提前准备:

1. 今天的历届师兄师姐问过的经典的高频的问题清单都已经整理在小程序->首页->VUE->day02里

2. 今天的课程中所需的单词列表都已经整理在小程序->单词->VUE->day02里

3. 今天的重点视频、扩展高频鄙视题视频已经整理在小程序->在线-> VUE ->day02里

如果不知道问什么: 谁问的越多！谁培训费花的越值！

1. 笔记中或画图中那句话或者那个位置不明白，立刻问！

2. 案例中哪行代码或注释不明白，立刻问！

3. 课下自己做项目，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！——公众号: 前端大全 定期总结大厂笔试题

5. 看小程序问题清单时，遇到看不懂的问题，立刻问！

正课:

1. 指令：

2. 双向绑定:

3. 绑定样式:

4. 自定义指令:

5. 计算属性:

一. 指令: 13个

1. 事件绑定:

(1). 标准写法: <元素 v-on:事件名="处理函数(实参值)"

(2). 原理: v-on:事件名 类似于 HTML中的<元素 on事件名="xxx">，只不过所有事件处理函数，应该定义在methods中。且事件处理函数中必须用"this."来操作data中的变量

(3). 简写:

a. "v-on:" 可简写为@

比如: <元素 v-on:click="xxx()"> 可简写为 <元素 @click="xxx()">

b. 如果事件处理函数不需要传参，则可省略()

比如: <元素 @click="xxx()"> 可简写为 <元素 @click="xxx">

(4). 获得事件处理函数: event

↓

a. 回顾DOM中: 元素.on事件名=function( e ){ ... }

b. vue中: 完全一样！

<元素 @事件名="处理函数()"

methods:{

处理函数(e){

e可获得和DOM一模一样的event对象

比如，可以获得鼠标位置: e.offsetX, e.offsetY

还可以阻止默认行为:e.preventDefault()

还可以获得目标元素: e.target

还可以取消冒泡: e.stopPropagation()

}

(5) 问题: 如果既想传自定义实参值，又想获得事件对象: （高频笔试题）

a. 错误的做法1:

<div id="d1" @click="han('d1')">

methods:{

han(e,name){ //name=undefined

b. 错误的做法2:

<div id="d1" @click="han('d1')">

methods:{

han(name, e){ //e=undefined 因为事件处理函数只能作为第一个实参值传入，无法作为第二个、第三个等实参值传入.

c. 正确: 手动传入一个Vue已经封装好的关键词$event。

// 原生event

// ↓

<div id="d1" @click="han('d1', $event)">

methods:{

han(name, e){

结果: name='d1' e=$event=原生event

(6). 示例: 既传入实参值，又获得事件对象

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <style>      #d1,#d2{        width:300px;        height:100px;      }      #d1{        background-color:#aaf      }      #d2{        background-color:#ffa      }    </style>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">    <div id="d1" @click="han('d1',$event)"></div>    <div id="d2" @click="han('d2',$event)"></div>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      },    methods:{      han(name,e){        alert(`点在${name}的x:${e.offsetX},y:${e.offsetY}位置，疼！`)      }    }  })  </script>  </body> |

2. 防止用户短暂看到{{}}

(1). 问题: 尤其是移动端网路不稳定，极有可能出现js下载比网页延迟的情况。如果js中的new Vue()延迟加载，则HTML中的{{}}就会短暂的被用户看到——尴尬

(2). 解决1: v-cloak

a. 如何:

1). 在内容中含有{{}}语法的元素上添加v-cloak属性: <元素 v-cloak>{{xxx}}</元素>

问题: v-cloak本身毫无隐藏的功能！

2). 必须手动在css中定义一个属性选择器配合v-cloak:

//凡是带有v-cloak的元素暂时隐藏

[v-cloak]{ display:none }

b. 原理:

1). 如果new Vue()加载慢，则v-cloak会暂时让当前元素隐藏

2). 当new Vue()加载完成后，会自动删除元素上的v-cloak属性，被隐藏的元素就重新显示出来

c. 问题: 必须2步，html+css都要写才能实现

(3). 解决2: 用v-text指令，代替{{}}

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

b. 原理: 当new Vue()加载完，会用v-text的内容，代替当前元素的textContent内容

c. 优点: 因为内容中没用{{}}，而且v-text是写在元素属性中的，所以即使new Vue()用户也不可能看到半成品的内容！

d. 缺点: {{}}至少有一个优点，就是可以和其他固定的文字轻松拼接在一起，不用写+或${}。但是v-text后需要的是一个js表达式，在js表达式中如果要实现拼接要么用+，要么用模板字符串。不能直接连着写！比如:

~~<h1 v-text="年龄:{{age}}"></h1>错误: 因为v-text后的""中不是合法的js表达式~~

<h1 v-text="'年龄:'+age"></h1> 或 <h1 v-text="`年龄:${age}`"></h1>

(4). 示例: 分别使用v-cloak和v-text防止用户短暂看到{{}}

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <style>      [v-cloak]{ display:none }    </style>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">    <h1 v-cloak>用户名:{{uname}}</h1>    <h1 v-text="`年龄:${age}`"></h1>  </div>  <script>  setTimeout(function(){    var vm=new Vue({      el:"#app",      data:{        uname:"dingding",        age:11      }    })  },3000)  </script>  </body> |

3. 绑定一段HTML代码片段内容:

(1). 问题: {{}}和v-text虽然都能绑定元素的内容，但是都是以textContent方式填充元素内容。如果要填充的内容是一段HTML片段，则交给textContent后，不会被浏览器解析为人能看的内容，而是保持HTML代码原样显示！

(2). 解决: 今后还要绑定给的内容是一段HTML，都要用v-html代替v-text和{{}}。意为v-html相当于innerHTML，赋值一段HTML代码会先交给浏览器解析后，再显示给人看！

(3). 示例：

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">    <h1>{{msg}}</h1>    <h1 v-text="msg"></h1>    <h1 v-html="msg"></h1>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      msg:`<p>来自<a href="javascript:;">&lt;&lt;新华社&gt;&gt;</a>的消息</p>`    }  })  </script>  </body> |

4. 只在首次渲染页面时绑定一次。之后即使变量值改变，也不会重新渲染元素内容

(1). 如何: <元素 v-once>

(2). 原理:

a. 只在首次渲染页面时，更新页面的内容。

b. 但是该元素并不加入虚拟DOM树，今后即使变量值更新，也不会重新渲染该元素。

(3). 何时: 如果我们确定一个元素的内容只在首次绑定一次，之后无需重复绑定时，都要用v-once修饰。可减少之后绑定的元素个数，提高之后每次页面渲染的效率。

(4). 示例: 使用v-once避免重复渲染元素的内容

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <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: {{now}}</h1>    <h1>当前系统时间: {{now}}</h1>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      now:new Date().toLocaleString()    }  })  setInterval(function(){    vm.now=new Date().toLocaleString()  },1000);  </script>  </body> |

5. 阻止内容中的{{}}被vue编译，保持内容中的{{}}原样显示: v-pre

二. 双向绑定:

(1). 需求: 在vue中如何获得用户在表单元素中输入的内容。

(2). 问题: 普通的:value="变量"这种绑定方式只能把程序中的变量值自动更新到页面上。但是，当页面上更改内容时，这种:的绑定，无法自动将页面的修改更新回程序中对应的变量上。——单向绑定(只能从data中到页面，无法从页面回到data中)

(3). 解决: 今后只要想获得页面中表单元素的值，都应该使用双向绑定: v-model:value="变量"

(4). 什么是双向绑定: 既能将data中的变量值自动更新到页面上，又能将页面上的更改实时更新回data中的变量里

(5). 如何: 用"v-model:"代替":"

<input type="text" v-model:value="keywords"

(6). 原理: 其实v-model会给当前表单元素自动绑定onchange事件，当用户修改了表单元素的值后，会自动触发onchange事件。在onchange事件中自动调用绑定的变量的set()访问器更新data中的变量值。

(7). 示例: 单击按钮，获得文本框中的值:

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">  <input type="text" v-model:value="keywords"><button @click="search">百度一下</button>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      keywords:""    },    methods:{      search(){        console.log(`搜索 ${this.keywords} 相关的内容...`)      }    }  })  </script>  </body> |

(8). 各种不同表单元素的双向绑定方式:

a. 用户输入值，值保存在元素value属性中:

1). 普通文本框: <input type="text" v-model:value="变量">

2). 密码框: <input type="password" v-model:value="变量">

3). 文本域: <textarea v-model:value="变量"></textarea>

b. 单选按钮:

1). 特点: 在定义radio时，就已经定死了value，供用户选择，而不是随机输入。

2). 如何绑定: 应该绑定checked属性，因为只有checked属性才是变化的.

