

Vue学习第一天

课程安排

1. 总共有27天的课程，从前几期的17天增加到27天
2. 每天上课时长6小时
3. 分为三个小阶段，Vue基础、pc端项目（黑马头条后台管理系统）还有移动端（黑马头条移动端）
4. Vue基础7天课程，包括基本使用，指令，一些小的案例串起这些知识点

Vue.js介绍

1. [官方文档](#)
2. Vue.js是一个js框架，相比jquery，**基本上不用**操作DOM
3. 大家将来工作很可能是写Vue的

如何学习Vue.js

1. Vue.js的学习会参考Vue官方文档，以项目驱动学习。
2. 顺应Vue的语法

Vue.js的基本使用

[传送门](#)

1. Vue的读音
 1. 官方的读音 /vju:/
 2. 国内很多人读 /uju:/
2. 渐进式 是说Vue能够做大的项目也能做小的项目

*Vue基本使用解析

使用Vue步骤

1. 导入vue.js
2. dom结构:Vue管理的容器
3. 实例化new Vue({})
 1. el:'id选择器' 关联到Vue管理的容器
 2. data的值是对象，是可以通过{}在Vue管理的容器里面进行渲染。

```
1  <!-- {{message}} -->
2  <div id="container" class="app">
3    {{ message }}
4    <h2>{{ info }}</h2>
5  </div>
6
7  <!-- 开发环境版本，包含了有帮助的命令行警告 jquery.js 提供更多的错误信息提示-->
8  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
9  <!-- 生产环境版本，优化了尺寸和速度 jquery.min.js 更适合发上线-->
10 <!-- <script src="https://cdn.jsdelivr.net/npm/vue"></script> -->
11 <script>
12   /*
13     el是关联到Vue管理的容器
14     id选择器，class选择器，标签选择器。
15     id是唯一的，所以推荐用id
16   */
17   var app = new Vue({
18     // el: '#container',
19     // el:'div',
20     el: ".app",
21     data: {
22       message: "Hello world!",
23       info: "今天天气如何？"
24     }
25   });
26 </script>
```

js表达式

[传送门](#)

1. {{}}也叫插值语法，胡子语法,mustache语法。都一个概念
2. 用法 {{js表达式}}
3. js表达式是js语句，但是得返回一个值

```
1  <!-- dom结构 -->
2  <div id="container">
3    <h2>{{ message }}</h2>
4    <h2>{{ num * 23 }}</h2>
5    <h2>{{ isRed ? 'red' : 'green' }}</h2>
6    <!-- <h2>{{const PI=3.1415926535}}</h2> -->
7  </div>
8  <!-- 导入vue.js -->
9  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
10 <!-- 实例化Vue -->
11
12 <script>
13   const PI=3.1415926535
14   new Vue({
15     el: '#container',
16     data: {
17       message: '中分还带波浪，是不是很萌',
18       num: 99,
19       isRed: true
20     }
21   })
22 </script>
```

*Vue指令

[传送门](#)

指令 (Directives) 是带有 `v-` 前缀的特殊特性。

Vue指令是提供给HTML标签新增的属性

使用是 `v-text="值"`

Vue指令的作用: 作用在它所在的标签上, 具体的作用跟是什么指令有关

```
1 | <img src="" title="鼠标悬停文案" v-text="js表达式(值)">
```

v-text指令(textContent)

[传送门](#)

1. v-text的作用: 把值作为文本插入到标签之间
2. 底层的实现textContent
3. 会覆盖掉标签之间的文本
4. 推荐用简写{{}}
5. 不能解析html

```
1 | <div id="app">
2 |   <h3>{{msg}}</h3>
3 |   <h3 v-text="msg"></h3>
4 |   <h3 v-text="msg">标签之间的文本</h3>
5 |   <h2>前面的文本--{{msg}}--之后的文本</h2>
6 |   {{alink}}
7 |
8 | </div>
9 | <script src="./lib/vue.js"></script>
10 | <script>
11 |   /*
12 |     v-text作用就是把值作为文本插入到所在的标签之间。
13 |     底层实现是textContent
14 |     会覆盖掉标签之间的文本
15 |     {{}}是简写,推荐用简写
16 |     不能解释html
17 |   */
18 |   const app = new Vue({
19 |     el: "#app",
20 |     data: {
21 |       msg: '还是吃得太饱了',
22 |       alink: '<a href="http://www.baidu.com">百度</a>'
23 |     }
24 |   });
25 | </script>
```

v-html指令(innerHTML)

[传送门](#)

