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:

1. 反复生成多个相同结构的HTML元素: v-for

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

强调: v-for一定要放在那个要反复生成的元素上，而不是放在父元素上！

(2). 原理:

a. 当new Vue()扫描到这里时，自动遍历of后的数组中每个元素

b. 每遍历一个元素，就创建一个当前HTML 元素的副本

c. of前的两个变量:

1). value会自动获得当前正在遍历的数组元素值

2). i 会自动获得当前正在遍历的下标位置

d. 如果当前元素或子元素中，需要使用当前正在遍历的元素值或下标，可用绑定语法来绑定value和i的值。

强调: value和i的使用范围仅限于当前元素及其子元素范围内，不能在当前元素外使用！

(3). 示例: 遍历数组元素，反复生成多个相同结构的元素

|  |
| --- |
| <div id="app">  <ul>  <!--本例中: 因为要反复生成多个li，所以v-for要写在li上，而不是li的父元素ul上-->  <li v-for="(value,i) of teachers" :key ="i">  第{{i+1}}阶段: {{value}}  </li>  </ul>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  teachers:["亮亮","然然","东东","涛涛"]  }  })  </script> |

(4)v-for还可:

a. 可遍历数字下标的一切: 比如字符串

b. 可遍历对象中每个属性

示例: 遍历对象中每个属性，反复生成多个html元素

|  |
| --- |
| <div id="app">  <!--希望遍历data中一个对象的每个属性，反复生成多个相同结构的HTML元素-->  <ul>  <li v-for="(value,key) of ym" :key ="key">{{key}} : {{value}}</li>  </ul>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  ym:{  math:89,  chs:69,  eng:91  }  }  })  </script> |

c. v-for还会数数: 给v-for一个数字，他可以生成从1开始依次递增的一个序列！

1).<要反复生成的元素 v-for="i of 数字">

2). 原理: v-for会像人一样，自动从1开始数数，每数一个数，就自动创建当前元素的一个副本。直到数到给定数字结束。  
 3). 其中: i会接住每次数的一个数字，可用于当前元素及子元素的绑定中。

示例: 根据页数，生成指定个数的分页按钮

|  |
| --- |
| <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>  ul{ list-style:none }  ul>li{  float:left;  border:1px solid #555;  width:36px;  height:36px;  line-height:36px;  text-align:center;  }  ul>li+li{  border-left:0  }  </style>  <script src="js/vue.js"></script>  </head>  <body>  <div id="app">  <ul>  <li v-for="i of pageCount">{{i}}</li>  </ul>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  pageCount:3  }  })  </script>  </body> |

(5). 坑: 如果v-for遍历的是数组时，在程序中通过下标修改数组元素值，页面上的HTML元素不会自动更改！

比如: this.teachers[0]="燕儿" 页面上是不会变的！

因为数组中的数字类型的下标012...无法添加访问器属性，也就不受监控！

解决: 今后，vue中修改数组中的元素值！必须用数组家函数！才能自动更新页面。因为函数都是受监控的。

比如: this.teachers.splice(0,1,"燕儿")

删除0位置的1个元素，再在0位置放入"燕儿"

结果: 页面会自动变化！

(6). 其实每次使用v-for时，都要同时绑定:key="i"

a. 鄙视: 为什么v-for必须加:key

b. 答: 因为v-for反复生成的多个元素，除了内容不同之外，从元素属性上来看多个元素毫无差别！每个反复生成的元素都是一样的。所以，如果将来修改了数组中一个元素时，v-for因为无法识别每个HTML元素，所以只能把所有的HTML元素重新生成一遍——效率低！

如果给每个元素都绑定:key="i"属性，则每个HTML元素上都有一个唯一的标识key="0" key="1" ... 。当将来修改了数组中每个位置的元素时，只需要修改对应key的HTML元素即可，其他HTML元素保持不变！——效率高！

c. 总结: 避免修改数组元素时，重新生成所有HTML元素，而是只更新其中一个HTML元素即可！提高修改效率！

d. 如何:

1). 当遍历数组时: <元素 v-for="(val,i) of 数组" :key="i"

Key的值依次是0 1 2 3..

