正课:

1. 绑定样式

2. 自定义指令

3. 计算属性

4. 过滤器

5. axios

一. 绑定样式:

1. 绑定内联样式

(1). 不好的做法: 将元素的整个style属性当做一个字符串来绑定

<元素 style="固定不变的css属性" :style="变量">

data:{

变量:"可能变化的css属性" //比如: "top:50px; left:100px"

}

缺点: 不便于修改其中某一个css属性。

(2). 好的做法: 将元素的style属性当做一个对象来绑定:

优点: 便于修改单个css属性

a. 只创建一个变量，但是变量值以对象方式保存多个css属性

<元素 style="固定不变的css属性" :style="变量">

data:{

变量:{

Css属性: 值,

... : ...

}

//自动翻译为: 一个字符串: "css属性:值; css属性:值; ..."

}

:style动态绑定的变量中的css属性，经过编译为字符串后，自动和不带:的固定不变的style中的css属性最终合并为一个style属性。

示例: 用键盘上下左右方向键控制一个小方块的移动:

|  |
| --- |
| <div id="app">    <div id="pop" style="position:fixed; width:100px; height:100px; background-color:pink" :style="popStyle"></div>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      popStyle:{        top:"0px",        left:"0px"      }      //自动翻译为: popStyle:"top:0px; left:0px"    }  })  window.onkeydown=function(e){    //左: 37  上: 38  右: 39  下: 40    if(e.keyCode==39){//右      var left=parseInt(vm.popStyle.left);      left+=20;      vm.popStyle.left=left+"px";    }else if(e.keyCode==40){//下      var top=parseInt(vm.popStyle.top);      top+=20;      vm.popStyle.top=top+"px";    }  }  </script> |

b. 为每个动态变化的css属性都创建一个变量（创建多个变量）,绑定时，""中使用匿名对象:style="{css属性: 变量1, css属性:变量2 }"

<元素 style="固定不变的css属性" :style="{css属性1: 变量1, css属性2: 变量2 , ...}">

data:{

变量1: 值1,

变量2: 值2

}

结果: :style="{css属性1: 变量1, css属性2: 变量2 , ...}"，会被自动翻译为:

style="css属性1:变量1的值; css属性2: 变量2的值"，再和其他写死的不带:的style合并

示例: 用键盘上下左右方向键控制一个小方块的移动:

|  |
| --- |
| <div id="app">    <div id="pop" style="position:fixed; width:100px; height:100px; background-color:pink" :style="{top, left}"></div>    <!--自动翻译为: top:0px; left:0px;-->  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      top:"0px",      left:"0px"    }  });  window.onkeydown=function(e){    //左: 37  上: 38  右: 39  下: 40    if(e.keyCode==39){//右      var left=parseInt(vm.left);      left+=20;      vm.left=left+"px";    }else if(e.keyCode==40){//下      var top=parseInt(vm.top);      top+=20;      vm.top=top+"px";    }  }  </script> |

2. 绑定class

(1). style绑定内联样式的问题: 多数样式的修改，都是同时修改多个css属性，如果用style绑定，每个css属性都要写出来，代码繁琐！

(2). 解决: 将来只要批量修改一个元素的多个css属性，应该用class方式，代替style方式

(3). 如何:

a. 不好的做法: class属性也可以作为一整条字符串绑定，但是问题依然是不便于只修改其中一个class。

b. 好的做法，将class作为一个对象绑定：

1). 只声明一个变量，但是变量值是一个对象，其中包含多个class名

i. <元素 class="固定不变的class" :class="变量"

ii. data:{

变量: { class1 : true或false, class2: true或false }

}

iii. 结果: 编译时，仅将变量对象中值为true的class，编译进最终的class中，值为false的class，不包含在最终的正式class字符串中。且动态绑定的:class会和不带:绑定的固定不变的class，合并为一个class！

iv. 示例: 验证手机号格式:

|  |
| --- |
| <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>    <style>      /\* 定义提示框的基础样式 \*/      .msg{        display:inline-block;        width:160px;        height:25px;        border:1px solid #555;        text-align:center;        line-height:25px;      }      /\* 定义提示框在验证通过时的样式 \*/      .success{        border:1px solid green;        background-color:lightgreen;        color:green      }      /\* 定义提示框在验证失败时的样式 \*/      .fail{        border:1px solid red;        background-color:pink;        color:red      }    </style>  </head>  <body>  <div id="app">    <!-- 因为要获得用户输入的手机号进行验证，所有必须用双向绑定 -->    <input type="text" v-model="phone">    <!-- 因为提示框的样式可能在success和fail之间来回切换，所以动态绑定span的部分class        又因为提示框的内容也可能随验证结果而动态变化，所以也要绑定一个变量msg-->    <span class="msg" :class="msgClass">{{msg}}</span>    <!--界面中共需要三个变量-->  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      //因为页面中需要三个变量，所以data中就要有三个变量      phone:"", //实时接用户在文本框中输入的手机号      msg:"", //根据验证结果的对错，绑定提示信息的内容      //根据验证结果的对错，动态在success和fail两个class之间来回切换。      msgClass:{        success:false,        fail:false      }    },    watch:{      //因为用户一边输入，vue就一边验证，所以必须用watch随时监控用于的输入变化。      phone(){        //定义正则验证phone的内容        var reg=/^1[3-9]\d{9}$/;        //如果用正则验证phone的内容通过        if(reg.test(this.phone)==true){          //就修改msgClass应用成功的样式，不应用失败的样式          this.msgClass={            success:true,            fail:false          }          //修改span的提示信息内容          this.msg="手机号格式正确！"        }else{//否则如果验证失败          //就修改msgClass应用失败的样式，不应用成功的样式          this.msgClass={            success:false,            fail:true          }          //修改span的提示信息内容          this.msg="手机号格式正确！"        }      }    }  })  </script>  </body> |

2). 为每个class都声明一个变量，在绑定时，可用{}对象语法绑定:

i. <元素 class="固定不变的class" :class="{class1: 变量1, class2: 变量2, ...}"

ii. data:{

变量1: true或false,

变量2: true或false

}

iii. 结果: 编译时，仅将:class="{}"中值为true的class，编译进最终的class中，值为false的class，不包含在最终的正式class字符串中。且动态绑定的:class会和不带:绑定的固定不变的class，合并为一个class！

iv. 示例: 验证手机号格式:

|  |
| --- |
| <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>    <style>      /\* 定义提示框的基础样式 \*/      .msg{        display:inline-block;        width:160px;        height:25px;        border:1px solid #555;        text-align:center;        line-height:25px;      }      /\* 定义提示框在验证通过时的样式 \*/      .success{        border:1px solid green;        background-color:lightgreen;        color:green      }      /\* 定义提示框在验证失败时的样式 \*/      .fail{        border:1px solid red;        background-color:pink;        color:red      }    </style>  </head>  <body>  <div id="app">    <!-- 因为要获得用户输入的手机号进行验证，所有必须用双向绑定 -->    <input type="text" v-model="phone">    <!-- 因为提示框的样式可能在success和fail之间来回切换，所以动态绑定span的部分class        又因为提示框的内容也可能随验证结果而动态变化，所以也要绑定一个变量msg-->    <!--                       class名  变量-->    <!-- <span class="msg" :class="{success:success, fail:fail}">{{msg}}</span> -->    <!--结果: 哪个class名后的变量值为true，才会进入最终的class字符串中。变量值为false的class，不会出现在最终的class字符串中-->    <span class="msg" :class="{success, fail}">{{msg}}</span>    <!--界面中共需要四个变量-->    <!--但是，两个class和错误提示三个变量的值，都和验证结果这一个数据有关！所有，其实只用一个变量就可控制三个值的变化-->  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      //因为页面中需要三个变量，所以data中就要有三个变量      phone:"", //实时接用户在文本框中输入的手机号      msg:"", //根据验证结果的对错，绑定提示信息的内容      //根据验证结果的对错，动态在success和fail两个class之间来回切换。      success:false,      fail:false    },    watch:{      //因为用户一边输入，vue就一边验证，所以必须用watch随时监控用于的输入变化。      phone(){        //定义正则验证phone的内容        var reg=/^1[3-9]\d{9}$/;        //如果用正则验证phone的内容通过        if(reg.test(this.phone)==true){          //就修改msgClass应用成功的样式，不应用失败的样式          this.success=true;          this.fail=false          //修改span的提示信息内容          this.msg="手机号格式正确！"        }else{//否则如果验证失败          //就修改msgClass应用失败的样式，不应用成功的样式          this.success=false;          this.fail=true          //修改span的提示信息内容          this.msg="手机号格式正确！"        }      }    }  })  </script>  </body> |

