Vue学习第一天

课程安排

1. 总共的课程27天
2. 三个阶段 Vue基础 、PC端项目和移动端项目
3. 基础课程7天

Vue.js介绍

1. 官方文档
2. Vue是一个js框架，对比jquery, **不需要操作DOM，精力集中在数据上。**
3. 大家就业后更有可能就是写Vue.js

Vue.js的HelloWorld

传送门

1. Vue.js的学习会参考Vue官方文档，以项目驱动学习。
2. vue读音,两种都要了解
    1. /vju:/
    2. /uju:/
3. 渐进式: vue能做小的项目也能做大的项目

Vue基本使用解析

使用vue.js

1. 引入vue.js
2. dom结构，这个Vue管理的容器
3. 实例化Vue
    1. el:'id选择器' 关联Vue管理的容器
    2. data是一个对象，里面的属性 ，在Vue管理的容器中用{{}}渲染

```
<!-- {{message}} -->
<div id="container" class="app">
  {{ message }}
  <h2>{{info}}</h2>
```

```
</div>
<!-- 开发环境版本，包含了有帮助的命令行警告 jquery.js-->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<!-- 生产环境版本，优化了尺寸和速度 jquery.min.js-->
<!-- <script src="https://cdn.jsdelivr.net/npm/vue"></script> -->
<script>
  var app = new Vue({
    // el: '#container',
    // el:'.app',
    el:'div',
    data: {
      // message: 'Hello Vue!'
      message:'Hello World!',
      info:'今天天气如何？'
    }
  })
</script>
```

js表达式

传送门

js表达式：结果是一个值的js语句

```
    <!-- dom结构 -->
    <div id="container">
      <h2>{{ message }}</h2>
      <h2>{{ num + 100 }}</h2>
      <h2>{{ isRed ? 'red' : '' }}</h2>
      <!-- <h2>{{const PI = 3.141592653}}</h2> -->
      <h2>{{ 1 + 1 }}</h2>
    </div>
    <!-- 引入vue.js -->
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

    <!-- 实例化Vue.js -->
    <script>
      new Vue({
        el: '#container',
        data: {
          message: '还是吃得太饱了',
          num: 100,
          isRed: true
        }
      })
    </script>
```

Vue指令

传送门

指令 (Directives) 是带有 `v-` 前缀的特殊特性。

vue提供给HTML标签新增的属性。

用法和HTML属性一样。 `v-text="js表达式"`

```
<img src="./img/boy.png" title="鼠标悬停文案" v-text="js表达式">
```

v-text指令(textContent)

传送门

1. v-text把值作为**文本插入**到所在的标签之间
2. Vue内部是textContent实现的
3. 会覆盖掉标签之间的文本
4. 简写{{}}，简写用得更多
5. 不能解析html

```
<div id="app">
  <h2>{{ message }}</h2>
  <h2 v-text="message"></h2>
  <h2 v-text="message">标签之间的文本</h2>
  <h2>之前的文本--{{ message }}--之后的文本</h2>
  {{ alink }}
</div>
<script src="./lib/vue.js"></script>
<script>
  /*
      v-text:把值作为文本插入到它所在的标签之间 textContext，会覆盖掉标签之间的文案
      简写就是{{}}，推荐使用{{}}
      v-text不能解析html
    */
  const app = new Vue({
    el: '#app',
    data: {
      message: '今天天气如何',
      alink: '<a href="http://www.baidu.com">百度</a>'
    }
  })
</script>
```

v-html指令(innerHTML)