2). 当遍历对象时: <元素 v-for="(val,key) of 对象" :key="key"

Key的值依次是: 属性名1 属性名2 ...

因为一个对象内的属性名肯定不会重复，所以，属性名也可以当做:key唯一标识一个HTML元素

2. 绑定HTML片段内容:

(1). 问题: {{}}不能绑定HTML片段内容。

因为: {{}}本质相当于DOM中的textContent，会将HTML内容原样显示，不会被编译！

(2). 解决: 今后只要要绑定的变量内容是一段HTML片段时，都用v-html来绑定！

(3). 如何:<元素 v-html="包含HTML内容的变量或表达式">

(4). 强调:

a. v-html会将绑定内容中的HTML内容，编译后再显示给人看

b. v-html也是指令，所以v-html后的""中可以写js表达式，比如字符串拼接！

c. 用了v-html，就不要再元素开始标签和结束标签直接写内容！因为会被v-html内容替换！

(5). 示例: 绑定HTML内容

|  |
| --- |
| <div id="app">  <h1>消息来源:{{html}}</h1>  <h1 v-html="'消息来源:'+html">  Welcome  </h1>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  html:`<p>来自<a href="javascript:;">&lt;&lt;新华社&gt;&gt;</a>的消息</p>`  }  })  </script> |

3. 防止用户短暂看到{{}}

|  |
| --- |
| 问题: 因为vue代码是放在js文件中，所以，如果网速慢，vue代码暂时没有下载下来时，用户很可能短暂看到页面上的绑定语法，用户体验不好！  解决: 2个办法: |

(1). 用v-cloak暂时隐藏带有{{}}内容的元素:

a. 2步:

1). 在包含绑定语法{{}}的元素上添加v-cloak属性

2). 在css中手动添加样式: [v-cloak]{ display:none }

b. 原理：

1). 用属性选择器查找所有带有v-cloak属性的元素，暂时隐藏

2). 当new Vue()渲染完成时，自动找到所有v-cloak属性，自动移除。

c. 示例: 使用v-cloak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>Welcome: {{uname}}</h1>  </div>  <script>  setTimeout(function(){  var vm=new Vue({  el:"#app",  data:{  uname:"dingding"  }  })  },2000)  </script>  </body> |

d. 问题: 既要在HTML中写指令，又要手动添加css选择器，步骤繁琐的！

(2). 用v-text代替内容中{{}}语法，来绑定非HTML片段内容:

a. <元素 v-text="原{{}}内容"></元素>

b. 原理:

1). 因为绑定语法写在了元素的属性里，所以，如果不是vue帮忙，用户无论如何是看不到元素属性中的内容的！

2). New Vue()读取到v-text时，会解析v-text的内容，替换元素开始标签和结束标签之间的内容

c. 强调:

1). 和v-html不同，v-text等效于{{}}等效于DOM中的textContent，所以如果v-text中包含HTML片段，是不会被编译，而是原样显示给人看！

2). v-text也是指令，所以v-text后的""中也可以写js表达式，比如字符串拼接！

3). 用了v-text，也不要在元素开始标签和结束标签直接写内容！因为会被v-text内容替换！

d. 示例: 使用v-text防止用户短暂看到{{}}

|  |
| --- |
| <div id="app">  <h1 v-text="'Welcome:'+uname"></h1>  </div>  <script>  setTimeout(function(){  var vm=new Vue({  el:"#app",  data:{  uname:"dingding"  }  })  },2000)  </script> |

4. 事件绑定:

(1). 标准写法:

a. div#app中: <元素 v-on:事件名="处理函数()"

b. new Vue()内:

methods:{

处理函数(形参变量){

//可用"this.属性名"方式访问data中的模型变量

}

}

(2). 简写:

a. v-on: 可用@简写

<元素 @事件名="处理函数()"

b. 如果处理函数不需要传参，()可省略

<元素 @事件名="处理函数"