1. v-html指令是把值作为用html插入到所在的标签之间。
2. 底层innerHTML实现的。
3. 会覆盖标签之间的文本
4. 没有简写
5. 使用得比较少

```
1 <div id="app">
2   <h4>{{alink}}</h4>
3   <h4 v-html="alink"></h4>
4   <h4 v-html="alink">标签之间的文本</h4>
5 </div>
6 <script src="./lib/vue.js"></script>
7 <script>
8   /*
9     v-html指令是把值作为用html插入到所在的标签之间。
10    底层innerHTML实现的。
11    会覆盖标签之间的文本
12    没有简写
13    使用得没那么多
14    */
15    const app = new Vue({
16      el: "#app",
17      data: {
18        alink: '<a href="http://www.baidu.com">百度</a>'
19      }
20    });
21 </script>
```

*v-on指令

基本使用

注册事件

1. 使用方法 v-on:事件名="事件处理方法"
2. 简写@，推荐用简写
3. 事件名和原生html标签里注册事件的事件名是一样的。可以是
click,dblclick,mouseover,mouseenter,keyup,keydown,keypress,blur,focus
4. methods是el、data是平级的
5. 事件处理方法应该声明在methods里面
6. methods里面的方法，推荐用简洁写法

```
1 <div id="app">
2   <input type="button" value="点我呀" v-on:click="sayHello" />
3   <input type="button" value="简写" @click="sayHello" />
4   <input type="button" value="双击" @dblclick="sayHello" />
5 </div>
6 <script src="./lib/vue.js"></script>
7 <script>
8   /*
9     注册事件的方法 v-on:事件名="事件处理方法"
10    事件处理方法声明在methods
11    事件名和原生html里面注册事件的事件名是相同
12    dblclick,click,mouseover,input,keyup,keydown...
13    简写@(at) 推荐用简写
14    */
15    const app = new Vue({
16      el: '#app',
17      data: {},
18      // 一些方法
19      methods: {
20        // sayHello:function(){
21        //   alert('function')
22        // },
23        sayHello: () => {
24          alert('=>')
25        }
26        // 简洁写法
27        // sayHello() {
28        //   alert('Hello')
29        // }
30      }
31    })
```

```
30 |   })
31 | </script>
```

事件处理方法的参数

事件处理方法里面，会默认接受一个事件对象event，比较少使用

事件传参和html注册事件一样的。

如果方法无须参数的话，就不要给括号

```
1 | <div id="app">
2 |   <button @click="sayHello">点我呀</button>
3 |   <button @click="call()">括号</button>
4 |   <button @click="bigger(77)">逼格</button>
5 | </div>
6 | <script src="../lib/vue.js"></script>
7 | <script>
8 |   /*
9 |     方法默认接受一个事件对象，比较少使用
10 |    */
11 |   const app = new Vue({
12 |     el: "#app",
13 |     data: {},
14 |     methods: {
15 |       sayHello(event){
16 |         // alert('hello')
17 |         console.log(event)
18 |       },
19 |       call(event){
20 |         console.log(event) //undefined
21 |       },
22 |       bigger(num){
23 |         alert(num*55)
24 |       }
25 |     },
26 |   });
27 | </script>
```

*事件修饰符

[传送门](#)

1. 使用方法 v-on:事件名.修饰符="事件处理方法"
2. 常用的三个事件修饰符
 1. @keyup.enter enter键抬起的时候触发的事件
 2. .stop阻止事件冒泡
 3. .prevent阻止默认事件

```
1 <!-- 阻止单击事件继续传播 -->
2 <a v-on:click.stop="doThis"></a>
3
4 <!-- 提交事件不再重载页面 -->
5 <form v-on:submit.prevent="onSubmit"></form>
6
7 <!-- 修饰符可以串联 -->
8 <a v-on:click.stop.prevent="doThat"></a>
9
10 <!-- 只有修饰符 -->
11 <form v-on:submit.prevent></form>
12
13 <!-- 只当在 event.target 是当前元素自身时触发处理函数 -->
14 <!-- 即事件不是从内部元素触发的 -->
15 <div v-on:click.self="doThat">...</div>
16
17 <!-- 只有在 `key` 是 `Enter` 时调用 `vm.submit()` -->
18 <input v-on:keyup.enter="submit">
```

```
1 <div id="app">
2   <a href="http://www.baidu.com" @click.prevent="clickHandler">百度
3 </a>
4 </div>
5 <script src="../lib/vue.js"></script>
6 <script>
7   /*
8     使用方法：v-on:事件名.修饰符 = "事件处理方法"
9     常用的三个事件修饰符
10    1. @keyup.enter enter键抬起时触发
```



```

10      2. .stop阻止事件冒泡
11      3. .prevent阻止默认事件
12      */
13      const app = new Vue({
14        el: '#app',
15        data: {},
16        methods: {
17          clickHandler(event) {
18            //阻止默认事件
19            // event.preventDefault();
20            alert('Cool')
21          }
22        }
23      })
24    </script>