传送门

1. v-html是把值作为html插入到所在的标签之间，内部实现是innerHTML。会覆盖标签之间的html
2. 没有简写

```
<div id="app">
  <h2>{{ alink }}</h2>
  <h2 v-html="alink"></h2>
  <h2 v-html="alink">标签之间的文本</h2>
</div>
<script src="./lib/vue.js"></script>
<script>
  /*
        v-html把值作为html插入到它所在的标签之间。底层实现 innerHTML
        没有简写
      */
  new Vue({
    el: '#app',
    data: {
      alink: '<a href="http://www.baidu.com">百度</a>'
    }
  })
</script>
```

v-on指令

基本使用

1. v-on:事件名="事件处理方法"
2. 事件处理方法声明在methods对象里边，methods和el、data并列
3. 简写@事件名="事件处理方法" 当然推荐用简写
4. 事件名和原生的事件名一致，可以是dblclick,click,mouseover,keyup,keyenter....

```
<div id="app">
  <input type="button" value="点我呀" v-on:click="sayHello">
  <input type="button" value="简写" @click="sayHello">
  <input type="button" value="双击" @dblclick="sayHello">
</div>
<script src="./lib/vue.js"></script>
<script>
  /*
        v-on:事件名="事件处理方法"
        事件处理方法声明在methods对象里面,methods和el、data并列
        简写 @事件名="事件处理方法" 当然推荐用简写
        事件名和原生相同,事件名可以是dblclick,click,mouseover,focus,blur,keyup,keyenter...
      */
  const app = new Vue({
```

```
    el: "#app",
    data: {

    },
    methods: {
      sayHello(){
        alert('hello')
      }
    },
  });
</script>
```

事件参数

1. 默认事件参数是event对象
2. 事件处理方法的调用如果给括号的话，无法接受Vue传过来的event对象
3. 方法传参和原来HTML注册事件方法没什么分别
4. methods是一个对象，里面方法声明有三种，推荐用ES6简洁写法

```
<div id="app">
  <button @click="sayHello">点我呀</button>
  <button @click="callMethod()">有刮号</button>
  <button @click="bigger(99)">逼格</button>
</div>
<script src="./lib/vue.js"></script>
<script>
  /*
        事件处理方法，默认接受一个事件对象，少部分情况下会用到的
        传参和原来HTML注册事件没什么不同
      */
  const app = new Vue({
    el: '#app',
    data: {},
    methods: {
      // 方法的简洁写法
      sayHello(event) {
        // alert('ok')
        console.log(event)
      },
      // sayHello:function(){
      //   alert('ok')
      // },
      sayHello: () => {
        alert('ok')
      },
      callMethod(event) {
        console.log(event) //undefined
      },
```

```
      bigger(num) {
        alert(99 * 77)
      }
    }
  })
</script>
```

事件修饰符

传送门

使用方法： v-on:事件名.修饰符 = "事件处理方法"

```
<!-- 阻止单击事件继续传播 -->
<a v-on:click.stop="doThis"></a>

<!-- 提交事件不再重载页面 -->
<form v-on:submit.prevent="onSubmit"></form>

<!-- 修饰符可以串联 -->
<a v-on:click.stop.prevent="doThat"></a>

<!-- 只当在 event.target 是当前元素自身时触发处理函数 -->
<!-- 即事件不是从内部元素触发的 -->
<div v-on:click.self="doThat">...</div>

<!-- 只有在 `key` 是 `Enter` 时调用 `vm.submit()` -->
<input v-on:keyup.enter="submit">
```

```
<div id="app">
  <input type="text" value="这是一个寂寞的天" @keyup="keyupHanlder" />
  <input type="text" value="下着有些伤心的雨" @keyup.enter="enterHandler" />
</div>
<script src="./lib/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {},
    methods: {
      // enter键抬起的时候，打印input value
      keyupHanlder(event) {
        //判断是否是enter键
        if (event.keyCode === 13) {
          console.log(event.target.value)
        }
      },
      enterHandler(event){
        console.log(event.target.value)
      }
    }
```

```
  })
</script>
```