iv. 示例: 优化: 尽量减少vue中data中的变量，便于维护。

|  |
| --- |
| <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>    <style>      /\* 定义提示框的基础样式 \*/      .msg{        display:inline-block;        width:160px;        height:25px;        border:1px solid #555;        text-align:center;        line-height:25px;      }      /\* 定义提示框在验证通过时的样式 \*/      .success{        border:1px solid green;        background-color:lightgreen;        color:green      }      /\* 定义提示框在验证失败时的样式 \*/      .fail{        border:1px solid red;        background-color:pink;        color:red      }    </style>  </head>  <body>  <div id="app">    <!-- 因为要获得用户输入的手机号进行验证，所有必须用双向绑定 -->    <input type="text" v-model="phone">    <!-- 因为提示框的样式可能在success和fail之间来回切换，所以动态绑定span的部分class        又因为提示框的内容也可能随验证结果而动态变化，所以也要绑定一个变量msg-->    <!--                       class名  变量-->    <!-- <span class="msg" :class="{success:success, fail:fail}">{{msg}}</span> -->    <!--结果: 哪个class名后的变量值为true，才会进入最终的class字符串中。变量值为false的class，不会出现在最终的class字符串中-->    <!-- <span class="msg" :class="{success:isRight, fail:isRight==false}">{{isRight==true?"手机号可用":"手机号格式错误！"}}</span> -->    <span class="msg" :class="isRight==true?'success':'fail'">{{isRight==true?"手机号可用":"手机号格式错误！"}}</span>    <!--界面中共需要四个变量-->    <!--但是，两个class和错误提示三个变量的值，都和验证结果这一个数据有关！所有，其实只用一个变量就可控制三个值的变化-->  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      //因为页面中需要三个变量，所以data中就要有三个变量      phone:"", //实时接用户在文本框中输入的手机号      isRight:false    },    watch:{      //因为用户一边输入，vue就一边验证，所以必须用watch随时监控用于的输入变化。      phone(){        //定义正则验证phone的内容        var reg=/^1[3-9]\d{9}$/;        //将验证结果保存到变量isRight中，依次牵连着三个位置发生变化！        this.isRight=reg.test(this.phone);      }    }  })  </script>  </body> |

二. 自定义指令:

1. 向Vue大家庭中添加自定义指令:

Vue.directive("指令名", { //向Vue大家庭中添加一个新的指令，并制定指令名

//回调函数: 当渲染后的元素被插入到DOM树上之后，自动执行该回调函数

inserted(当前指令所在的DOM元素对象){

//对当前指令所在的DOM元素对象，执行DOM操作

//在这里对当前DOM元素执行的DOM操作，会自动应用到页面上

}

})

强调: 添加自定义指令时，指令名一定不要带v-前缀！比如，想添加一个让元素自动获得焦点的自定义指令，可命名为"focus"

2. 使用自定义指令: 只要在想应用对应效果的元素上，添加"v-指令名"属性即可。

强调: 虽然定义指令时，指令名没有加v-前缀，但是使用指令时，必须加v-前缀

3. 原理:

(1). Vue.directive("指令名",{

inserted(domElem){

对domElem执行原生DOM操作

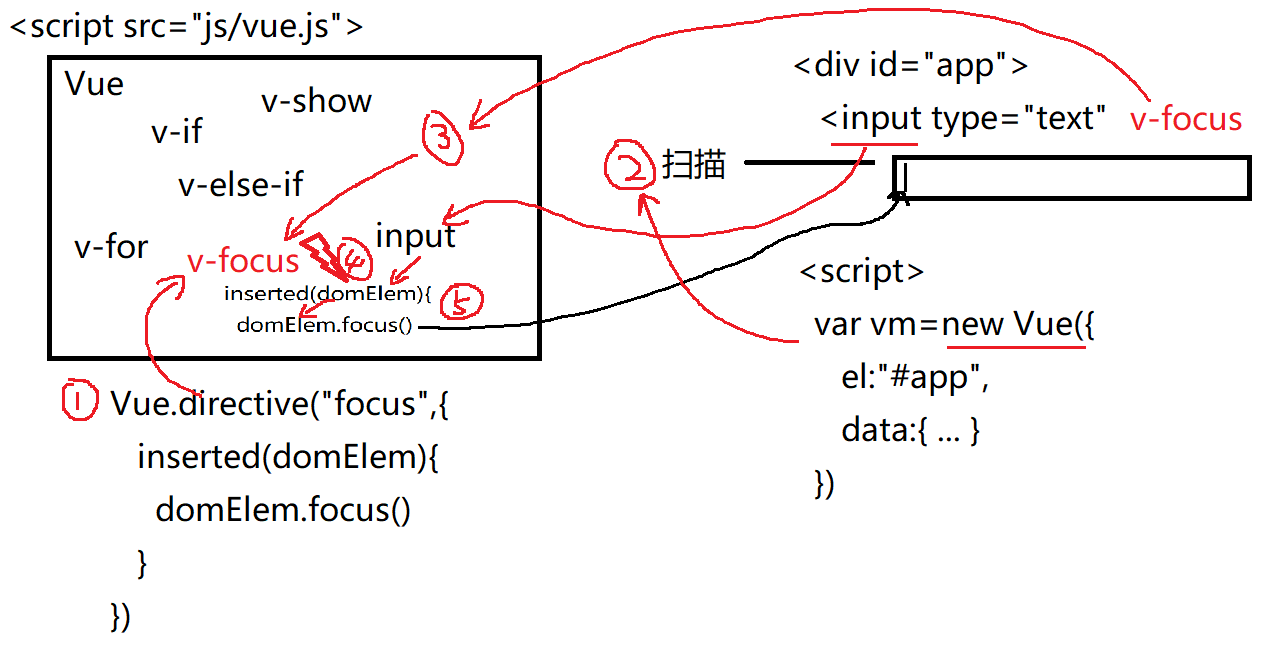
}

})

向Vue大家庭中添加一个新的自定义指令，关联一个处理函数

(2). new Vue()时，会扫描<div id="app">下的所有内容。

(3). 每扫描到一个v-开头的指令属性，就会回Vue大家庭中找是否有对应的指令。如果有对应的指令，就调用指令关联的处理函数，对指令所在的元素执行原生DOM操作。



4. 示例: 让文本框自动获得焦点:

|  |
| --- |
| <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("focus",{        inserted(domElem){          //让当前元素自动获得焦点          domElem.focus();          //      DOM原生        }      })    </script>  </head>  <body>  <div id="app">    <input type="text" v-focus><button>百度一下</button>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      }  })  </script>  </body>  强调: new Vue()起到最重要的扫描HTML内容的作用，只有new Vue()才认识v-开头的指令，所有，即使data中没有值，只要用到了Vue相关的功能，new Vue()都不能省略！ |

三. 计算属性：

1. 什么是: 程序中没有保存该属性的属性值，每次绑定属性时，都需要根据其他属性的值动态计算获得该属性的属性值。

2. 为什么: 有些值，总是可以根据其他属性值的变化，动态计算获得。这样的值，就没必要在程序中再保存一份。因为就算保存了，这个属性所依赖的其他属性值一旦发生变化，这个属性的属性值连带也要改变。

3. 何时: 今后，只要一个属性的值可以根据其他属性计算出来，都没必要保存！

4. 如何:

(1). 定义计算属性:

new Vue({

el:"#app",

data:{

页面所需的变量

},

methods:{

自定义函数或事件处理函数

},

watch:{

监视某个变量的监视函数

},

computed:{

属性名(){

return 根据其他属性值计算后获得的计算结果

}

}

})

(2). 使用计算属性: 计算属性，虽然本质是一个函数，但是在HTML中绑定语法中使用时，不要加()!