```

vue方法中的this

[传送门](#)

方法中的 `this` 自动绑定为 Vue 实例。

1. 方法中的this就是Vue实例
2. 方法中的this可以直接访问到data和methods的属性，`this.`
3. Vue的属性值改变，对应的视图会响应。
4. 方法声明建议用方法的简洁写法

```

1  <div id="app">
2    <h3>{{ message }}</h3>
3    <button @click="sayHello">点我呀</button>
4  </div>
5  <script src="./lib/vue.js"></script>
6  <script>
7    /*
8      方法中的this就是Vue实例
9      Vue把data和methods里面属性直接设置到Vue实例上 app.
10     改变data值，对应的视图会响应
11     方法声明建议用方法的简洁写法
12     */
13    const app = new Vue({

```

```

14   el: '#app',
15   data: {
16     message: '这是一个寂寞的天',
17     person: {
18       name: 'Joven',
19       age: 30
20     }
21   },
22   methods: {
23     sayHello() {
24       // alert('hello')
25       // console.log(this === app) //true
26       this.message = '是兄弟，就来玩贪玩蓝月'
27       console.log(this.message)
28     },
29     // sayHello: () => {
30     //   console.log(this) //window
31     // }
32   }
33 })
34 console.log(app)
35 </script>

```

js表达式的作用域

[传送门](#)

表达式会在所属 Vue 实例的数据作用域下作为 JavaScript 被解析

js表达式里面的变量是Vue实例的属性

1. data和methods的属性在表达式里可以直接使用，不要加this

```

1 <div id="app">
2   <h3>{{ message }}</h3>
3   <button @click="sayHello">点我呀</button>
4   <button @click="message = '是兄弟，就来玩贪玩蓝月'">行内</button>
5   <button @click="print">打印</button>
6 </div>
7 <script src="./lib/vue.js"></script>
8 <script>

```

```

9   const app = new Vue({
10     el: '#app',
11     data: {
12       message: '这是一个寂寞的天'
13     },
14     methods: {
15       sayHello() {
16         this.message = '是兄弟，就来玩贪玩蓝月'
17       },
18       print(){
19         console.log(5555)
20       }
21     }
22   })
23 </script>

```

v-bind指令

[传送门](#)

动态地绑定一个或多个html标签属性

如果html标签的属性不是写死的，都应该用v-bind

1. 使用方法: v-bind:属性名="js表达式"
2. v-bind:src="imgUrl"，src属性绑定到imgUrl，改变imgUrl，会动态改变src的值
3. 简写是 :属性名="js表达式" 当然推荐用简写
4. :src="imgUrl2" 当我们改变imgUrl2的时候，就会改变src的值，图片改变了。
5. v-bind:class = "添加的类名" 可以用三元运算，值为字符串
6. v-bind:disabled="是否禁用" 值为true的时候，禁用;值为false的时候，会移除disabled

```

1  <div id="app">
2    <!-- {{imgUrl}} -->
3    
4    
5  </div>
6  <script src="./lib/vue.js"></script>
7  <script>
8    const app = new Vue({
9      el: "#app",

```

```
10   data: {
11     imgUrl: './img/cat.gif',
12     imgUrl2: './img/girl.png'
13   },
14   methods: {
15     changeImg() {
16       this.imgUrl2 = './img/boy.png'
17     }
18   },
19   });
20 </script>
```

```
1 <div id="app">
2   <div :class="bgRed"></div>
3   <div :class="isRed?'red':''" @click="isRed=!isRed"></div>
4 </div>
5 <script src="./lib/vue.js"></script>
6 <script>
7   const app = new Vue({
8     el: '#app',
9     data: {
10       bgRed: 'red',
11       isRed: true
12     }
13   })
14 </script>
```

```
1 <div id="app">
2   <button @click="isDisabled=!isDisabled">禁用</button>
3   <button :disabled="isDisabled">点我呀</button>
4 </div>
5 <script src="./lib/vue.js"></script>
6 <script>
7   const app = new Vue({
8     el: "#app",
9     data: {
10       isDisabled: false
11     }
12   });
13 </script>
```

