我为大家准备了: 微信小程序——“WEB问题速查”——每天必须跟着看

1. 历届师兄师姐的问题清单: 小程序->首页->VUE->day01~day02 点击右侧列表中的问题，就可弹出窗口，查看问题的详情

2. 当天重点视频片段: 小程序->在线->VUE -> day01~day02

3. 当天的单词列表: 小程序->单词-> VUE -> day01~day02

如果tmooc上的视频有问题看不了，可以直接看小程序中的视频

小程序中内容到鄙视面试前看完就行！不用现在都看完！

关于提问: (剩4天)直接私信我是最快的解决途径——谁问的问题越多，谁的培训费交的越值

1. 上课时哪个问题没听明白，立刻问

2. 笔记中哪句话/哪个图不明白，立刻问

3. 小程序中问题清单里哪个问题看不懂，立刻问

4. 如果个人项目中功能不会做，立刻问

5. 刷笔试题时，遇到不会的笔试题，立刻问

约定: 每个重要的案例，至少抄三遍，直到自己可以写出注释才算自己会。（个别代码写不出来，都没关系！）

正课:

1. 指令

一. 指令(directive): 为HTML元素添加新功能的特殊属性——13个

1. v-bind:

(1). 问题: 如果元素的属性值可能随变量自动变化，则不能用{{}}标记！

(2). 解决:

a. 标准: <元素 v-bind:属性名="js表达式">

b. 原理: new Vue()在扫描页面时，只要发现v-bind，就会自动计算=""中的js表达式，并将计算结果自动设置为当前属性的属性值

c. 简写: <元素 :属性名="js表达式">

d. 强调: 如果属性名前加了:，则=后""中不需要再写{{}}，""就充当了{{}}的作用！原来{{}}中能写什么，""中就能写什么！

(3). 总结: 今后只要元素的属性值可能随变量自动变化时，都用:属性名="js表达式"方式来绑定，不能用{{}}。

(4). 示例: 根据空气质量，动态显示不同的表情

|  |
| --- |
| <!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>    <!--根据空气质量pm2.5的数值，更改页面中显示不同图片-->    <!--1. 先定义界面    问题: 页面中哪里可能随变量自动变化    本例中: img元素的src属性随变量自动变化,所以需要用特殊标记标出来-->    <div id="app">      <h3>{{pm25<100?'img/1.png':            pm25<200?'img/2.png':            pm25<300?'img/3.png':                    'img/4.png'}}</h3>      <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，指定el属性指向要监控的元素区域      new Vue({        el:"#app",        //3. 定义模型对象        //问题: 页面上共需要几个变量？        //本例中: 只需要一个pm25变量        //问题: 因为pm25只是一个数值，不是图片路径        //解决: 只要变量的数值不能直接使用，都可以在HTML中{{}}里对数值进行加工！        data:{          pm25:70        }      })    </script>  </body>  </html>  运行结果: |

说明：红色框圈住的部分不是错误！只是vue的一个一般性提示。不用关注。

2. v-show:

(1). 需求: 控制一个元素的显示隐藏:

(2). 如何: <元素 v-show="返回bool类型的js表达式"

(3). 原理:

a. new vue()扫描到v-show时，就会先自动计算=后的js表达式的值，得到一个bool值。 B. 如果执行结果为true则，当前元素什么都不干，默认显示！

