小程序: WEB问题速查

1. 问题清单: 历届师兄师姐在今天课程中集中问过的高频的重点的问题！

那些自己发现不了问题，又迫切想知道其它同学都问了什么问题，的同学

小程序->首页->VUE->day01、day02

当老师在回答其它同学的问题时，你不要闲着！你可以打开问题清单浏览别人曾经问过的经典问题！

2. 高清无码小视频: 小程序->在线->VUE->day01、day02

2部分: 1. 课上重点知识和案例的回放！2. 重点高频笔试面试题讲解扩展视频

万一家里或中心网络断了！暂时看不了直播了！

因为某些事儿请假了，错过了当天的直播！

课下复习时，觉得TMOOC视频太大，浏览不方便或TMOOC不能看了！

——打开小程序中"在线"，就可下载观看当前所讲内容和当天所讲全部内容的回放。

基础比较好的，确实学有余力的同学，想拿高薪的同学，在我答疑时，不要闲着:

要看今天扩展视频:

小程序->在线->VUE->day01

作业: 为v-show和v-if添加过渡动画效果

作业: 使用vue实现标签页效果tabs

小程序->在线->VUE->day02

10. 计算属性 computed 作业 vue 购物车 仅包含客户端

3. 双向绑定 v-model 绑定文本框...边输入边搜索，按回车搜索，防抖和节流

4. 双向绑定 v-mode radio ... 选择飞机

5. 双向绑定 v-model select ... 级联选择省份和城市

6. 双向绑定 v-model checkbox ... 多选 爱好

3. 单词列表: 小程序->单词->VUE->day01、day02

第三阶段上课约定:

1. 上课千万不要跟着记笔记和写代码！

2. 凡是拼写错误，自己懒得找！我可以帮你找，但是找到后，错误的单词你要手抄20遍，并拍照！

出错: 先开f12，console，看错误提示！

3. 每个带注释的案例至少写三遍:

(1). 注释+代码抄一遍

(2). 保留注释，删除代码，自己试着把代码翻译回来！

(3). 删除注释和代码，自己试着用自己的话把注释写回来！

一个例子只有自己可以把人话的注释写回来！才算自己会！

问题: 一听就会一做就废！原因: 你自己无法用人话精确描述自己想干什么！

如何提问: （vip待遇的剩余时间4+2天）

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻问！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻问！

3. 个人项目中哪个功能不会做！立刻问！

4. 刷笔试题时，遇到不会的笔试题！立刻问！

谁问的问题越多！谁的培训费交的越值！

正课:

1. 绑定语法:

2. 指令:

3. 双向绑定:

对应小程序视频: 小程序->在线->VUE->day01 3. 绑定语法 {{}} v-bind ...v-show...v-if...

一. 绑定语法: 学名 Interpolation 插值语法、填空

1. 问题: 旧DOM和jq中，程序中的变量虽然发生了变化，但是界面不会自动更新，必须依靠js手工将变量的新值更新到页面，导致大量重复的增删改查操作！

2. 解决: VUE中让HTML中也支持变量了！做到程序中的变量变化，HTML界面中的显示自动变化！

3. 什么是绑定语法: 让HTML中也支持变量的一种专门的语法格式

4. 何时: 只要HTML中有的元素**内容**需要随变量自动变化时，都要用绑定语法

5. 如何: **<元素>xxxx{{自定义变量名}}xxxx</元素>**

6. 原理: 当new Vue()扫描页面时，只要扫描到{{}}这种语法，就会将{{}}中的js变量或表达式的值计算出来，再替换{{}}的位置最终显示给用户。

7. {{}}中可以放什么，不能放什么？和ES6的模板字符串${}规定是完全一样的！

(1). 可以放: 变量，运算，三目，有返回值的函数调用，创建对象，访问数组元素

(2). 不能放: 程序结构(分支和循环)，也不能放没有返回值的函数调用！

8. 示例: 使用{{}}将程序中的变量值加工后，显示到界面上

1\_{{}}.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>    <!--1. 先做界面-->    <div id="app">      <h3>用户名:{{uname}}</h3>      <!--如果  性别==1，就显示"男",否则 就显示"女"-->      <h3>性别:{{sex==1  ?   "男"   :       "女"}}</h3>      <!--        单价  x 数量  按2位小数四舍五入-->      <h3>小计:¥{{(price\*count).toFixed(2)}}</h3>      <!--新建日期对象，保存毫秒数，并转为当地时间的字符串格式-->      <h3>下单时间: {{        new Date(orderTime).toLocaleString()      }}</h3>      <h3>今天星期{{week[new Date().getDay()]}}</h3>    </div>    <script>      //2. 创建new Vue()对象监控id为app的区域      new Vue({        el:"#app",        //3. 创建模型对象保存页面所需的所有变量        data:{          uname:"dingding",          sex:1,//性别          price:12.5,//单价          count:5,//数量          orderTime:1595296375921,//下单时间          week:["日","一","二","三","四","五","六"]          //     0    1    2    3    4    5   6          //               ↑          //               2          //new Date().getDay()          //创建日期对象          //并自动获得当前系统时间          //           获得星期对应的数字        }      })    </script>  </body>  </html>  运行结果: |