(3). 示例: 点击按钮，数量增长

|  |
| --- |
| <div id="app">  <button @click="change(-1)">-</button>  <span>{{n}}</span>  <button @click="change(+1)">+</button>  </div>  <script>  var data={ n:0 }  var vm=new Vue({  el:"#app",  data:data,  methods:{  //本例中，因为页面上只需要一个change函数，所以methods中只添加一个change函数  //但是页面上调用change()函数时，传入了一个参数值,所以定义change函数时需要定义一个形参来接住实参值  change(i){  //如果i==+1,说明本次想+1  if(i==+1){  this.n++;  }else{//否则如果i!=+1，说明本次想-1  //只有n>0时，才能-1  if(this.n>0){  this.n--;  }  }  }  }  });  </script> |

(4). 事件对象:

a. 回顾: DOM中事件对象总是作为事件处理函数的第一个参数值默认传入:

元素.on事件名=function(e){

e->event

e.stopPropagation()

e.preventDefault()

e.target

e.offsetX e.offsetY

e.clientX e.clientY

}

b. vue中如何获得事件对象: (vue中如何获得鼠标位置)

1). 如果事件处理函数不需要传入实参值时，则:

事件对象也是作为处理函数第一个参数值自动传入，也是在函数定义时，用一个形参e，就可接住——同DOM

示例: 使用e获得鼠标位置:

|  |
| --- |
| <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>  div{  width:300px; height:100px;  margin:20px;  }  #d1{  background-color:#aaf  }  #d2{  background-color:#ffa  }  </style>  <script src="js/vue.js"></script>  </head>  <body>  <div id="app">  <div id="d1" @click="doit">d1</div>  <div id="d2">d2</div>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{    },  methods:{  doit(e){//同DOM的e  console.log(`点在d1的: x:${e.offsetX},y:${e.offsetY}`);  }  }  })  </script>  </body> |

2). 鄙视: 如果既想传入自定义实参值，又想获得事件对象:

i. 借助$event关键字: vue框架内部内置的专门指向事件对象的关键词。用$event等效于用事件对象e

ii. 调用函数时，可将$event和其他实参值一起传入函数中

iii. 定义函数时，可用普通的形参变量接住$event的值。

iv. 示例: 使用$event获得鼠标位置并传入自定义实参值

|  |
| --- |
| <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>  <style>  div{  width:300px; height:100px;  margin:20px;  }  #d1{  background-color:#aaf  }  #d2{  background-color:#ffa  }  </style>  <script src="js/vue.js"></script>  </head>  <body>  <div id="app">  <div id="d1" @click="doit('d1',$event)">d1</div>  <div id="d2" @click="doit('d2',$event)">d2</div>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{    },  methods:{  doit(name,e){//同DOM的e  console.log(`点在${name}的: x:${e.offsetX},y:${e.offsetY}`);  }  }  })  </script>  </body> |

5. 仅在首次渲染页面时绑定一次，即使之后模型变量再改变，也不会自动更新页面: <元素 v-once>

示例: 显示时间

|  |
| --- |
| <h1>当前系统时间: {{time}}</h1>  <!--希望上线时间只在首次打开网页时绑定一次，之后，即使time变量值发生变化，也不会自动更新上线时间-->  <h1 v-once>上线时间: {{time}}</h1>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  time:new Date().toLocaleString()  }  })  setInterval(function(){  vm.time=new Date().toLocaleString()  },1000)  </script> |

6. 防止元素内容中的{{}}被vue编译，让内容中的{{}}原样显示！

<元素 v-pre> xxx{{xx}}xxx </元素>

二. 双向绑定:

1. 单向绑定: 只能将data中的变量值，自动同步更新到HTML页面中。但是，页面上的修改，无法自动更新回data的变量中。

(data ---> div#app 但是 div#app -x-> data)

2. 双向绑定: 既能将data中的变量值，自动同步更新到HTML页面中。又能讲页面上的修改自动更新回data的变量中。

(data <===> div#app)

3. 何时使用双向绑定: 只有绑定表单元素时，才有必要用双向绑定！因为只有表单元素，用户才能在页面上修改的它的内容。

4. 如何: 每种表单元素绑定的原理不同:

1). 文本框/文本域: 绑定的是value属性

a. <input v-model:value="变量"/>

<textarea v-model:value="变量"></textarea>

b. 原理: new Vue()扫描到这里时，自动为当前元素绑定事件，比如，如果<input type="text" v-model:value="变量">，就会翻译为: oninput="vm.变量=当前文本框的新value"。

c. 示例: 点按钮，按回车，在文本框中输入内容，都可获得文本框中输入的关键词，执行搜索操作。

|  |
| --- |
| <div id="app">  <!--在文本框上按回车可以查找-->  <input type="text" v-model:value="keywords" @keyup="myKeyUp">  <!--点击按钮可以查找-->  <button @click="search">百度一下</button>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  keywords:"macbook i5"  },  methods:{  search(){  console.log(`查找 ${this.keywords} 相关的内容...`)  },  // $event  // ↓  myKeyUp(e){  //只有按回车才查找  if(e.keyCode==13){  //调用旁边的search()函数  this.search();  }  }  },  //想变量keywords只要被更改，就重新执行一次搜索  watch:{  keywords(){  console.log("自动调用一次watch中的keywords函数...")  this.search();  }  }//进入new Vue()中的一切data:{}, methods:{}, watch:{}都会被打散，最终都直接隶属于new VUe()对象，都是平级的，所以可以this.方式，互访。  })  </script> |

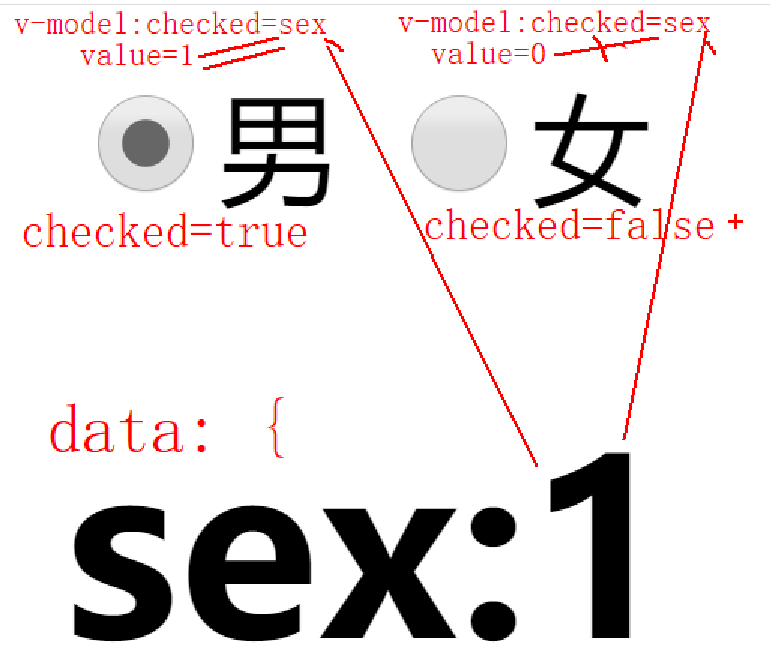
(2). 单选按钮: 绑定的是checked属性

a. <input type="radio" name="sex" value="1" v-model:checked="变量">男

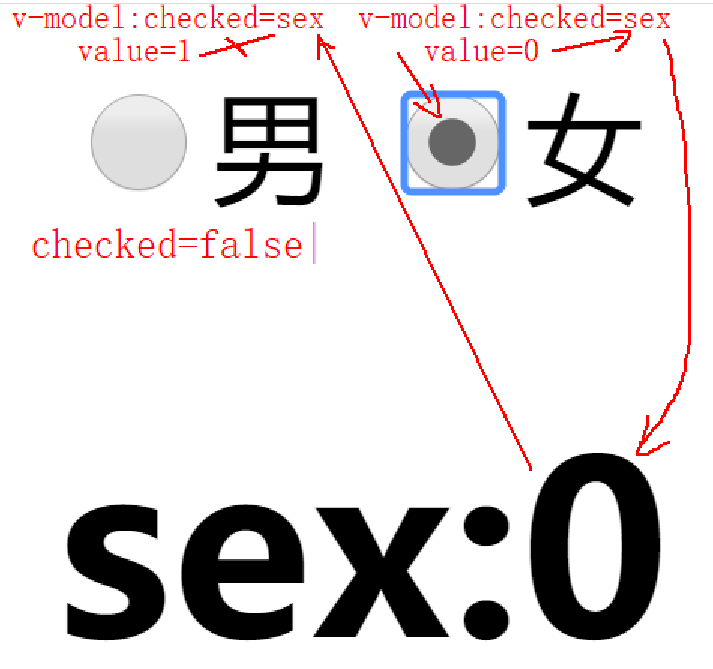
<input type="radio" name="sex" value="0" v-model:checked="变量">女