vue方法中的this

传送门

方法中的 `this` 自动绑定为 Vue 实例。

1. 方法里面的this就是Vue实例
2. 方法里面的this可以访问到data的属性和methods的方法. 用法this.
3. Vue里面属性值改变，对应视图将会"响应"

```
<div id="app">
  <h2>{{message}}</h2>
  <input type="button" value="修改data" @click="changeData">
</div>
<script src="./lib/vue.js"></script>
<script>
  const app = new Vue({
    el: "#app",
    data: {
      message:'中分还带波浪',
      person:{
        name:'Joven',
        age:18
      }
    },
    methods: {
      // changeData(){
      //   // console.log(this===app)
      //   console.log(this.message)
      //   this.message = '李晨又分手了'
      // },
      // 箭头函数的地方式，方法里面的this是window
      changeData:()=>{
        console.log(this)
      },
      run(){
        console.log('biu biu biu ...')
      }
    },
  });
  console.log(app)
</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里面的属性在表达式里面可以直接使用，不要加this

```
<div id="app">
  <h2>{{msg}}</h2>
  <input type="button" value="点我呀" @click="sayHello">
  <input type="button" value="行内" @click="msg='中分还带波浪。。。'">
  <input type="button" value="打印" @click="print">
</div>

<script src="./lib/vue.js"></script>
<script>
  const app = new Vue({
    el: "#app",
    data: {
      msg:'这是一个寂寞的天。。'
    },
    methods: {
      sayHello(){
        this.msg='中分还带波浪。。。'
      },
      print(){
        console.log('结婚证')
      }
    },
  });
</script>
```

v-bind指令

传送门

动态地绑定一个或多个特性

v-bind指令动态绑定HTML标签的属性。当属性值不是一个写死的值时，都需要用动态绑定v-bind

1. v-bind指令的使用 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. v-bind：可以简写为 :属性名 = "js表达式" 当然推荐简写
4. v -bind:class="需要添加的class"
5. v-bind:diabled="是否添加disabled属性"

```
<div id="app">
```

```
  <h2>{{ imgUrl }}</h2>
  <img v-bind:src="imgUrl" alt="" @click="changeImg" />
  <img :src="imgUrl2" />
</div>
<script src="./lib/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      imgUrl: './img/01.png',
      imgUrl2: './img/cat.gif'
    },
    methods: {
      changeImg() {
        this.imgUrl = './img/03.png'
      }
    }
  })
</script>
```

```
<div id="app">
  <div :class="bgRed"></div>
  <div :class="isRed?'red':''" @click="isRed=!isRed"></div>
</div>
<script src="./lib/vue.js"></script>
<script>
  /*
      v-bind：简写 :属性名="js表达式"
      */
  new Vue({
    el: "#app",
    data: {
      bgRed:'red',
      isRed:true
    }
  });
</script>
```