二. 指令: 13种

1. 问题: {{}}可让元素的内容随变量自动变化，但是页面中可能发生变化的不止是内容！元素的属性和样式也可能发生变化！但是**{{}}只支持内容**变化，**不支持属性值**和样式变化！

2. 解决: 指令

3. 什么是: 指令就是为HTML元素添加更多新功能的特殊属性！

4. 何时: 今后只要希望给HTML添加更多新功能，比如分支，循环等等，都要用指令来实现

5. 包括:

(1). v-bind:

a. 什么是: 专门让属性值也能根据变量值自动变化

b. 何时: 今后只要希望属性值也能根据变量值自动变化时，都用v-bind

c. 如何: <元素 v-bind:属性名="变量或js表达式">

d. 强调: 一旦使用了v-bind，""中就**不要再加{{}}**了！""就承担了{{}}的作用！

e. 简写: 其实今后v-bind: ，只需要写**:**即可！v-bind可省略 ！

**<元素 :属性名="变量或js表达式">**

f. 原理: 当new Vue()扫描到v-bind: 或 : 时会自动计算""中的变量值或js表达式的值，将计算后的新值，作为该属性的属性值！

g. 示例: 根据pm25的数值显示不同的图片

2\_v-bind.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>    <!--1. 做界面-->    <div id="app">      <!--想实现根据空气质量pm25的数值，修改页面中img元素显示的人物表情-->      <!--      //如果pm25<100，则使用img/1.png            pm25<100  ?'img/1.png':      //否则如果pm25<200，则使用img/2.png               pm25<200?'img/2.png':      //否则如果pm25<300，则使用img/3.png               pm25<300?'img/3.png':      //否则，其它情况，都使用img/4.png                        'img/4.png'      -->      <h3>空气质量: {{        pm25<100?'img/1.png':        pm25<200?'img/2.png':        pm25<300?'img/3.png':                 'img/4.png'      }}</h3>      <!--<img v-bind:src="-->      <img :src="        pm25<100?'img/1.png':        pm25<200?'img/2.png':        pm25<300?'img/3.png':                 'img/4.png'      " alt="">    </div>    <script>      //2. 创建new Vue()对象监控id为app的区域      new Vue({        el:"#app",        //3. 创建模型对象保存界面所需的变量        data:{          pm25:350        }      })    </script>  </body>  </html>  运行结果: |

(2). v-show:

a. 什么是: 专门根据一个条件控制一个元素的显示隐藏的指令

b. 何时: 今后只要控制一个元素显示隐藏，都用v-show

c. 如何: **<元素 v-show="bool变量或js条件表达式">**

d. 原理: 当new Vue()扫描到v-show时，会自动计算""中变量或表达式的bool值。 1). 如果计算结果为true，则什么也不做，当前元素默认显示。

2). 如果计算结果为false，则自动给当前元素加display:none，则当前元素隐藏。

e. 示例: 使用vue控制对话框显示隐藏

3\_v-show.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>      .alert{        position:fixed;        width:400px;        height:200px;        background-color:#afa;        top:50%;        left:50%;        margin-left:-200px;        margin-top:-100px;      }      .alert>a{        position:absolute;        right:10px;        top:10px;        padding: 5px 10px;        border:1px solid blue;      }    </style>  </head>  <body>    <!--1. 做界面-->    <div id="app">      <!--点击按钮可以让对话框显示-->      <button @click="show">click me</button>      <!--本例中: 希望点按钮对话框div才显示，点x对话框div就隐藏，所以需要控制对话框div的显示和隐藏状态，        所以需要一个自定义变量isShow：        isShow为true时，对话框才显示        isShow为false时，对话框就隐藏 -->      <div v-show="isShow" class="alert">        <!--点×可以隐藏对话框-->        <a href="javascript:;" @click="hide">×</a>      </div>    </div>    <script>      //2. 创建new Vue()对象,监控id为app的区域      new Vue({        el:"#app",        //3. 创建模型对象，保存界面所需的所有变量        data:{          isShow:false //开局，对话框默认不显示        },        //因为页面中需要两个事件处理函数，所以        methods:{          show(){            this.isShow=true;          },          hide(){            this.isShow=false;          }        }      })    </script>  </body>  </html>  运行结果: |