b. 原理:

(1). 从data->input: 用checked绑定的变量的值和当前radio的value做比较。如果checked绑定的值和value值相等，则当前radio选中。否则不选中。



(2). 切换选中一个radio后，用当前选中的radio的value值代替checked属性绑定的data中的变量值。导致，页面中其它关注这个变量的位置都自动发生改变。



c. 示例: 选择性别

|  |
| --- |
| <div id="app">  <label><input type="radio" name="sex" value="1" v-model:checked="sex">男</label>  <label><input type="radio" name="sex" value="0" v-model:checked="sex">女</label>  <h1>sex:{{sex}}</h1>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  sex:1  }  })  </script> |

(3). 双向绑定select元素: 因为改变select的选中项，改变的是中整个select的value值，所以应该绑定select元素的value

a. <select v-model:value="变量">

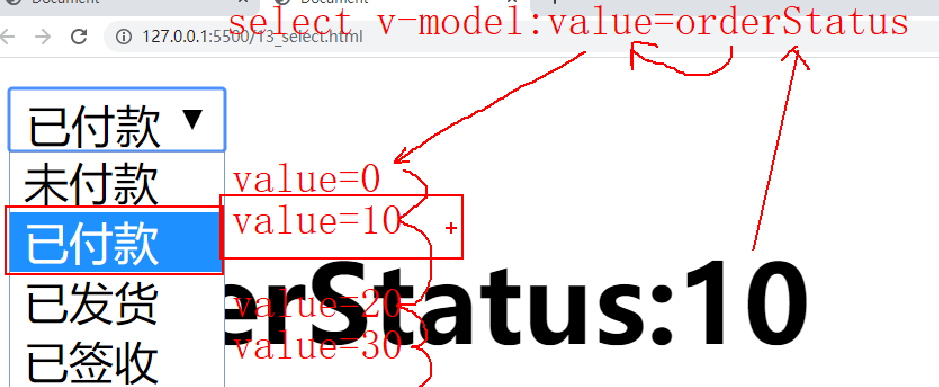
<option value="值1">文本1</option>

<option value="值2">文本2</option>

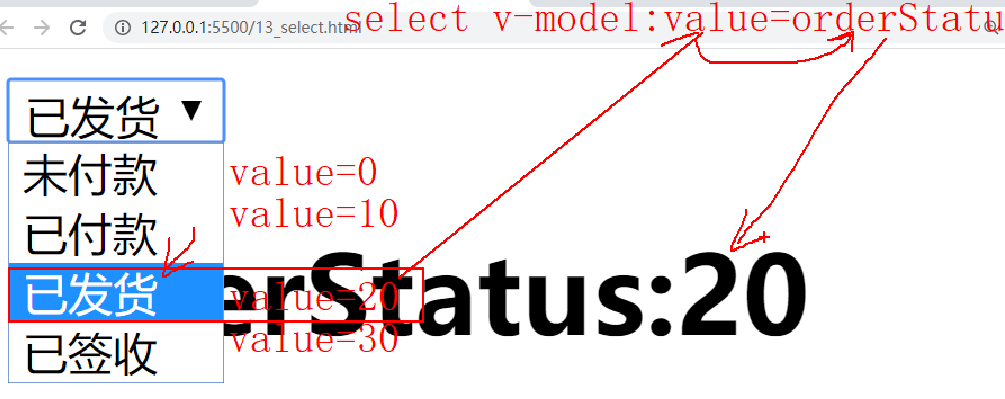
... ...

