后自动执行！

(2). 问题: vue框架中几乎不用jquery，自然也就没有$(function(){ ... })，比如自动发送ajax请求获取首屏数据的代码应该写在哪儿呢？

(3). 错误: 将axios请求的代码放在new Vue()外部。

因为将来vue采用组件化开发，规定一个组件的所有js代码必须放在这个组件的对象内部！组件对象之外不允许出现任何逻辑相关的代码

(4). 正确: 将要执行的页面初始化js代码，绑定到vue的某个生命周期阶段，自动执行！

4. 何时: 今后只要希望在vue对象创建和挂载过程中，自动执行某些操作时，都要用生命周期

5. 如何: vue对象有四个生命周期，每个生命周期前后都有一对儿（2个）钩子函数(回调函数)

beforeCreate() 在new Vue()创建前自动触发，没有data和变量呢，无法修改data中的变量——不适合发送首屏ajax请求

(1). 创建阶段 create

created() 创建完new Vue()和data对象之后自动触发

a. 可以操作data中的变量

b. 但是暂时无法执行DOM相关的操作，因为还没有扫描DOM树

beforeMount() 在开始扫描DOM树之前自动触发

a. 也可以操作data中的变量

b. 但是暂时也无法执行DOM相关的操作，因为还没有扫描DOM树

(2). 挂载阶段 mount

mounted() 在创建完虚拟DOM树，首次挂在页面内容完成之后自动触发

a. 已经有了data对象，可以操作data中的变量

b. 已经有了虚拟DOM树，且数据已经显示在页面上了，可以执行DOM相关操作

总结: mounted()才是最适合做页面初始化工作

beforeUpdate() 在开始修改data中的变量之前自动触发

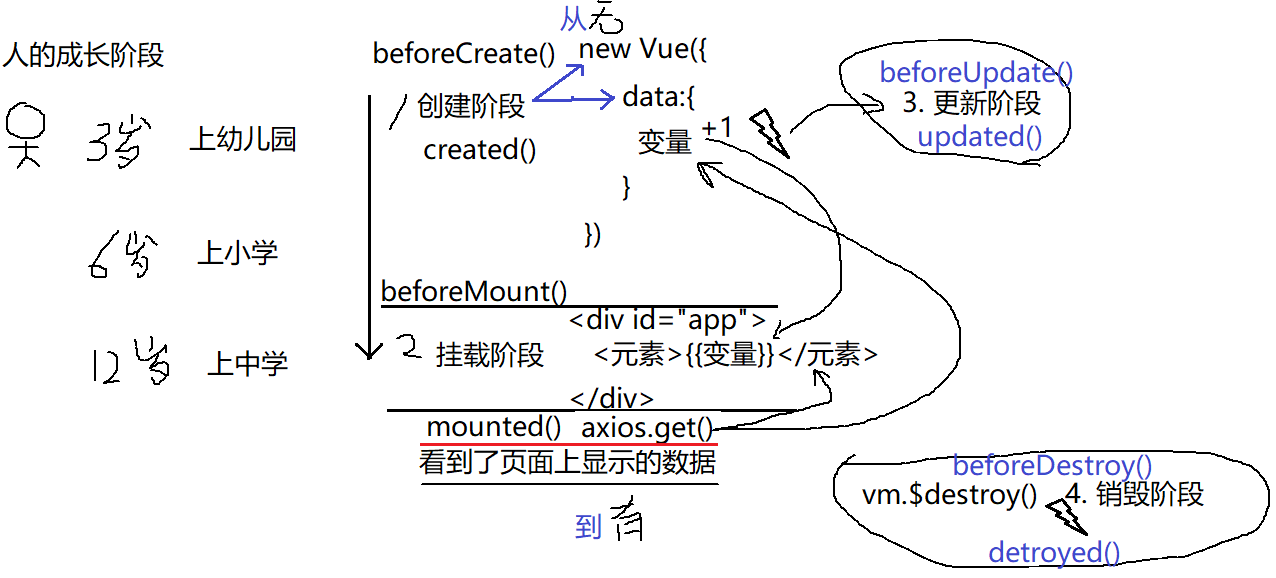
(3). 更新阶段 update

updated() 修改完data中的变量之后自动触发

beforeDestroy() 在开始销毁当前组件之前自动触发触发

(4). 销毁阶段 destroy

destroyed() 在销毁当前组件之后自动触发



6. 示例: 演示一个vue组件的4个生命周期和8个钩子函数

强调: 页面上<div id="app">中必须绑定并使用了data中的products变量才能出发更新阶段

8\_lifecycle.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <script src="js/vue.js"></script>  <script src="js/axios.min.js"></script>  </head>  <body>  <!--希望页面加载时，就加载学子商场首页的六个商品-->  <div id="app">  <ul>  <li v-for="(p,i) of products" :key="i">  {{p.title}} | ¥{{p.price}} | {{p.details}}  </li>  </ul>  </div>  <script>  //配置  axios.defaults.baseURL="http://xzserver.applinzi.com";  var vm=new Vue({  