(3). v-if和v-else:

a. 什么是: 专门根据一个条件控制两个元素二选一显示

b. 何时: 今后只要**两个元素二选一显示**时，都用**v-if和v-else**

c. 如何:

**<元素1 v-if="bool变量或js条件表达式"></元素1>**

**<元素2 v-else></元素2>**

d. 强调: v-if和v-else两个元素**必须紧挨着写**！中间不能插入任何其它内容！

e. 原理: new Vue()扫描到v-if和v-else这里时，会自动计算变量值或条件表达式的结果。

1). 如果v-if中的条件表达式计算结果为true，则保留v-if所在的元素，删除v-else所在的元素

2). 如果v-if中的条件表达式计算结果为false，则删除v-if所在元素，保留v-else所在元素

f. 高频笔试题: v-show和v-if都能控制元素显示隐藏，差别:

**1). v-show是通过display:none来控制显示隐藏**

**2). v-if是通过删除元素来控制显示隐藏**

g. 示例: 根据用户的登录状态，显示不同的内容

4\_v-if\_v-else.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>    <!--1. 做界面-->    <div id="app">      <!--如果已登录，显示Welcome，用户名          否则如果未登录，显示登录和注册按钮          所以,需要一个自定义变量isLogin来记录当前用户的登录状态-->      <h3 v-if="isLogin">        <!--点注销按钮，将当前用户设置为未登录状态-->        Welcome dingding | <a href="javascript:;" @click="logout">注销</a>      </h3>      <h3 v-else>        <!--点登录按钮，将当前用户设置为已登录状态-->        <a href="javascript:;" @click="login">登录</a> | <a href="javascript:;">注册</a>      </h3>    </div>    <script>      //2. 创建new Vue()对象，监控id为app的区域      new Vue({        el:"#app",        //3. 创建模型变量保存界面上所需的所有变量        data:{          isLogin:false, //开局用户默认处于未登录状态        },        //因为界面上需要两个事件处理函数，所以        methods:{          login(){            this.isLogin=true;          },          logout(){            this.isLogin=false;          }        }      })    </script>  </body>  </html>  运行结果： |

(4). v-else-if

a. 什么是: 专门和v-if和v-else配合来控制多个元素多选一显示

b. 何时: 只要**多个元素多选一显示时**

c. 如何:

**<元素1 v-if="条件1">**

**<元素2 v-else-if="条件2">**

**... ...**

**<元素n v-else>**

d. 强调: v-if v-else-if v-else的几个元素必须紧挨着写，中间不能插入其他内容。

e. 原理:

1). 当new Vue()扫描到v-if时，先计算v-if后的变量或条件表达式，如果v-if后的条件为true，则保留v-if删除其它v-else-if和v-else

2). 如果v-if的条件为false，则继续向后扫描每个v-else-if。只要任意一个v-else-if后的条件为true，则删除除该v-else-if之外的其余兄弟元素，只显示当前v-else-if的元素.

3). 如果所有v-if和v-else-if的条件都为false，则删除所有v-if和v-else-if的元素，仅保留v-else所在元素显示。

f. 示例: 根据pm25数值选择一张图片显示

5\_v-else-if.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>    <!--1. 做界面-->    <div id="app">      <!--想根据pm25的数值在四张图片中选其一显示-->      <!--如果pm25<100，显示第一张图片-->      <!--否则如果pm25<200，显示第二张图片-->      <!--否则如果pm25<300, 显示第三张图片-->      <!--否则如果以上条件都不满足，才显示第四张图片-->      <img v-if="pm25<100" src="img/1.png" alt="">      <img v-else-if="pm25<200" src="img/2.png" alt="">      <img v-else-if="pm25<300" src="img/3.png" alt="">      <img v-else src="img/4.png" alt="">    </div>    <script>      //2. 创建new Vue()对象，监视id为app的区域      new Vue({        el:"#app",        //3. 创建模型对象，保存界面中所需的所有变量        data:{          pm25:350        }      })    </script>  </body>  </html>  运行结果: |

(5). v-for

对应小程序视频列表: 小程序->在线->VUE->day01 4. v-for 遍历数组 ...