b. 原理:

1). 从data->select:用value绑定变量值，和每个option的value做比较，哪个option的value等于变量值，就选中哪个option



2). 切换select中的选中项: 用新选中的option的value值替换select的value绑定的data中的变量值。导致所有关系这个变量的其它位置自动变化。



c. 示例: 选择订单状态:

|  |
| --- |
| <div id="app">  <select v-model:value="orderStatus">  <option value="0">未付款</option>  <option value="10">已付款</option>  <option value="20">已发货</option>  <option value="30">已签收</option>  </select>  <h1>orderStatus:{{orderStatus}}</h1>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  //0:未付款 10:已付款 20:已发货 30:已签收  orderStatus:10  }  })  </script> |

(4). 双向绑定一个checkbox的情况: 因为选中或取消选中checkbox改变的它的checked属性，所有v-model应该绑定checked属性。且绑定的变量应该是bool类型

a. <input type="checkbox" v-model:checked="变量">

b. 原理:

1). 从data->input checkbox: 根据变量是true还是false，设置当前checkbox是否选中

2). 当切换checkbox的选中状态后，就用新的checked属性值(true或false)，更新到绑定的data中的变量中

c. 示例: 同意，启用表单元素，不同意，则禁用表单元素





|  |
| --- |
| <div id="app">  <input type="text" placeholder="请输入用户名" :disabled="!isAgree"><br/>  <input type="password" placeholder="请输入密码" :disabled="!isAgree"><br/>  <label><input type="checkbox" v-model:checked="isAgree">同意</label></br>  <button :disabled="!isAgree">注册</button>  <h1>isAgree:{{isAgree}}</h1>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  isAgree:false //表示不同意  }  })  </script> |

(5). 简写: 其实以上所有v-model后的":属性名"都可省略！v-model可自动根据所在的元素不同，选择对应的的元素自动绑定。

5. 监控函数:

(1). 双向绑定过程中只要希望变量值一发生变化，就自动执行一项操作，可用watch，添加监控函数。

(2). new Vue({

el:"#app",

data:{ 变量: 值, ... },

methods:{

函数(){ ... }

},

watch:{ //监视

要监视的变量(){ //只要data中的变量被修改，watch中与变量同名的监视函数，就会被自动执行！

... ...

}

}

})

(3). 强调: watch中的函数名必须和要监视的变量同名！

总结:

1. 元素内容需要动态改变: {{变量或js表达式}}

2. 元素属性值需要动态改变: :属性名="变量或js表达式"

3. 控制一个元素显示隐藏: 首选 v-show="条件"

4. 控制两个元素二选一显示:

<元素1 v-if="条件">

<元素2 v-else>

5. 控制多个元素多选一显示:

<元素1 v-if="条件1">

<元素2 v-else-if="条件2">

... ...

<元素n v-else>

6. 反复生成多个相同结构的HTML元素时:

<元素 v-for="(value,i) of 数组/字符串/对象/数字" :key="i">

7. 绑定HTML片段内容:

<元素 v-html="包含HTML内容的变量或表达式"></元素>

8. 防止用户短暂看到{{}}语法:

(1). <style>[v-cload]{display:none}</style>

<元素 v-cloak>  
 (2). <元素 v-text="包含{{}}的js表达式"></元素>

9. 绑定事件: <元素 @事件名="处理函数(参数值,$event)"

10. 只在首次加载页面时绑定一次: v-once

11. 阻止内容中的{{}}被vue编译: v-pre

12. 只要绑定表单元素的值，都用双向绑定v-model

<表单元素 v-model="变量">

今日网盘地址:

https://pan.baidu.com/s/1FkOP8DTlH4w4Y-wz518IFQ

作业：

1. 看视频学习v-for制作购物车:

homework/vue\_cart.mp4

2. 看视频学习选飞机案例:

homework/radio\_plains.mp4

3. 看视频学习笔试题，模拟实现vue绑定原理:

homework/vue\_bind.mp4

4. 看视频学习js模块化开发:

homework/modules

5. 预习: 1\_组件(component)，2\_组件化(components)开发，3\_SPA应用

preview文件夹