c. 如果执行结果为false则，当前元素自动添加display:none，隐藏

(4). 示例: vue实现弹出对话框

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <style>      \*{margin:0; padding:0}      .alert{        width:300px;        height:100px;        border:1px solid #555;        position:fixed;        top:50%;        left:50%;        margin-left:-150px;        margin-top:-50px;      }      .alert>p{        text-align:right;      }    </style>    <script src="js/vue.js"></script>  </head>  <body>    <!--1. 做界面      问题: 哪里可能发生变化      本例中: 只有div.alert元素的显示隐藏会发生变化      所以: 在div.alert元素上添加v-show="show"      意为: 用一个变量show的值，来控制div.alert是显示还是隐藏      问题: 点按钮，让对话框显示，点x可让对话框隐藏      本例中: 为button按钮绑定单击事件，当单击时，自动执行login函数。为a按钮绑定单击事件，当单击时，自动执行close函数-->    <div id="app">      <button @click="login">登录</button>      <div class="alert" v-show="show">        <p><a href="javascript:;" @click="close">×</a></p>        用户名:<input><br/>        密码:<input type="password"><br/>        <button>登录</button>      </div>    </div>    <script>      //2. 创建new Vue()，监控id为app的元素区域      new Vue({        el:"#app",        //3. 定义对象data保存页面上所需的所有变量        //问题: 页面上共需要几个变量?        //本例中: 页面上只需要一个变量show,且因为控制显示隐藏，所以变量show的值应该是一个bool值！        data:{          show:false //希望div.alert开始时默认隐藏        },          //问题: 页面上共需要几个事件处理函数?        //本例中: 页面上需要2个事件处理函数login和close。        //login让div.alert显示，就要将show变量值改为true        //close让div.alert隐藏，就要将show变量值改为false        //强调: methods中的事件处理函数只要专心操作变量值就够了！无需管页面上如何修改！        methods:{          login(){ this.show=true },          close(){ this.show=false }        }      })    </script>  </body>  </html>  运行效果： |

3. v-if和v-else:

(1). 需求: 两个元素二选一显示

(2). 如何:

<元素1 v-if="返回bool值的表达式">

<元素2 v-else>

(3). 强调:

a. v-if和v-else两个元素必须紧挨着，中间不能插入其他元素

b. v-else后不要加任何=，和程序中if else的else后不加表达式道理一样。

(4). 原理:

a. new Vue()扫描到v-if时，先自动计算=后的表达式的值

b. 如果v-if后表达式的值为true，则保留v-if所在的元素1，删除v-else所在元素2

c. 如果v-if后表达式的值为false，则删除v-if所在的元素1，保留v-else所在元素2

(5). v-show vs v-if 差别: （高频笔试题）

a. v-show通过display:none方式控制显示隐藏，因为不修改DOM树，所以效率略高

b. v-if通过删除元素方式控制显示隐藏，所以要修改DOM树，效率略低。

(6). 示例: 实现登录和注销状态切换

|  |
| --- |
| <!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>    <!--想实现登录状态的切换-->    <!--1. 定义界面      问题: 页面中哪里可能发生变化      本例中: 两个div交替显示隐藏      所以: 用v-if和v-else，并且用一个变量isLogin来记录用户是否登录。如果isLogin值为false，说明用户未登录，则只显示第一个div。如果isLogin值为true，说明用户已登录，则只显示第二个div。      本例中: 点登录按钮，切换为已登录状态，点注销按钮，切换为未登录状态。所以需要给登录按钮和注销按钮都绑定单击事件-->    <div id="app">      <div v-if="isLogin==false"><a href="javascript:;" @click="login">登录</a> | <a href="javascript:;">注册</a></div>      <div v-else>Welcome dingding, <a href="javascript:;" @click="logout">注销</a></div>    </div>    <script>      //2. 创建new Vue()对象指向id为app的元素区域      new Vue({        el:"#app",        //3. 定义模型对象保存界面所需的所有变量和事件处理函数        //问题: 界面中共需要几个变量        //本例中: 界面中只需要一个isLogin变量，表示用户是否登录        data:{          isLogin:false,//开始时，用户默认是未登录        },            //本例中: 界面中共需要几个事件处理函数        //login() 将isLogin改为true，说明已登录        //logout() 将isLogin改为false，说明未登录        methods:{          login(){ this.isLogin=true },          logout(){ this.isLogin=false }        }      })    </script>  </body>  </html>  运行效果： |

4. v-else-if:

(1). 需求:多个元素多选一显示

(2). 如何:

<元素1 v-if="返回bool值的表达式">

<元素2 v-else-if="返回bool值的表达式">

... ...