a. 什么是: 专门根据数组中的元素内容，自动反复生成多个相同结构的但是内容不同的页面元素列表 的指令

b. 何时: 只要在页面中反复生成多个相同结构的元素组成的列表

c. 如何: **<要反复生成的元素 v-for="(元素值, 下标) of 数组" :key="下标">**

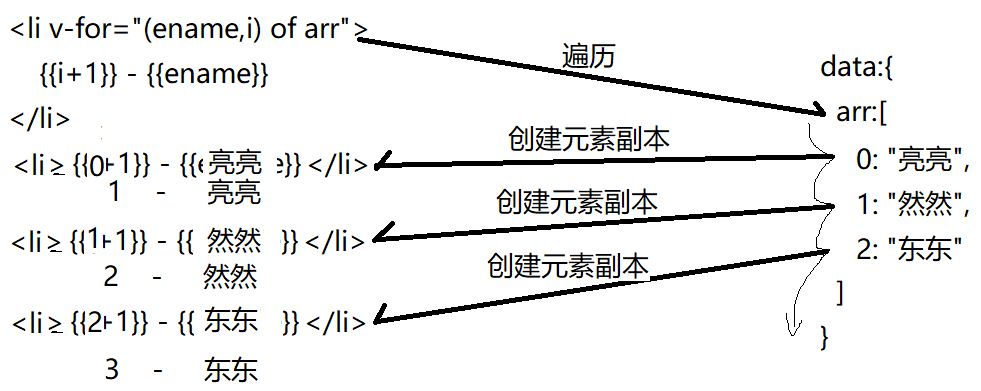
d. 原理:

1). 当new Vue()扫描到v-for时，会自动遍历of后的数组

2). 每遍历数组中的一个元素值，

i. 就自动将当前元素值和当前下标位置保存到of前的两个变量中（顺序不能变）。

ii. 还会自动创建当前v-for所在的元素副本，并用of前的两个变量内容，填充新生成的相同结构的元素副本。



e. 示例: 使用v-for遍历人名数组，并在页面显示人名列表

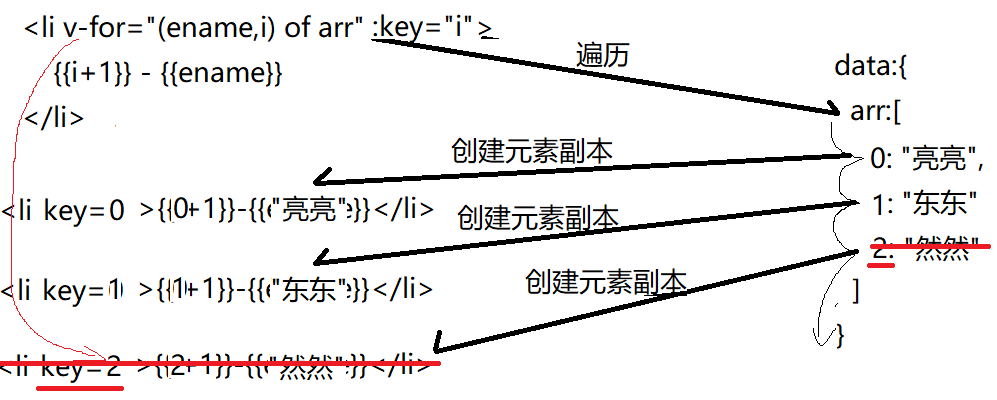
6\_v-for.html

|  |
| --- |
| <!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>    <!--1. 做界面-->    <div id="app">      <!--想显示一个人名的列表,所以在程序中应该有一个数组arr来保存所有的人名,并HTML中应该用v-for来遍历数组中每个人名，每遍历一个人名就创建一个li元素的副本，每个li元素中都显示当前人名的下标和人名-->      <ul>        <li v-for="(ename,i) of arr">{{i+1}} - {{ename}}</li>      </ul>    </div>    <script>      //2. 创建new Vue()对象，监视id为app的区域      new Vue({        el:"#app",        //3. 创建模型对象，保存页面所需的所有变量        data:{          arr:["亮亮","然然","东东"]        }      })    </script>  </body>  </html>  运行结果: |

f. 问题: v-for生成的多个相同结构的元素副本除了内容不同之外，元素本身毫无差别！所以，万一数组中某个位置的元素被删除一个，v-for无法知道该精确的删除哪一个页面元素副本，v-for只能采用一个很笨的办法，删除所有副本，重建整个列表！——效率极低！

g. 解决: 今后只要使用v-for反复生成多个相同结构的元素副本时，都要为每个元素副本额外绑定一个专门的属性: **:key="不重复的值"**，key绑定的值要求必须是唯一的值不能重复