<input type="radio" value="固定值" name="分组" v-model:checked="变量">

3). 原理:

i. 从data中到页面绑定时: 用变量值和value属性的值做比较。如果变量的值等于value属性的值，则当前radio选中。否则如果变量的值和value属性的值不相等，则当前radio不选中！

ii. 如果用户在页面上改变了radio的选中状态，则如果选中该radio，则自动将value属性的值反向更新回data中的变量中。

4). 示例: 选择性别

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">    性别:    <label><input type="radio" value="1" name="sex" v-model:checked="sex">男</label>    <label><input type="radio" value="0" name="sex" v-model:checked="sex">女</label>    <h3>您选的是:{{sex}}</h3>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      sex:0    }  })  </script>  </body> |

c. select元素:

1). 特点:

i. 所有的备选的value都写死在每个备选项option上

ii. 当用户选择其中某个一option时，这个option的value又会成为整个select元素的value。

2). 如何:

<select v-model:value="变量" >

<option value="值1">xxx</option>

<option value="值2">xxx</option> ←当用户选择一个option后

... ...

3). 原理:

i. 从data中到页面绑定时: 用data中变量的值和每个option的value做比较。哪个option的value等于data中变量的值，则当前option选中！

ii. 当用户切换选择不同的option时，会将当前选中的option的value保存回data中变量的值中。

4). 示例: 使用select选择不同城市的图片

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">    <select v-model:value="city">      <option value="bj">北京</option>      <option value="sh">上海</option>      <option value="hz">杭州</option>    </select>    <h3>您选的城市是:{{city}}</h3>    <img :src="`img/${city}.jpg`" alt="">  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      city:"bj" //保存选中的city的value    }  })  </script>  </body> |

d. 复选框checkbox（单独使用）:

1). 特点: 没有value，单纯切换选中与不选中，所以v-model只要绑定checked属性即可。且属性值应该为bool类型

2). 如何: <input type="checkbox" v-model:checked="变量">

3). 原理:

i. 从data中到界面中绑定时: 变量值为true，checkbox就选中，变量值为false，checkbox就不选中

ii. 当用户切换checkbox的选中状态时: 会自动将checkbox新的checked状态更新回data中的变量中

4). 示例: 根据是否同意启用禁用表单元素

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">    <input type="text" placeholder="请输入用户名" :disabled="isAgree==false"><br/>    <input type="password" placeholder="请输入密码" :disabled="isAgree==false"><br/>    <label><input type="checkbox" v-model:checked="isAgree">同意本网站的管理条款</label><br/>    <button :disabled="isAgree==false">注册</button>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      isAgree:false, //记录是否同意    }  })  </script>  </body> |

(9). 简写：其实以上案例只写v-model即可，不用写具体绑定哪个属性。因为v-model可自动根据所在的元素判断应该绑定哪个属性。

(10). 监视函数:

a. 什么是监视函数: 只要一个变量变化，就会自动调用的函数

b. 何时: 今后只要一个变量变化，希望立刻执行一项操作时

c. 如何:

new Vue({

el:"#app", //要监控的页面区域的父元素的选择器

data:{ ... }, //页面所需的所有变量

methods:{ ... }, //页面所需的所有函数

watch:{ //保存所有的监视函数

要监视的data中变量的名字(){

... ...

}

}

})

d. 示例: 边输入，就边自动搜索了

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">  <input type="text" v-model:value="keywords"><button @click="search">百度一下</button>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      keywords:""    },    methods:{      search(){        if(this.keywords.trim()!==""){          console.log(`搜索 ${this.keywords} 相关的内容...`)        }      }    },    watch:{      keywords(){//当同名变量keywords发生变化时，自动调用        this.search();      }    }  })  </script>  </body> |

三. 绑定样式：

1. 修改一个元素的某一个css属性:

(1). 不好的做法: 把整个style也看做时一个普通的字符串类型的属性:

<元素 :style="变量">

data:{

变量: "css属性1:值1; css属性2:值2"

问题: 不便于修改其中某一个css属性

(2). 好的做法: 把整个style看作一个对象，用对象语法绑定style的值

<元素 :style="变量">

data:{

变量: {

Css属性名1: "属性值1",

Css属性名2: "属性值2",

... : ...

}

}

运行时: 对象会被自动翻译为HTML中标准的style属性字符串:

style=" Css属性名1: 属性值1; Css属性名2: 属性值2;..."

