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:

6. 过滤器:

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> |

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