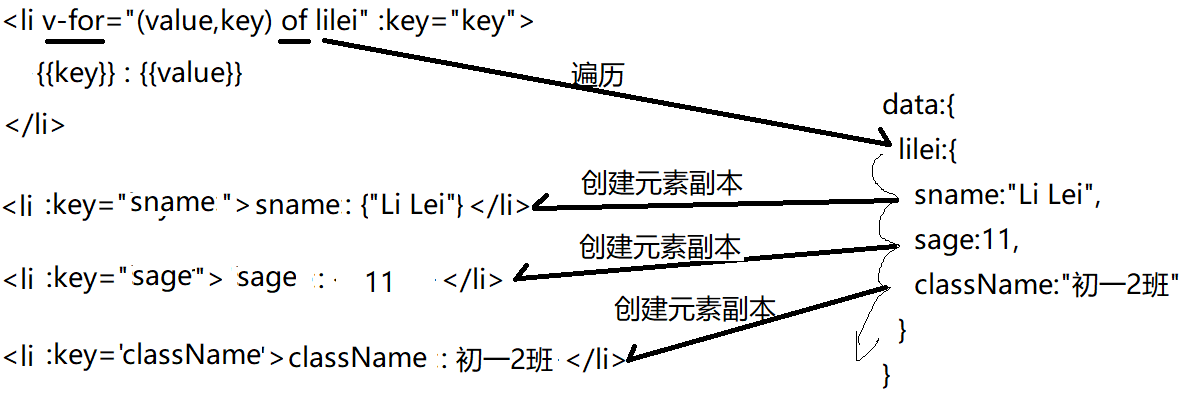
h. 优点: 从此每个元素副本都有一个唯一的标识，当数组中某个位置的元素被删除时，v-for只要根据下标找到对应key值的一个元素对象，删掉即可，其余元素不受影响——效率高！



i: 高频笔试题: v-for为什么必须绑定:key:

答: 如果不绑定:key，则每删除数组中一个元素，v-for都会删除所有元素副本，重建整个列表——修改效率低。绑定:key是给每个元素副本添加一个唯一的标识，在删除数组元素时，可根据唯一标识找到对应的一个元素对象，删除即可，其它元素不受影响——修改效率高！

j: 其实v-for不但可以遍历数字下标的数组，还可遍历自定义下标名称的对象或关联数组。



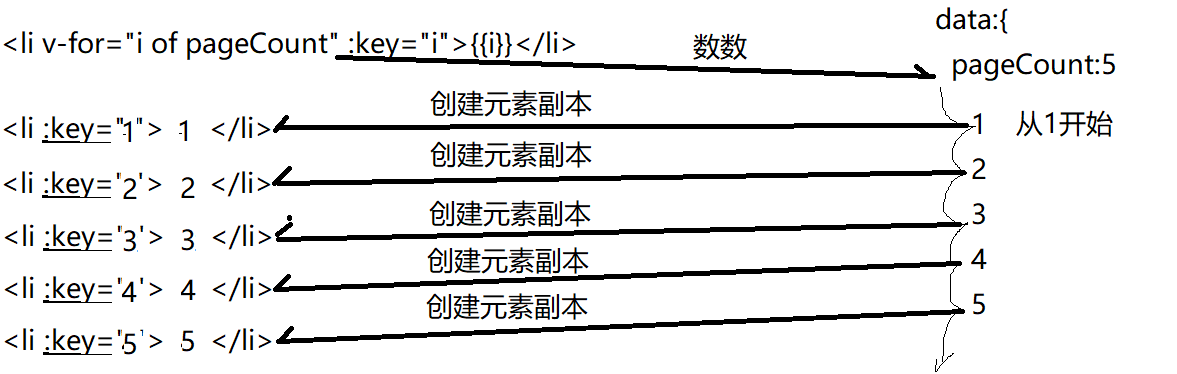
k. 示例: 遍历对象中每属性，在页面创建属性列表

7\_v-for\_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>  </head>  <body>    <!--1. 做界面-->    <div id="app">      <!--想显示一个学生对象的个人信息        用v-for去遍历lilei对象中每属性        自定义的key变量，接住正在遍历的属性的属性名        自定义的value变量，接住正在遍历的属性的属性值        因为对象中，属性名一定是不重复的！所以:key应该绑定不重复的属性名-->      <ul>        <li v-for="(value,key) of lilei" :key="key">{{key}} : {{value}}</li>      </ul>    </div>    <script>      //2. 创建new Vue()对象，监视id为app的区域      new Vue({        el:"#app",        //3. 创建模型对象，保存页面所需的所有变量        data:{          lilei:{            sname:"Li Lei",            sage:11,            className:"初一2班"          }        }      })    </script>  </body>  </html>  运行结果: |