el:"#app",  data:{  products:[]  },  methods:{  },  beforeCreate(){  console.log(`创建data:{}前自动触发`)  },  created(){  console.log(`创建data:{}后自动触发`)  },  beforeMount(){  console.log(`挂载页面元素和内容前自动触发`)  },  mounted(){  console.log(`挂载页面元素和内容后自动触发`)  //为了让.then()中的this和外部this保持一致，都指向当前new Vue()对象  axios.get("/index").then(result=>{  //想把请求回来的数据，保存到data中products变量上  this.products=result.data;  })  },  beforeUpdate(){  console.log(`修改data中的变量前自动触发`)  },  updated(){  console.log(`修改data中的变量后自动触发`)  },  beforeDestroy(){  console.log(`销毁当前组件前自动触发`)  },  destroyed(){  console.log(`销毁当前组件后自动触发`)  }  })  //在控制台中: vm.$destory()  </script>  </body>  </html>  运行结果: |

总结:

1. MVVM: 界面View+模型对象Model+视图模型ViewModel

2. Vue的绑定原理: 访问器属性+虚拟DOM树(+观察者模式)

3. 虚拟DOM树的优点:

(1). 小，仅包含可能变化的元素

(2). 遍历查找快

(3). 更新效率高，只更新受影响的元素

(4). 避免重复编码，已封装了DOM的增删改查操作

总结: 今后所有VUE功能三步:

1. 创建界面(增强)，界面中标记变量和事件处理函数

2. 创建new Vue()对象，用el:"选择器"，找到自己负责的页面区域的唯一父元素

3. 在new Vue()中添加data和methods，分别保存界面所需的所有变量和事件处理函数

总结:

1. 绑定语法(1种)和指令(13种):

(1). 只要元素的内容随变量自动变化: {{}} v-text

(2). 只要元素的属性值随变量自动变化: :

(3). 控制一个元素显示隐藏: v-show （display:none）

(4). 控制两个元素二选一显示: v-if v-else （删除元素）

(5). 控制多个元素多选一显示: v-if v-else-if v-else

(6). 反复创建多个相同结果的元素: v-for :key="下标"

(7). 绑定事件: @ $event

(8). 防止用户短暂看到{{}}: v-cloak v-text

(9). 只要绑定的内容是HTML片段: v-html

(10). 只要希望一个元素只在首次加载时绑定一次，之后都不会自动更新: v-once

(11). 只要不希望元素内容中的{{}}被vue解析，而是保持原样显示: v-pre

(12). 只要希望随时获得页面上用户在表单元素中修改的新值: v-model

2. 绑定样式:

(1). 如果希望精确修改某一个css属性的值时，绑定style内联样式:

a. 为每个要修改的css属性分别设置变量

<元素 style="固定不变的css属性 " :style="{ css属性1: 变量1, css属性2: 变量2, ... }"

data:{

变量1: css属性值1,

变量2: css属性值2

}

b. 为整个style属性设置一个变量对象，其中包含多个css属性和属性值

<元素 style="固定不变的css属性 " :style="xxxStyle"

data:{

xxxStyle:{

css属性1: css属性值1,

css属性2: css属性值2

}

}

(2). 如果希望批量修改一个元素的多个css属性的值时，就要绑定class属性:

a. 为每个class分别设置条件

<元素 class="不变的class " :class="{class1: bool表达式或变量, class2:bool表达式或变量 }"

bool表达式或变量的值为true就应用，false就不应用

b. 为整个class属性设置一个变量对象，其中包含多个备选的class名极其对应判断条件

<元素 class="不变的class " :class="xxxClass"

data:{

xxxClass:{

class1: true或false, //true就应用，false就不应用

class2: true或false //true就应用，false就不应用

}

}

3. 自定义指令: 2步

(1). 添加自定义指令:

Vue.directive("指令名",{

inserted(domElem){

对domElem执行DOM操作

}

})

(2). 使用自定义指令: <元素 v-自定义指令名>

4. 计算属性:

(1). 定义:

new Vue({

... ,

computed:{

自定义属性名(){

计算过程

return返回值

}

}

})

(2). 使用计算属性: <元素>{{自定义属性名}}</元素>

5. 过滤器:

(1)不带参过滤器:

a. 添加过滤器:

Vue.filter("过滤器名", function(oldVal){

return 根据不同的oldVal加工出不动的新值

})

b. 使用过滤器: <元素>{{ 变量 | 过滤器 }}</元素>

(2). 带参数过滤器：

a. 添加过滤器:

Vue.filter("过滤器名", function(oldVal, 自定义形参){

return 根据不同的oldVal加工出不动的新值

})

b. 使用过滤器: <元素>{{ 变量 | 过滤器(自定义实参) }}</元素>

(3). 多个过滤器连用: <元素>{{ 变量 | 过滤器1 | 过滤器2 | ...</元素>

6. axios:

(0). axios.default.baseURL="http://所有服务器端接口统一的域名部分"

(1). axios.get("/接口名", { params:{ 参数:值, 参数:值, ... } })

.then(function(result){

result.data ...

})

(2). axios.post("/接口名", "参数1=值1&参数2=值2&...")

.then(function(result){

result.data ...

})

7. vue生命周期: 4个阶段 8个钩子函数

beforeCreate()

(1). 创建阶段 create

created()

beforeMount()

(2). 挂载阶段 mount

mouted()

beforeUpdate()

(3). 更新阶段 update

updated()

beforeDestroy()

(4). 销毁阶段 destroy

destroyed()

总结: this 7种:

1. obj.fun() this->obj

2. new Fun() this->new正在创建的子对象

3. 类型名.prototype.共有方法=function(){ ... } this->将来调用这个共有方法的.前的子对象

4. fun() 和 (function(){ })() 和回调函数中的this->window

5. 事件处理函数中的this->当前正在触发事件的这个元素对象

6. $jquery对象.click(function(){ this->将来点击的那个DOM元素对象})

7. new Vue()中的几乎所有this，都指向当前new Vue()对象。（回调函数用箭头函数）

今日对应小程序视频:

小程序->在线->VUE->day02 8. 绑定样式...

9. 自定义指令...

10. 计算属性...

小程序->在线->VUE->day03 3. 过滤器 ...

4. axios

5. 生命周期

作业:

1. 复习今日小程序问题清单(小程序->首页-> VUE ->day02、day03)，预习明日小程序问题清单(小程序->首页->VUE->day04)

2. 回看TMOOC第一阶段NODEJS第一天、第二天 模块相关内容——重要

3. 看小程序视频预习组件化开发:

小程序->在线->VUE->day03 6. Vue组件化开发 todo 。。。

4. 看小程序视频预习SPA内容

小程序->在线->VUE->day04 1. SPA...