<元素n v-else>

(3). 强调:

a. v-if、v-else-if和v-else多个元素必须紧挨着，中间不能插入其他元素

b. v-else后不要加任何=，和程序中if else的else后不加表达式道理一样。

(4). 原理:

a. new Vue()扫描页面时，如果遇到v-if就先执行=后的表达式，获得bool值结果

b. 如果v-if表达式的值返回true，则保留v-if，删除其余v-else-if和v-else的元素

c. 如果v-if表达式的值返回false，则先后继续执行后续v-else-if的表达式。只要碰到一个v-else-if表达式的值为true，则仅保留着个v-else-if的元素，删除其余v-if v-else-if和v-else

d. 如果之前的v-if和v-else-if的表达式值都为false，才保留最后一个v-else元素，删除其余v-if和v-else-if的元素

(5). 示例: 根据pm25数值，显示不同表情

|  |
| --- |
| <!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>    <!--想实现根据pm25的数值显示不同的表情-->    <!--1. 定义界面      问题: 页面上哪些地方会发生怎样的变化      本例中: 4个img根据条件多选一显示      应该用v-if,v-else-if,v-else，并且用一个变量pm25控制应该显示那个img-->    <div id="app">      <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，指向id为app的元素区域      new Vue({        el:"#app",        //3. 定义模型变量        //本例中: 只需要一个pm25数值就能控制页面上的显示隐藏        data:{          pm25:75          }      })    </script>  </body>  </html>  运行效果： |

5. v-for:

(1). 需求: 根据数组中的内容反复生成多个相同结构的元素

(2). 如何:

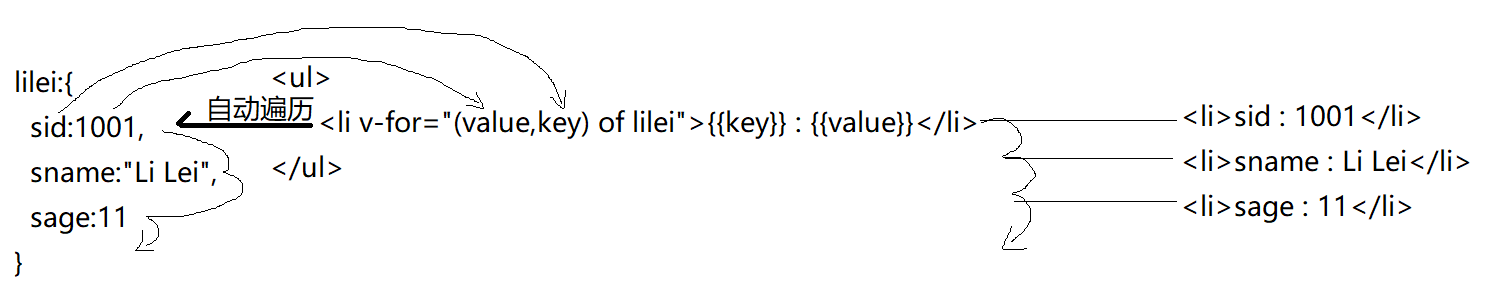
<要反复生成的元素 v-for="(value,i) of 数组/对象">

(3). 强调:

a. 要反复生成的元素，只需要写一个当做模板即可！

b. v-for一定要写在要反复生成的元素上，~~而不是要反复生成的元素的父元素上。~~

c. v-for of，即可遍历数组，又可以遍历对象——统一了for in和for of

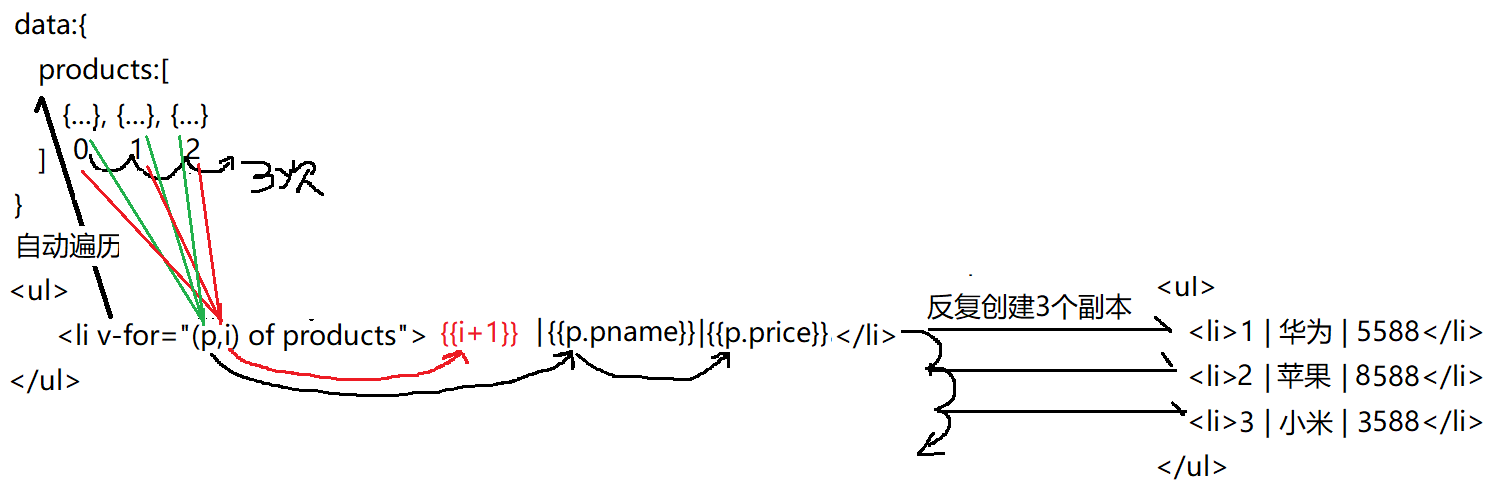


(4). 原理:

a. new Vue()扫描到v-for，会自动遍历of后的数组或对象

b. 每遍历到数组中一个元素或对象中一个成员，都会将当前HTML元素自动创建一个新的副本。

c. 在要反复生成的元素身上或其子元素中，可以将v-for的两个变量value和i用于动态绑定内容。其中: value变量会自动获得当前正在变量的元素值或属性值。i会自动获得当前数组元素的下标或对象成员的属性名。



(5). 强调: 凡是用v-for时，必须同时绑定一个:key属性

a. 问题: 因为v-for每次反复生成的元素之间除了内容之外，单看元素是无差别的。即使将来只更改了数组中一个元素值，因为v-for无法区分反复生成的多个元素，就无法只精准更新一个元素。V-for只能采用笨办法，将v-for负责的所有元素重新生成一遍。——效率极低

b. 解决: 让v-for反复生成的每个元素都有一个唯一的标识。如果数组中某一个元素发生改变，v-for可凭借这唯一的标识找到一个页面元素，只更新一个页面元素即可。其余页面元素保持不变——效率高！

c. 如何: <要反复生成的元素 v-for="(值,下标) of 数组或对象" :key="下标"

(6). 示例: 使用v-for遍历数组和对象

|  |
| --- |
| <!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">      <ul>        <li v-for="(p,i) of products" :key="i">{{i+1}} | {{p.pname}} | {{p.price}}</li>      </ul>      <ul>        <li v-for="(value,key) of lilei" :key="key">{{key}} : {{value}}</li>      </ul>    </div>    <script>      new Vue({        el:"#app",        data:{          products:[            {pname:"华为", price:5588},            {pname:"苹果", price:8588},            {pname:"小米", price:3588},          ],          lilei:{            sid:1001,            sname:"Li Lei",            sage:11          }        }      })    </script>  </body>  </html>  运行效果: |

(7). V-for还会数数！

a. <要反复生成的元素 v-for="i of 一个整数" :key="i">

b. 原理:

1). V-for从1开始循环，每次递增1，到of后给的整数值结束.

相当于for(var i=1;i<=一个整数;i++){ ... }

2). 每循环一次就将当前元素创建一个新的副本

3). 变量i会依次接住每次+1后的新值。变量i可用于元素内容中的绑定语法。

c. 示例: 根据页数，生成分页按钮

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <style>      .pages{        list-style:none      }      .pages>li{        float:left;        padding:5px 10px;        border:1px solid #555      }      .pages>li~li{        border-left:none;      }      .pages>li:first-child{        border-radius:5px 0 0 5px;      }      .pages>li:last-child{        border-radius:0 5px 5px 0;      }    </style>    <script src="js/vue.js"></script>  </head>  <body>    <!--需求: 根据变量pageCount反复生成指定数量的分页按钮-->    <div id="app">      <ul class="pages">        <li>上一页</li>        <li v-for="i of pageCount" :key="i">{{i}}</li>        <li>下一页</li>      </ul>    </div>    <script>      new Vue({        el:"#app",        data:{          pageCount:8 //共有8页数据        }      })    </script>  </body>  </html>  运行效果: |

6. v-on: 事件绑定:

(1). 如何:

a. 标准: <元素 v-on:事件名="处理函数名(实参值)">

b. 简写:

1). <元素 @事件名="处理函数名(实参值)"