l: 其实: v-for还会**数数**，v-for可根据给定的一个数字，从1开始依次生成指定数量的元素副本。

**<元素 v-for="i of 整数">**



m: 示例: 使用v-for懂啊提生成一组分页按钮

8\_v-for\_n.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>      ul{ list-style:none; }      ul>li{        float:left;        padding: 5px 10px;        border:1px solid #aaa      }      ul>li~li{        border-left:0;      }    </style>  </head>  <body>    <!--1. 做界面-->    <div id="app">      <!--想生成一组分页按钮，根据程序中总页数(pageCount)变量值动态生成指定数量的分页按钮-->      <ul>        <li v-for="i of pageCount" :key="i">{{i}}</li>      </ul>    </div>    <script>      //2. 创建new Vue()对象监控id为app的区域      new Vue({        el:"#app",        //3. 创建模型对象，保存界面所需的所有变量        data:{          pageCount:5        }      })    </script>  </body>  </html>  运行结果: |

(6). v-on:

对应小程序视频: 小程序->在线->VUE->day02 1. 事件绑定 v-on @...

a. 什么是: 专门为元素绑定事件处理函数的指令

b. 何时: 今后只要为元素绑定事件处理函数都用v-on

c. 如何: 标准:

1). HTML中: **<元素 v-on:事件名="处理函数(实参值,...)">**

2). new Vue()中methods中: **处理函数(形参变量,...){ ... }**

d. 简写: **<元素 @事件名="处理函数">**

1). v-on: 可用@代替

2). 如果事件处理函数不需要传入实参值，则()可省略！

e. 需求1: 希望触发事件处理函数时可以传入实参值

1). <元素 @事件名="处理函数(实参值, ...)">

methods:{

处理函数(形参变量, ...){ ... }

}

2). 示例: 点哪个div，就喊哪个div疼！

9\_v-on.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>      #d1,#d2{        width:400px;        height:300px;      }      #d1{        background-color:#afa      }      #d2{        background-color:#aaf      }    </style>  </head>  <body>    <!--1. 做界面-->    <div id="app">      <!--需求1: 两个div，点哪个div，就让哪个div喊疼!            所以在绑定事件处理函数时，将每个div的名字字符串，提前放入事件处理函数的实参中！-->      <div id="d1" @click="say('亮亮')">成亮</div>      <div id="d2" @click="say('然然')">李然</div>    </div>    <script>      //2. 创建new Vue()对象，监视id为app的区域      new Vue({        el:"#app",        //3. 创建模型对象，保存页面所需的变量和事件处理函数        //没有变量，所以不用写data        //但是有事件处理函数        methods:{          //因为事件绑定时会传入一个实参值，所以定义事件处理函数时，需要定义一个形参变量，准备接将来传入的实参值。          say(ename){            console.log(`${ename} 疼!`);          }        }      })    </script>  </body>  </html>  运行结果: |

f. 需求2: 希望事件发生时，获得**事件对象**——同DOM

1). <元素 @事件名="事件处理函数"> //强调: **只获得事件对象e时，不要加()**

//一旦HTML中加了()，就不再自动传事件对象e了

**methods:{**

// event

// ↓  
 **事件处理函数(e){ ... }** **}**

2). 示例: 点div哪个位置，就喊哪个位置疼

10\_v-on\_e.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>      #d1,#d2{        width:400px;        height:300px;      }      #d1{        background-color:#afa      }      #d2{        background-color:#aaf      }    </style>  </head>  <body>    <!--1. 做界面-->    <div id="app">      <!--需求1: 点在div的某个位置，喊某个位置疼！            所以在绑定事件处理函数时什么都不传入            而是在定义事件处理函数中添加形参e，自动获得事件对象，从而进一步自动获得事件对象中的鼠标位置-->      <div id="d1"**@click="say"**>成亮</div>      <div id="d2" **@click="say"**>李然</div>    </div>    <script>      //2. 创建new Vue()对象，监视id为app的区域      new Vue({        el:"#app",        //3. 创建模型对象，保存页面所需的变量和事件处理函数        //没有变量，所以不用写data        //但是有事件处理函数        methods:{          //希望在事件发生时自动获得事件对象e          //event 自动创建          //  ↓ 自动传入  **say(e){**  **console.log(`x:${e.offsetX},y:${e.offsetY}位置疼!`);**          }        }      })    </script>  </body>  </html>  运行结果: |

g.需求3: 笔试题: **如何既传入自定义实参值，又获得事件对象**

1). 错误做法:

<元素 @事件名="处理函数(实参值,...)" //只要HTML中写()

methods:{

~~event~~

~~↓~~

处理函数(e) //事件发生时就不会自动传入event了

}

2). 正确做法:

DOM event

↓

**<元素 @事件名="处理函数(实参值, $event,...)">**

**methods:{**

**处理函数(形参变量, e, ...){ ... }**

**}**

其中: **$event**是vue框架内置的一个关键字(**不能改名**)，先于事件处理函数获得event对象，包装起来备用。在HTML中传入$event时，和其它实参值**没有必然的顺序要求！**

3). 示例: 点哪个div的哪个位置，就喊哪个div的哪个位置疼

11\_v-on\_$event.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>      #d1,#d2{        width:400px;        height:300px;      }      #d1{        background-color:#afa      }      #d2{        background-color:#aaf      }    </style>  </head>  <body>    <!--1. 做界面-->    <div id="app">      <!--需求3: 点在哪个div的某个位置，喊谁的某个位置疼！            既要传入人名，又要传入事件对象-->      <!--                  写死字符串 关键字-->      <div id="d1" @click="say('亮亮',$event)">成亮</div>      <div id="d2" @click="say('然然',$event)">李然</div>    </div>    <script>      //2. 创建new Vue()对象，监视id为app的区域      new Vue({        el:"#app",        //3. 创建模型对象，保存页面所需的变量和事件处理函数        //没有变量，所以不用写data        //但是有事件处理函数        methods:{          //希望在事件发生时同时获得自定义实参值和事件对象          say(ename, e){            console.log(`${ename}的x:${e.offsetX},y:${e.offsetY}位置疼!`);          }        }      })    </script>  </body>  </html>  运行结果: |

(7). **防止用户短暂看到{{}}**

对应小程序视频: 小程序->在线->VUE->day02 2. 其它指令 绑定HTML内容 防止用户短暂看到{{}}

a. 问题: 当网速慢的时候，new Vue()所在的js文件，可能下载延迟，就会让用户短暂看到HTML中的{{}}语法.

b. 解决: 2种:

1). **v-cloak**指令: //幕布、隐身斗篷

i. 什么是: 专门在new Vue()下载完成前暂时用于隐藏元素的特殊指令

ii. 何时: 今后所有用{{}}绑定的位置，为了防止用户短暂看到{{}}语法，都应该用v-cloak暂时隐藏

iii. 如何: 2步:

①先在网页的**style中用属性选择器为所有v-cloak的元素添加display:none属性**

<style>

/\*选择所有带有v-cloak属性的元素\*/

**[v-cloak]{** //隐身斗篷

**display:none;**

**}**

</style>

②再为**要临时隐藏的元素添加v-cloak属性**

iv. 原理:

①在new Vue()加载出来之前，[v-cloak]{display:none}发挥作用，找到所有带有v-cloak指令的元素，让他们暂时隐藏

②在new Vue()加载出来之后，new Vue()会自动找到所有v-cloak属性，并删除他们。结果: 原来由于v-cloak属性而隐藏的元素，才显示出来！

v. 问题: 被迫不但要写HTML和js，还要记得去写css，繁琐

2). **v-text:**

i. 什么是: 专门**代替{{}}**语法来绑定元素内容的特殊指令

ii. 为什么: {{}}有可能导致用户短暂看到{{}}语法。如果不使用{{}}，而改为使用元素开始标签中的v-text属性。则就算属性值没有加载出来，用户也不可能看到元素开始标签中的属性。