(3). 示例: 使用vue操作飞机飞行

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <style>      img{        position:fixed;        margin-left:-100px;        margin-top:-50px;      }    </style>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">    <img src="img/p5.png" :style="imgStyle" alt="">  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      imgStyle:{        left:window.innerWidth/2+"px",        top:window.innerHeight-50+"px"      }    }  })  window.onkeydown=function(e){    if(e.keyCode==38){//如果按↑      vm.imgStyle.top=parseInt(vm.imgStyle.top)-10+"px";    }else if(e.keyCode==39){//按→      vm.imgStyle.left=parseInt(vm.imgStyle.left)+10+"px";    }else if(e.keyCode==40){//按↓      vm.imgStyle.top=parseInt(vm.imgStyle.top)+10+"px";    }else if(e.keyCode==37){//按←      vm.imgStyle.left=parseInt(vm.imgStyle.left)-10+"px";    }  }  </script>  </body> |

(4). 简写：（但不便于管理）

a. 可以用匿名对象来临时组织data中多个变量:

<元素 :style="{ css属性名1, css属性名2 }">

data:{

Css属性名1: "属性值1",

Css属性名2: "属性值2",

}

b. 风险: 如果要控制多个元素的css属性，很容易变量重名

c. 示例: 使用简写实现飞机的例子

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <style>      img{        position:fixed;        margin-left:-100px;        margin-top:-50px;      }    </style>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">    <img src="img/p5.png" :style="{left, top}" alt="">  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      left:window.innerWidth/2+"px",      top:window.innerHeight-50+"px"    }  })  window.onkeydown=function(e){    if(e.keyCode==38){//如果按↑      vm.top=parseInt(vm.top)-10+"px";    }else if(e.keyCode==39){//按→      vm.left=parseInt(vm.left)+10+"px";    }else if(e.keyCode==40){//按↓      vm.top=parseInt(vm.top)+10+"px";    }else if(e.keyCode==37){//按←      vm.left=parseInt(vm.left)-10+"px";    }  }  </script>  </body> |

2. 修改一个元素的class来批量修改元素的样式:

(1). 不好的方式: 也是将class属性看作一个普通的字符串来绑定

问题: 不便于修改其中某一个class  
 (2). 好的方式: 依然是用对象语法来绑定class

a. 如何:

<元素 :class="变量">

data:{

变量: {

Class名1: true,

Class名2: false,

Class名3: true

}

}

b. 结果: 只有值为true的class名，才会被自动翻译进元素最终的class属性字符串中

<元素 class="class1 class3">

(3). 问题: 一个元素上可能同时被多个class修饰，有的class是固定不变，之后少部分class是来回切换的。

解决: 其实不带:的普通的class属性和动态绑定的:class属性可以并存！

1). 固定不变的class，放在不带:的class中

2). 可能变化的class，放在带:的class绑定中

3). 结果: 两个class会合并为一个class字符串！

(4). 示例: 实现带样式的表单验证:

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <style>      .msg{        padding:5px 10px;      }      .success{        background-color:green;        color:lightgreen      }      .fail{        background-color:red;        color:#fff;      }    </style>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">    手机号: <input type="text" v-model="phone" @blur="vali">            <span class="msg" :class="spanClass">{{msg}}</span>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      phone:"",//通过双向绑定获得文本框中的手机号      spanClass:{//控制span的样式        success:false,        fail:false      },      msg:""//控制span的错误提示    },    methods:{      vali(){        var reg=/^1[3-9]\d{9}$/;        if(reg.test(this.phone.trim())){          this.spanClass={success:true, fail:false}          this.msg="手机号格式正确"        }else{          this.spanClass={success:false, fail:true}          this.msg="手机号格式不正确"        }      }    },    watch:{      phone(){//只要phone变化就自动执行验证操作        this.vali();      }    }  })  </script>  </body> |

(5). 也可以有简写: 用匿名对象语法来绑定class

<元素 :class="{ class名1, class名2, class名3 }">

data:{

Class名1: true,

Class名2: false,

Class名3: true

}

问题: 万一别的元素上刚好有同名的class，就容易冲突！

(6). 示例: 使用简写实现带样式的表单验证:

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <style>      .msg{        padding:5px 10px;      }      .success{        background-color:green;        color:lightgreen      }      .fail{        background-color:red;        color:#fff;      }    </style>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">    手机号: <input type="text" v-model="phone" @blur="vali">            <!--                      匿名对象        -->            <span class="msg" :class="{success, fail}">{{msg}}</span>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      phone:"",//通过双向绑定获得文本框中的手机号      success:false,      fail:false,      msg:""//控制span的错误提示    },    methods:{      vali(){        var reg=/^1[3-9]\d{9}$/;        if(reg.test(this.phone.trim())){          this.success=true;          this.fail=false;          this.msg="手机号格式正确"        }else{          this.success=false;          this.fail=true;          this.msg="手机号格式不正确"        }      }    },    watch:{      phone(){//只要phone变化就自动执行验证操作        this.vali();      }    }  })  </script>  </body> |

四: 自定义指令

1. 何时: 如果我们需要的功能，vue官方没有提供，则可以自定义指令:

2. 如何创建自定义指令:

//为Vue大家庭中添加一种新的自定义指令

Vue.directive("指令名",{

//插入后: 当带有这个指令的元素渲染到页面中之后自动触发的回调函数

inserted(domElem){

//domElem形参变量会自动接住当前这个带有指令的DOM元素对象

//对domElem执行vue中少有的DOM操作.

}

})

强调: 定义指令时一定不要加v-前缀！

3. 如何使用指令: 和使用官方的指令完全一样！

强调: 虽然定义指令时不加v-前缀，但是使用指令时，必须加v-前缀

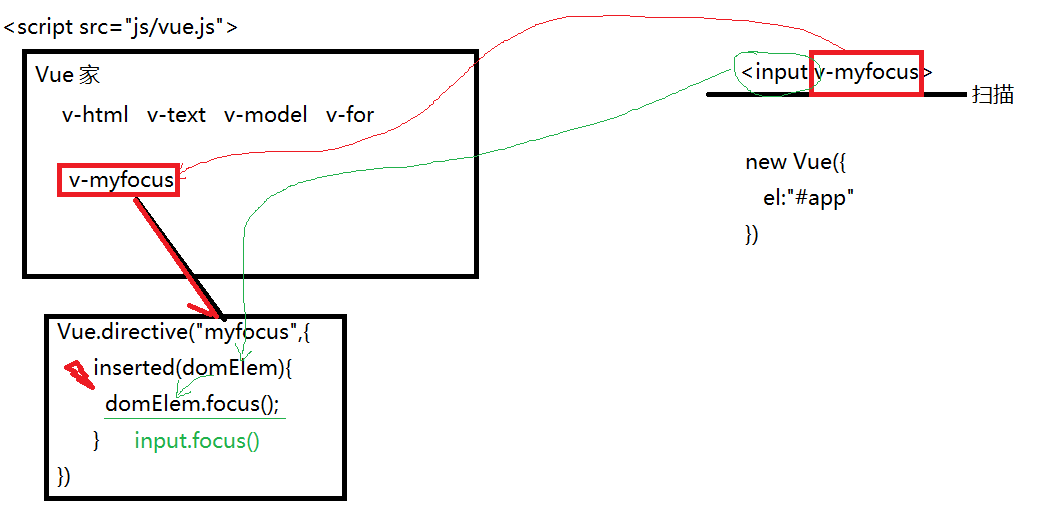
<元素 v-指令名>

4. 原理:

1). 当new Vue()渲染页面过程中，加载到这个元素时，发现了一个不认识的指令，于是就回Vue大家庭中查找是否有同名的指令。

2). 找到自定义指令后，会自动调用指令中的inserted函数，并自动将当前带有指令的DOM元素传入inserted函数中。

3). 在inserted函数中就可对当前这个DOM元素执行任何初始化的原生DOM操作！



5. 示例: 定义一个指令，让元素自动获得焦点

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="js/vue.js"></script>    <script>      Vue.directive("myfocus",{        //当当前元素插入到页面后        inserted(domElem){          //让当前元素自动获得焦点          domElem.focus();        }      })    </script>  </head>  <body>  <div id="app">    <input type="text" v-myfocus><button>百度一下</button>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      }  })  </script>  </body> |

五. 计算属性:

1. 什么是计算属性: 自己不保存属性值，而是依靠其他属性的值临时计算获得结果。

2. 为什么: 因为不是所有页面上需要的东西，后台都会给我们现成的！有些数据，后台没给！但是，页面上却需要。

3. 何时: 只要页面上需要的一个值，后台没有直接给，而是需要根据其他属性计算获得，就用计算属性

比如: 购物车的总价，不会在数据库里保存，因为变化很频繁！

但是购物车的总价却可以根据遍历购物车中每个商品累加出来！

4. 如何定义计算属性:

new Vue({

el:"#app",

data:{ ... },

methods:{ ... },

watch:{ ... },

computed:{

//计算属性本质是一个名称为属性名的函数:

属性名(){

//根据其他属性经过复杂的计算过程，获得结果

return 计算结果

}

}

})

5. 如何使用计算属性: 计算属性的用法和data中的普通变量完全一样！

强调: 在计算属性时，一定不要加()

6. computed vs methods 高频笔试题

(1). 用法: computed中的属性，使用时不要加()，而methods中的方法，调用时必须加()。

(2). 效率:

a. methods中的函数的执行结果，不会被缓存。只要重复调用一次，就会重新计算一次——效率低

b. computed中的计算属性的计算结果，会被vue缓存起来。只需要在首次加载时计算一次，然后即使反复使用，也不会反复计算——效率高！

除非计算属性所依赖的另一个变量值发生了变化，才会重新计算一次。但是计算后的结果，又会再次被vue缓存起来，反复使用。

(3). 如何选择:

a. 如果更侧重于获得一个计算结果的值时，首选computed

b. 如果更侧重于执行一项任务的步骤，而不关心结果时，首选methods

7. 示例: 使用计算属性计算购物车总价:

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="js/vue.js"></script>  </head>  <body>  <div id="app">    <h3>总价: ¥{{total.toFixed(2)}}</h3>    <ul>      <li v-for="(p,i) of cart" :key="i">        {{p.pid}} | {{p.pname}} | ¥{{p.price.toFixed(2)}} | {{p.count}} | 小计:¥{{(p.price\*p.count).toFixed(2)}}      </li>    </ul>    <h3>总价: ¥{{total.toFixed(2)}}</h3>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      cart:[        {pid:1, pname:"华为", price:5688, count:2},        {pid:2, pname:"小米", price:3688, count:3},        {pid:3, pname:"苹果", price:8688, count:1}      ]    },    methods:{      },    computed:{      total(){        console.log(`计算了一次总价`)        var sum=0;        for(var p of this.cart){          sum+=p.price\*p.count;        }        return sum;      }    }  })  </script>  </body> |

总结:

MVVM: 界面(View) 数据模型(Model) 视图模型(ViewModel)

Vue的绑定原理: 访问器属性+虚拟DOM树

new Vue({

el:"#app", //告诉new Vue()应该监控页面中哪个区域

data: { 页面所需的变量 },

methods: { 页面所需的函数 }

})

绑定与指令:

1. 如果元素内容可能发生变化: {{变量或js表达式}} 或 v-text="变量或js表达式"都行

2. 如果元素的属性值可能发生变化: :属性名="变量或js表达式"

3. 控制一个元素显示隐藏: v-show="条件" display:none

4. 控制两个元素二选一显示隐藏: v-if="条件" v-else 删除元素

5. 控制多个元素多选一显示隐藏: v-if="条件1" v-else-if="条件2" ... v-else

6. 反复生成多个相同结构的元素: v-for="(elem,i) of 数组/对象/数字" :key="i"

7. 绑定事件: @事件名="处理函数(实参值或$event)"

8. 防止短暂看到{{}}: v-cloak v-text

9. 绑定一段HTML内容: v-html

10. 只在首次渲染时绑定一次: v-once

11. 防止内容中的{{}}被编译: v-pre

12. 只要想获得表单元素的新值: v-model

作业:

1. (必须)看小程序视频学习vue中如何实现防抖和节流

小程序->在线->VUE->day02 3. 双向绑定 v-model 绑定文本框...

2. (必须)看小程序视频学习vue中三种表单元素常见效果的实现

小程序->在线->VUE->day02

4. 双向绑定 v-model radio sex ...

5. 双向绑定 v-model select 切换图片...

6. 双向绑定 v-model checkbox ...

3.(必须)看小程序中视频学习绑定样式的终极简化技巧

小程序->在线->VUE->day02 8 绑定样式 style 移动飞机 class 手机号验证 终极简写

4. (必须)看小程序视频学习vue实现标签页效果tab

小程序->在线->VUE->day01 作业: 使用vue实现标签页效果tabs

5. (必须)看小程序视频学习v-show时的过渡动画效果

小程序->在线->VUE->day01 作业: 为v-show添加过渡效果