2). 如果处理函数不需要实参值，则()可省略: <元素 @事件名="处理函数名"

c. 页面所需的所有事件处理函数都应该集中定义在methods对象中

(2). 示例: 点哪个div，喊哪个div疼

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <style>      #app>div{        width:300px;        height:150px;      }      #d1{        background-color:red      }      #d2{        background-color:green;      }    </style>    <script src="js/vue.js"></script>  </head>  <body>    <div id="app">      <div id="d1" @click="say('d1')"></div>      <div id="d2" @click="say('d2')"></div>    </div>    <script>      new Vue({        el:"#app",        methods:{          //dname形参可接受点击div时传过来的字符串div名          say(dname){            alert(`${dname} 疼!`)          }        }      })    </script>  </body>  </html>  运行结果: |

(3). Vue中如何获得事件对象: （Vue中如何获得鼠标点击位置）

a. 如果只获取事件对象，不需要传入其他实参值时: 同DOM

<元素 @事件名="处理函数">

methods:{

//自动创建event对象

处理函数(e){

//e得到的事件对象和DOM中的事件对象完全一样！

}

}

示例: vue中获得鼠标点击位置

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <style>      #app>div{        width:300px;        height:150px;      }      #d1{        background-color:red      }      #d2{        background-color:green;      }    </style>    <script src="js/vue.js"></script>  </head>  <body>    <div id="app">      <div id="d1" @click="say"></div>      <div id="d2" @click="say"></div>    </div>    <script>      new Vue({        el:"#app",        methods:{          //当单击div时，显示鼠标点击的位置          //自动创建event,自动传入事件处理函数第一个参数          //        ↓          say(      e    ){            alert(`x:${e.offsetX}, y:${e.offsetY}`)          }        }      })    </script>  </body>  </html>  运行效果: |

b. 既想用事件对象，又要传入自定义实参值：（笔试题）

1). 错误做法1:

<元素 @事件名="处理函数(实参值)">

methods:{

处理函数(形参1, e){

//虽然实参可以传进来，但是e就无法获得了！

}

}

2). 错误做法2:

<元素 @事件名="处理函数(实参值)">

methods:{

//实参放在第一个位置，形参却定义在第二个位置！位置就不对！

处理函数(e, 形参1){

}

}

3). 正确: 手动使用vue提供的$event关键词代替e，手动传参

i. $event: vue自动封装的保存DOM事件对象的一个特殊关键词，可用于传参！

ii. 如何:

//当事件发生时, vue会自动封装"DOM事件对象e"

<元素 @事件名="处理函数($event, 实参值)">

methods:{

处理函数(e, 形参1){

//e得到的就是DOM中的事件对象，用法也和DOM中一样

}

}

强调: 处理函数中$event与实参值的位置没有固定顺序，但是必须和methods中处理函数定义中的形参列表顺序保持一致！

4). 示例: 点击按钮，既获得div名字，又获得鼠标点击位置

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <style>      #app>div{        width:300px;        height:150px;      }      #d1{        background-color:red      }      #d2{        background-color:green;      }    </style>    <script src="js/vue.js"></script>  </head>  <body>    <div id="app">      <div id="d1" @click="say('d1', $event)"></div>      <div id="d2" @click="say('d2', $event)"></div>    </div>    <script>      new Vue({        el:"#app",        methods:{          say(dname,e){            alert(`点在${dname}的x:${e.offsetX}, y:${e.offsetY}位置`)          }        }      })    </script>  </body>  </html>  运行效果: |

7. v-html:

(1). 问题: 如果要绑定的变量值是一段HTML代码，则使用{{}}绑定，会保持HTML代码的原样，而不经过编译，直接显示在页面上。

(2). 解决: 今后，只要要绑定的变量值是一段HTML代码，则必须用v-html指令代替{{}}来绑定。

(3). 如何: <元素 v-html="变量或js表达式"> </元素>

(4). 优点: v-html会先将变量中的HTML片段交给浏览器解析成可以给人看的东西，然后才替换进元素的内容中。

(5). 示例: 绑定HTML内容

|  |
| --- |
| <!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">      <h3>{{content}}</h3>      <h3 v-html="content"></h3>    </div>    <script>      new Vue({        el:"#app",        data:{          content:`来自<a href="javascript:;">&lt;&lt;新华社&gt;&gt;</a>的消息`        }      })    </script>  </body>  </html>  运行效果: |