iii. 何时: 其实为了避免用户短暂看到{{}}语法，也可以用v-text来解决

iv. 如何: **<元素 v-text="变量或js表达式"> </元素>**

v. 原理: 当new Vue()扫描到v-text时，会先计算""中的变量值或表达式结果，然后用变量值或表达式的结果，覆盖元素开始标签到结束标签之间的内容

vi. 问题: 如果元素的内容需要写死的部分字符串和动态生成的部分字符串拼接而成！

解决: 那么，就不得不用模板字符串！而不能用{{}}

<元素 v-text="`xxxxxx${变量或js表达式}xxxxx`">

vii. 问题: ""里再套反引号，写法太繁琐！

解决: 其实有简写: 外围的""可省略，只写反引号``即可

**<元素 v-text=`xxxxxx${变量或js表达式}xxxxx`>**

viii. 问题: 在元素的属性中使用模板字符串拼接，可读性差

c. 以上两种情况都有问题: 根据个人喜好用哪个都行！

d. 示例: 分别使用v-cloak和v-text防止用户短暂看到{{}}

12\_v-cloak\_v-text.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>      /\*选择所有带有v-cloak属性的元素\*/      [v-cloak]{        display:none;      }    </style>  </head>  <body>    <!--1. 先做界面-->    <div id="app">      <h3 v-cloak>用户名: {{uname}}</h3>      <h3 v-text=`积分:${score}`></h3>    </div>    <script>      //假设网络延迟，new Vue()所在的js文件延迟2s下载      setTimeout(function(){        //2. 创建new Vue()对象，监视id为app的区域        new Vue({          el:"#app",          //3. 创建模型对象，保存界面所需的变量          data:{            uname:"dingding",            score:3000          }        })      },2000);    </script>  </body>  </html>  运行结果: |

|  |
| --- |
| 总结: this 8种: 判断this，~~一定不要看定义在哪儿！~~只看调用时！  1. obj.fun() **this->obj**  2. fun() 或 (function(){ ... })() 或 多数回调函数 或 定时器函数 **this->window**  3. new Fun() **this->new正在创建的新对象**  4. 类型名.prototype.共有方法=function(){ ... }  **this->将来谁调用指谁**，同第一种情况  5. 箭头函数中的this->**箭头函数外部作用域中的this**  6. DOM或jq中事件处理函数中的this->**当前正在触发事件的DOM元素对象**  如果需要使用简化版函数，必须$(this)  7. jQuery.fn.自定义函数=function(){ ... }  **this->将来调用这个自定义函数的.前的jQuery子对象**，~~不用再$(this)~~  8. new Vue()中methods中的函数中的this->当前**new Vue()对象** |

总结:

1. MVVM: 界面View+模型Model+视图模型ViewModel

2. Vue绑定原理: **访问器属性+虚拟DOM树**

变量被修改时: 访问器属性发出通知，虚拟DOM树扫描并仅更新受影响的元素

3. 虚拟DOM树优点:

(1). 小: 只包含可能变化的元素。

(2). 遍历查找快

(3). 修改效率高: 只修改受影响的元素。

(4). 避免重复编码: 已封装DOM增删改查代码

4. Vue功能3步:

(1). **先创建增强版的界面:**

a. 整个界面必须包含在一个唯一的父元素下:

通常是<div id="app">  
b. 可能变化的元素内容用{{自定义变量名}}标记

c. 触发事件的元素用@click="自定义处理函数名"标记

(2). **再创建new Vue()对象**,其中**el:指向new Vue()要监控的页面区域**

(3). **在new Vue()对象内定义模型对象data和methods**

a.界面所需的所有变量都放在data中

b.界面所需的所有事件处理函数都放在methods中

总结: 绑定语法+13种指令

1. 如果元素的内容需要随变量自动变化:  **{{}}**

2. 如果元素的**属性值**需要随变量自动变化:  **:**

3. 控制一个元素显示隐藏: **v-show**

4. 控制两个元素二选一显示:  **v-if v-else**

5. 多个元素多选一显示: **v-if v-else-if v-else**

6. 只要反复生成多个相同结构的元素组成列表时:

**v-for** **:key="唯一标识"**

7. 只要绑定事件: **@ $event**

8. 防止用户短暂看到{{}}: **v-cloak**和**v-text**

今日对应小程序视频列表:

小程序->在线->VUE->day01 3. 绑定语法 {{}} v-bind ...v-show...v-if...

4. v-for 遍历数组 ...

小程序->在线->VUE->day02 1. 事件绑定 v-on @...

2. 其它指令 绑定HTML内容 防止用户短暂看到{{}}

作业:

1. 复习今日小程序问题清单: 小程序->首页->VUE->day01、day02

2. 看小程序视频预习明日内容:

小程序->在线->VUE->day02 2、3、4、5、6、7、8、9、10

3. (学有余力)看小程序视频用vue实现学子商城首页和详情页部分功能: (持续到周四)

小程序->在线->VUE->day01 作业: 使用vue绑定学子商城首页 ...

下载网盘中public\_start文件夹

将今日上课素材day01中js文件夹里vue.js，拷贝到public\_start下的js文件夹下

在public\_start/js文件夹下新建index.js文件

在public\_start/js文件夹下新建details.js文件

然后, 在index.html页面body内，底部，引入vue.js和index.js

然后, 在product\_details.html页面body内，底部，引入vue.js和details.js

然后，看视频，在index.js和details.js中添加自定义代码

使用ajax请求的接口改为: http://xzserver.applinzi.com/index

http://xzserver.applinzi.com/details

网盘中public压缩包是最终结果，如果发现错误，可以参考