Demo-计数器



实现步骤

1. 显示数字
 1. 声明数字 `data.num:0`
 2. 显示数字 `{{data}}`
2. 点击+按钮，数字+1;点击-按钮，数字-1
 1. 点击+ `@click:add num++`
 2. 点击- `@click:sub num--`
3. 数字范围是0~10，边界问题的处理
 1. 值为0的时候给-加disabled属性和disabled的样式
 1. 值为0时添加disabled属性 `v-bind:disabled="num===0"`
 2. 值为0时候，添加disabled样式 `v-bind:class="num===0?'disabled':""`
 2. 值为10的时候给+加disabled属性和disabled样式
 1. 值为10时添加disabled属性 `v-bind:disabled="num===10"`
 2. 值为10时候，添加disabled样式 `v-bind:class="num===10?'disabled':""`

注意点

1. `v-bind:disabled="是否禁用"`
2. `v-bind:class="添加类名"`
3. html标签的disabled属性，只要添加了，就会禁用，无论disabled的值为多少。再者有disabled属性的button，并不会触发点击事件。

Demo-图片切换

图片切换



实现步骤

1. 显示图片
 1. 图片数组装四张图片 `imgList:[img1,img2,img3,img4]`
 2. 数组下标`index:0`
 3. 取图片 `v-bind:src=imgList[index]`
2. 点击上一张按钮，显示上一张图片;点下一张按钮，显示下一张图片
 1. 点击上一张 `@click:pre index--`
 2. 点击下一张 `@click:next index++`
3. 图片循环展示
 1. 如果当前是最后一张，点下一张，去到第一张
 2. 如果当前是第一张，点上一张，去到最后一张

注意点

1. 图片的展示 `v-bind:src="imgList[index]"`就取个值
2. 数组的边界问题
 1. 上一张：最后一张->第一张
 2. 上一张： 第一张->最后一张

*表单输入绑定 v-model

[传送门](#)

你可以用 `v-model` 指令在表单 `<input>`、`<textarea>` 及 `<select>` 元素上创建双向数据绑定。

1. v-model只能用在input、textarea、select三个元素上
2. 获取用户的输入
3. 双向数据绑定
 1. 获取用户的输入，value=>data属性
 2. 设置表单元素的值，data属性=>value

```
1 <div id="app">
2   <h2>{{message}}</h2>
3   <input type="text" v-model="message">
4 </div>
5 <script src="./lib/vue.js"></script>
6 <script>
7   const app = new Vue({
8     el: "#app",
9     data: {
10       message: '坏得很。。。'
11     }
12   });
13 </script>
```

*v-for指令

[传送门](#)

我们可以用 `v-for` 指令基于一个数组来渲染一个列表。

v-for遍历数组，渲染列表元素

1. 用法 `v-for="元素的别名 in 数组"`
2. 用法 `v-for="(元素的别名, 下标) in 数组"`
3. v-for需要写在需要重复的元素，比如li
4. 元素别名和下标也可以在v-for作用的标签上和标签之内使用。

5. in 是一个关键字不能改。

```
1 <div id="app">
2   <ul>
3     <li v-for="(movie,index) in movieList">{{movie}}--{{index}}</li>
4   </ul>
5 </div>
6 <script src="./lib/vue.js"></script>
7 <script>
8   const app = new Vue({
9     el: "#app",
10    data: {
11      movieList: [
12        '上海堡垒',
13        '哪吒',
14        '烈火英雄',
15        '沉默的证人'
16      ]
17    }
18  });
19 </script>
```

Demo-记事本

实现步骤

1. 显示列表

1. 列表的数据 todoList:['睡觉觉','吃饭饭','打豆豆']
2. v-for遍历todoList ,vfor加在li标签上

2. 输入任务，回车，添加任务

1. 获取到输入框的内容 v-model:inputVal
2. 回车 @keyup.enter:addTodo
3. 添加任务就是给数组添加一项 todoList.push()
4. 清空输入

3. 双击删除任务

1. 双击 @dblclick:delTodo(index)
2. index来自v-for里面
3. arr.splice(从哪个下标开始删除, 删除多少项)

注意点

1. v-model的修饰符
 - `.lazy` - 取代 `input` 监听 `change` 事件
 - `.number` - 输入字符串转为有效的数字
 - `.trim` - 输入首尾空格过滤
2. arr.splice(从哪一项开始删除, 删除多少项)

总结

插件推荐

Vetur 让vscode提高对vue的支持, 比如高亮, 比如图标...

Vue 2 Snippets vue关键语法的提示

prettier 格式化代码

path Intellisense 路径提示

用户代码片段