8. v-cloak和v-text: 防止用户短暂看到{{}}:

(1). 问题: 当网速慢时，如果js代码还未下载执行，则用户有可能短暂看到页面上的{{}}语法——尴尬！

(2). 解决: 2个办法:

a. v-cloak指令:

1). 原理: 在new Vue()下载执行前，先将所有带{{}}的元素暂时隐藏。等到new Vue()下载执行完，再自动将所有{{}}的元素显示出来。

2). 2步:

i. 先在网页的style中定义一个属性选择器: [v-cloak]{ display:none }。意为所有带有v-cloak属性属性的元素，暂时隐藏

ii. 在有{{}}内容的元素上加上v-cloak属性，不用加=

3). 结果: 所有带有v-cloak的元素，都暂时隐藏。直到new Vue()下载并执行完，自动找到所有v-cloak的元素，删除v-cloak属性。这些元素才显示出来！

4). 示例: 使用v-cloak防止用户短暂看到{{}}

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <style>      /\*凡是带有v-cloak属性的元素暂时都隐藏\*/      [v-cloak]{        display:none      }    </style>    <script src="js/vue.js"></script>  </head>  <body>    <div id="app">      <h1 v-cloak>用户名:{{uname}}</h1>      <h2 v-cloak>积分:{{score}}</h2>    </div>    <script>      setTimeout(function(){        new Vue({          el:"#app",          data:{            uname:"dingding",            score:1200          }        })      },3000);    </script>  </body>  </html>  运行结果: 等3s后出现 |

b. v-text指令:

1). 原理: 既然不想看到{{}}，就可以用一种特殊的指令代替{{}}，功能还一样！因为指令都是写在元素的属性中。即使没有加载完，用户也不可能看到元素的属性！

2). 如何: 1步

i. <元素 v-text="变量或js表达式"> </元素>

ii. 原理: 当v-text的内容计算好之后，会自动将执行后的结果代替元素的内容

3). 问题: 如果元素的内容就是由固定的内容和变化的变量拼接起来的，则被迫使用模板字符串！<元素 v-text="`模板字符串`"></元素>

4). 简写: 如果v-text后跟的是一个模板字符串，则可以省略""，只保留模板字符串的反引号``即可！<元素 v-text=`模板字符串`></元素>

5). 示例: 使用v-text代替防止用户短暂看到{{}}

|  |
| --- |
| <!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">      <h1 v-text=`用户名:${uname}`></h1>      <h2 v-text=`积分:${score}`></h2>    </div>    <script>      setTimeout(function(){        new Vue({          el:"#app",          data:{            uname:"dingding",            score:1200          }        })      },3000);    </script>  </body>  </html>  运行效果：等3s后出现 |

9. v-once:

(1). 问题:有些绑定只在首次加载时，修改一次。之后其他地方的修改，都不会影响这里的显示结果。如果一个元素之后不需要再更新，但是却占着虚拟DOM树中的一个位置，会影响虚拟DOM树的遍历速度！

(2). 优化: 如果只在首次加载时显示，之后其他地方修改，这里都不会受影响的绑定，可用v-once绑定。

(3). 如何: <元素 v-once>{{变量或js表达式}}></元素>

(4). 原理: 只在首次渲染页面时，替换元素的绑定语法内容。但是不会将当前元素加入虚拟DOM树。结果: 只在首次加载一次，之后都不会再改变。

(5). 示例: 使用v-once绑定页面加载时间:

|  |
| --- |
| <!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">      <h1 v-once>页面加载完成时间:{{new Date(timer).toLocaleString()}}</h1>      <h1>当前系统时间:{{new Date(timer).toLocaleString()}}</h1>    </div>    <script>      var vm=new Vue({        el:"#app",        data:{          timer:new Date().getTime() //获得当前系统时间的毫秒数        }        //get timer(){}        //set timer(value){}      })      //因为data中的变量进入new Vue()后会被自动请保镖，且保镖直接隶属于new Vue()对象，所以其实在new Vue()外部也可以用vm.属性名方式，修改data中的变量值。vm.timer      //每隔1秒钟，修改new Vue()对象中的timer为当前系统时间新值      setInterval(function(){        vm.timer=new Date().getTime();      },1000)      //因为timer是访问器属性，内置通知机制。所以，即使在new Vue()外部修改timer属性，也会触发页面自动更新！    </script>  </body>  </html>  运行结果: |