```
<div id="app">
  <button @click="isDisabled=!isDisabled">缴械</button>
  <input type="text" :disabled="isDisabled" />
</div>
<script src="./lib/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      isDisabled: true
    }
  })
</script>
```

Demo-计数器

实现步骤

1. 数字的显示 1.声明数字 data.num 2.显示数字 {{num}
2. +-按钮功能实现 1.+注册点击事件 @click:add num++ 2.-注册点击事件 @click:sub num--
3. 数字范围是0-10，边界问题的处理
    1. 数字为0的时候
        1. -按钮添加disabled属性
            1. v-bind:disabled="num===0"
        2. -按钮添加'disabled'样式
            1. v-bind:class="num==0?'diabled':''"
    2. 数字为10的时候
        1. +按钮添加disabled属性
            1. v-bind:disabled="num===10"
        2. +按钮添加'disabled'样式
            1. v-bind:class="num==10?'diabled':''"

注意点

1. v-bind:disabled="是否添加disabled属性" 值为true的时候，按钮disabled; 否则不disabled
2. v-bind:class="添加的样式" 三元运算的值为className

Demo-图片切换

实现步骤

1. 图片的展示
    1. 图片数组装四张图片imgList:[img1,img2,img3,img4] 下标0,1,2,3
    2. 数组下标
    3. 取出数组中一张图片 imgList[index]
    4. 显示图片v-bind:src="imgList[index]"
2. 上一张和下一张功能
    1. 上一张 @click:pre index--
    2. 下一张 @click:next index++
3. 图片循环展示
    1. 点下一张的时候，如果当前显示最后一张，下一张是第一张
    2. 点上一张的时候，如果当是显示是第一张，上一张是最后一张

注意点

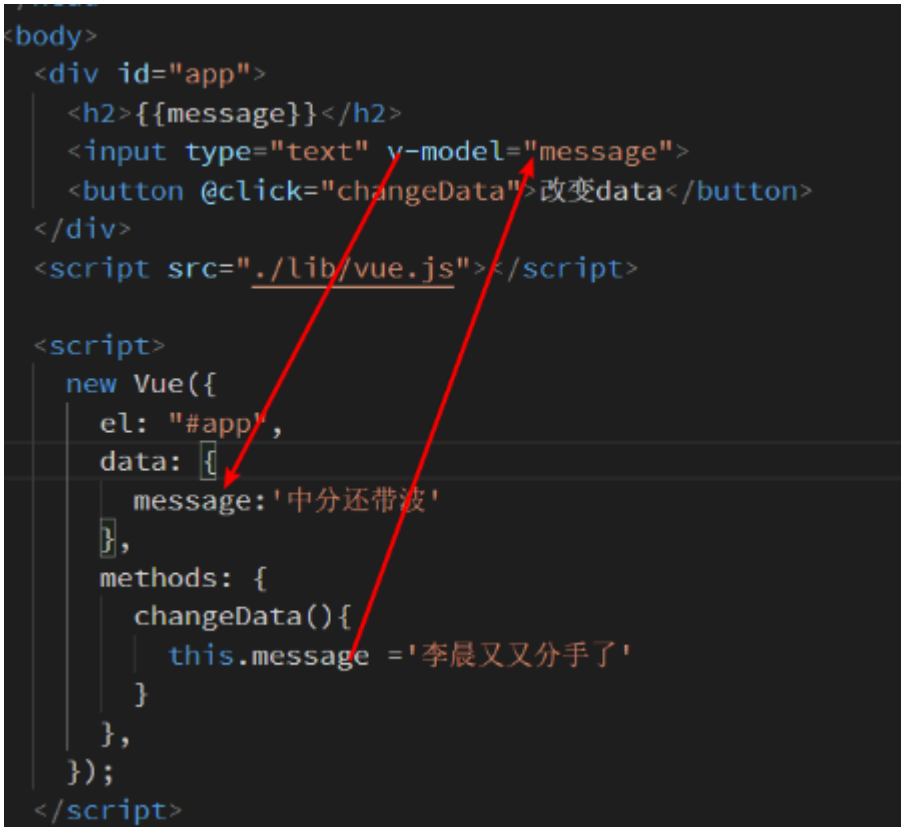
1. 图片的展示 v-bind:src="imgList[index]"
2. 数组的边界的问题
    1. 下一页： 数组元素最后一项->第一项
    2. 上一页： 第一项->数组最后一项

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只能用在表单标签上, input, textarea, select
2. 双向数据绑定：
    1. 表单用户输入改变引起data.message改变
    2. data.message改变引起表单用户输入值的改变
3. v-model是用来获取用户输入的




```
<div id="app">
      <h2>{{message}}</h2>
      <input type="text" v-model="message">
      <button @click="changeData">改变data</button>
    </div>
    <script src="./lib/vue.js"></script>

    <script>
      new Vue({
        el: "#app",
        data: {
          message:'中分还带波'
        },
        methods: {
          changeData(){
            this.message ='李晨又又分手了'
          }
        },
      });
    </script>
```

v-for指令

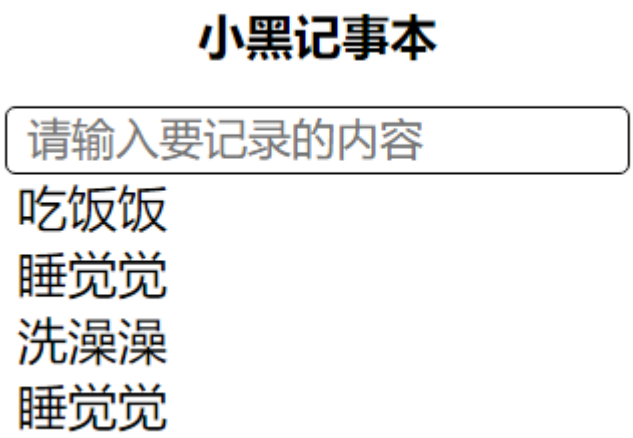
传送门

用 `v-for` 指令基于一个数组来渲染一个列表

1. v-for用来遍历数组并生成多个标签
2. v-for作用在需要重复的元素上
3. 语法 v-for="元素别名 in 数组" 或者 v-for="(元素别名,下标别名) in 数组"
4. 在v-for作用的标签之间就可以用使用元素别名和下标别名
5. in是固定的，不能变

```
<div id="app">
  <ul>
    <li v-for="(movie,index) in movieList">
      {{movie}} --{{index}}
 </li>
  </ul>
</div>
<script src="./lib/vue.js"></script>
<script>
  new Vue({
    el: "#app",
    data: {
      // 电影列表
      movieList:[
        '战狼1',
        '战狼2',
        '战狼3'
      ]
    }
  });
</script>
```

Demo-记事本



实现步骤

1. 展示任务列表
    1. todoList:['睡觉觉','吃饭饭','打豆豆']
    2. v-for="元素别名 in 数组" li就可以使用元素别名
2. 用户输入内容，回车，添加任务
    1. 获取用户输入 v-model:inputVal
    2. enter键抬起时候触发事件 @keyup.enter="addTodo"
    3. 添加任务 todoList.push(inputVal)
    4. 清空输入
3. 双击删除任务
    1. 双击事件 @dblclick:delTodo(index)
    2. index来自v-for
    3. todoList.splice(从哪个下标开始删除，删除多少项)

注意点

1. v-for="（元素别名,下标别名） in 数组"
2. v-model修饰符

    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
3. array.splice(从哪个下标开始删除，删除元素的个数)

v-if,v-else-if,v-else指令

传送门

`v-if` 指令用于条件性地渲染一块内容。这块内容只会在指令的表达式返回 truthy 值的时候被渲染。

1. v-if="js表达式"和v-else-if="js表达式"，如果表达式值为true，才会渲染该元素

2. v-else 以上的条件不满足时，添加所在有元素

```
<div id="app">
  <h2>老弟，你今年{{ age }}岁</h2>
  <input type="text" v-model="age" />
  <h2 v-if="age<18">偷偷摸摸去网吧！</h2>
  <h2 v-else-if="age<25">光明正大去网吧</h2>
  <h2 v-else>你到了晚婚的年龄了</h2>
</div>
<script src="./lib/vue.js"></script>
<script>
  const app = new Vue({
    el: "#app",
    data: {
      age:18
    }
  });
</script>
```

v-show指令

传送门

`v-show` 只是简单地切换元素的 CSS 属性 `display`。

1. v-show隐藏元素通过改变元素css属性display:none
2. v-if 隐藏元素直接移除dom
3. 对于频繁切换隐藏和显示的元素，就用v-show

```
<div id="app">
  <button @click="isShow=!isShow">切换显示</button>
  <h2>使用v-if</h2>
  <h2 v-if="isShow">中分还带波浪，是不是很萌呢</h2>
  <h2>使用v-show</h2>
  <h2 v-show="isShow">中分还带波浪，是不是很萌呢</h2>
</div>
<script src="./lib/vue.js"></script>
<script>
  new Vue({
    el:'#app',
    data:{
      isShow:true
    }
  })
</script>
```

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

用户代码片段