10. v-pre:

(1). 极端的问题: 如果元素的内容正文中刚好也包含{{}}，但是不想被vue编译，而是原样显示！

(2). 如何:<元素 v-pre> ... {{xxx}}...</元素>

(3). 示例: 防止内容中的{{}}被vue编译

|  |
| --- |
| <!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">      <h1 v-pre>Vue框架中采用{{变量名}}语法动态绑定元素的内容</h1>    </div>    <script>      new Vue({        el:"#app"      })    </script>  </body>  </html>  运行效果： |

总结:

1. MVVM:

(1). 界面(View) 增强版，支持变量、分支、循环等功能

(2). 模型对象(Model) 保存界面所需的一切变量和函数:

a. data对象: 专门保存界面所需的所有变量

b. methods对象: 专门保存界面所需的所有事件处理函数

(3). 视图模型(ViewModel) 快递员 将data和methods中的变量值和函数自动送到界面中所需的位置。并始终保持同步！

2. Vue绑定原理:

(1). 简单: 访问器属性+虚拟DOM树

(2). 完整: 访问器属性+观察者模式+新DOM树

3. 虚拟DOM树:

(1). 什么是: 仅保存可能变化的DOM元素的简化版DOM树  
 (2). 优点: 4个

a. 小: 仅保存可能变化的DOM元素

b. 快: 因为小，所以遍历快

c. 减少重复代码: 自动对受影响的元素执行增删改查操作

d. 更新效率高: 只更新受影响的元素。不受影响的元素，保持不变的！

4. Vue功能: 3步——招数

(1). 编写界面: 要求:

a. 整个界面必须包裹在一个唯一的父元素内，且父元素习惯上id="app"

b. 所有可能发生变化的地方，用{{变量名}}方式标记

c. 所有触发事件的元素，用@事件名="函数名"标记

(2). 先创建new Vue()对象：

new Vue({

el:"#app", //选择器

... ...

})

(3). 为new Vue()添加data和methods模型对象

a. 界面中所需的所有变量都放在data对象中

b. 界面中所需的所有函数都放在methods对象中

c. methods中的函数，想操作data中的变量，必须加this.

5. 绑定语法和指令: ——招数

(1). 如果元素的内容可能随变量自动变化:

<元素>{{变量名}}</元素> 或 <元素 v-text="变量或js表达式"></元素>

(2).只要元素的属性值可能随变量自动变化: :属性名="js表达式"

(3). 控制一个元素显示隐藏: v-show="返回bool值的js表达式"

(4). 控制两个元素二选一显示: <元素1 v-if="返回bool值的js表达式">

<元素2 v-else>

(5). 控制多个元素多选一显示：<元素1 v-if="返回bool值的js表达式">

<元素2 v-else-if="返回bool值的js表达式">

... ...

<元素n v-else>

(6). 反复生成多个相同结构的元素：

<要反复生成的元素 v-for="(value,i) of 数组/对象/整数" :key="i">

(7). 绑定事件: <元素 @事件名="处理函数($event, 实参值)">

(8). 绑定原始HTML代码内容: <元素 v-html="变量"></元素>

(9). 防止用户短暂看到{{}}:

<元素 v-cloak>...</元素> 或 <元素 v-text="变量或js表达式"></元素>

(10). 今后只要一个绑定的值，在首次加载之后不再发生变化:

<元素 v-once>{{变量或js表达式}}</元素>

(11). 防止元素内容中的{{}}被new Vue()编译，保持{{}}原样显示:

<元素 v-pre>{{xxx}}</元素>

作业:

1. 复习今日问题清单(VUE->day01~day02),预习明日问题清单(VUE->day02~day03)

2. 看小程序视频学习将绑定语法和指令应用到学子商城项目中:

小程序->在线->VUE->day01 作业: 使用VUE绑定学子商城首页和详情页部分内容...

下载public\_start文件夹，照着视频的做

视频中所有http://localhost:5050地址，全部改为我的新浪云服务器地址: http://xzserver.applinzi.com/接口名

3. 看小程序视频预习双向绑定:

小程序->在线->VUE->day02 3.4.5.6.7双向绑定