5. 结果：

(1). 只要计算属性所依赖的另一个属性值发生改变，同样会通知计算属性重新计算新的属性值。

(2). 计算属性计算出的结果，会被Vue缓存起来，反复使用，避免重复计算。即使反复使用多次，也只在首次计算一次。

6. 鄙视: methods vs computed

(1). 用法:

a. methods必须加()

b. computed 不用加()

(2). 反复使用:

a. methods中的方法，每调用一次，就会重新执行一次，执行结果，不会被vue缓存起来。

b. computed中的计算属性，只在第一次使用时计算一次，然后，结算结果，就会被Vue缓存起来，即使在别的地方反复使用这个计算属性，也不会重复计算，而是直接从缓存中获取值。但是，当所依赖的其他属性值发生变化时，计算才被迫重新计算一次。

7. 如何选择methods和computed:

(1). 如果这次使用时，更关心函数的执行结果数据时，首选计算属性

(2). 如果这次使用时，更关心函数执行操作的过程，结果数据无所谓，甚至函数没有返回值，则首选methods中的方法。

8. 示例: 使用计算属性绑定购物车总价:

|  |
| --- |
| <div id="app">    <h3>总计: ¥{{total.toFixed(2)}}</h3>    <ul>      <li v-for="(item,i) of cart" :key="i">        {{item.pid}} | {{item.pname}} | ¥{{item.price.toFixed(2)}} | {{item.count}} —— 小计：¥{{(item.price\*item.count).toFixed(2)}}      </li>    </ul>    <h3>总计: ¥{{total.toFixed(2)}}</h3>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      cart:[        {pid:1, pname:"华为",price:4455, count:2},        {pid:2, pname:"小米",price:3455, count:1},        {pid:3, pname:"OPPO",price:3356, count:3},      ]    },    methods:{ //叫方法，所以用法同方法的用法——必须加()      },    computed:{ //叫属性，所以用法同属性的用法——不加()      total(){        console.log("调用了一次total()");        var sum=0;        for(var p of this.cart){          sum+=p.price\*p.count;        }        return sum;      }    }  })  </script> |

四. 过滤器:

1. 什么是: 专门对变量的原始值进行加工后，再显示的特殊函数

2. 为什么: 个别变量的原始值不能直接给人看！

比如: 性别 0和1 日期的毫秒数

3. 何时: 如果一个变量的值不能直接给人看时，必须经过加工，才能给人看时

4. 如何:

(1). 向Vue大家庭中添加过滤器函数

Vue.filter("过滤器名字", function(oldVal){ //接受一个变量的原始值

return 根据oldVal的不同，动态返回的新值

})

(2). 在绑定语法中使用过滤器函数：

{{变量 | 过滤器名 }}

5. 结果:

(1). 变量的原始值不会立刻显示出来，而是先交给|后的过滤器函数

(2). 再将过滤器处理后的返回值，返回出来，显示在元素内容中

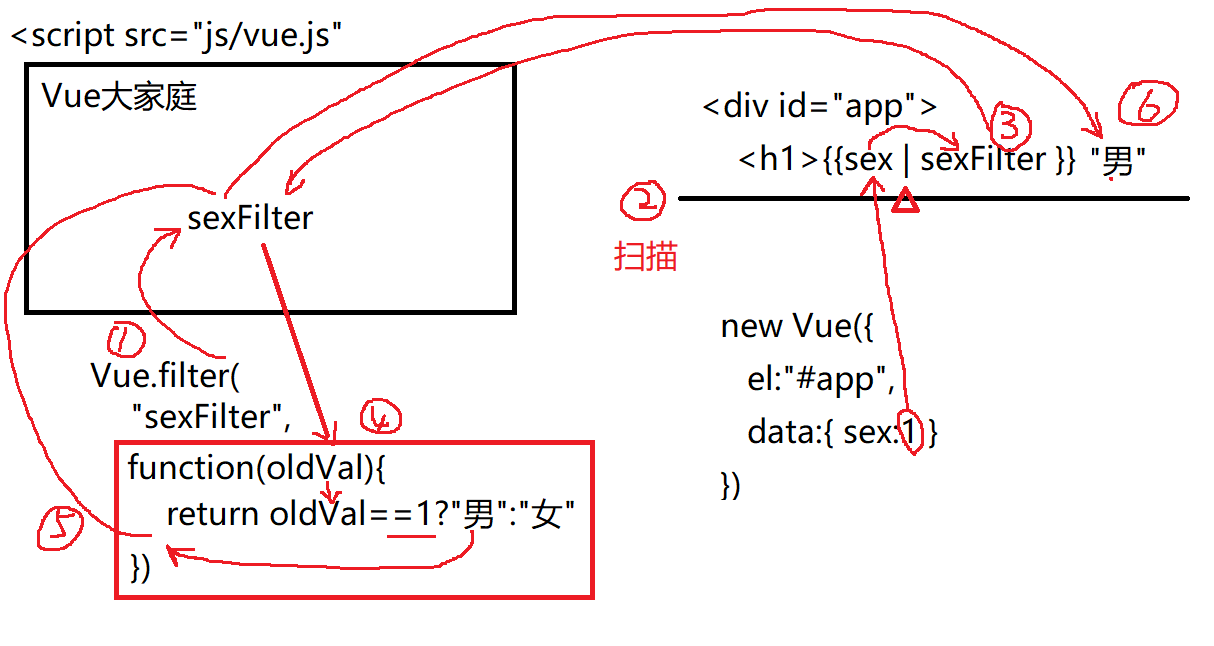
6. 原理:

(1). Vue.filter("过滤器名", function(oldVal){ return 新值 })

定义一个过滤器函数，加入到Vue大家庭中备用

(2). 当new Vue()扫描到{{}}中的|时，会回Vue大家庭中查找相应名称的过滤器函数。

(3). 只要找到，就先将|前的变量原始值，交给过滤器函数的oldVal参数，经过过滤器函数的加工，返回新值。显示到当前绑定语法的位置。



7. 示例: 过滤性别的1和0为男和女

|  |
| --- |
| <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>      //sexFilter=function(){ ... }      Vue.filter("sexFilter",function(oldVal){        //性别接住的旧值可能是1或0        return oldVal==1?"男":"女"      })    </script>  </head>  <body>  <div id="app">    <h1>性别: {{sex | sexFilter}}</h1>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      sex:1    }  })  </script>  </body> |

8. 过滤器可以加参数:

(1). 定义过滤器时:

Vue.filter("过滤器名",function(oldVal, 自定义形参, ...){

Return 新值

})

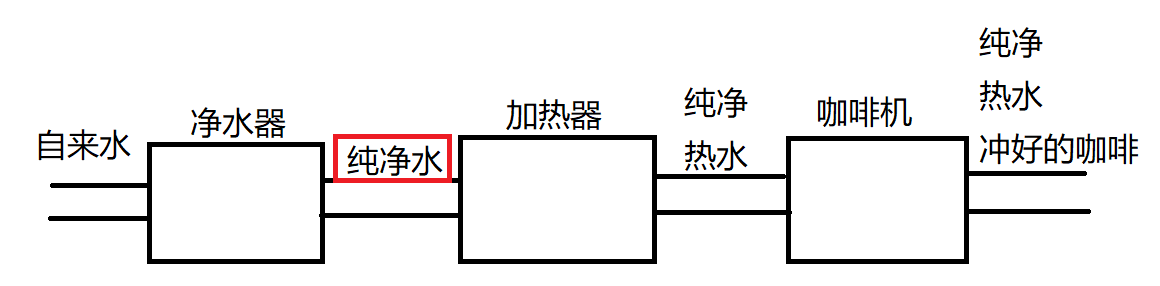
(2). 使用过滤器时:

{{变量 | 过滤器名(自定义实参, ...) }}

(3). 示例: 根据参数值返回不同语言的性别

|  |
| --- |
| <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>    <script>      //sexFilter=function(){ ... }      Vue.filter("sexFilter",function(oldVal,language){        //性别接住的旧值可能是1或0        //language参数可能接住cn或en,其中，默认是cn        if(language=="en"){          return oldVal==1?"Male":"Female"        }else{          return oldVal==1?"男":"女"        }      })    </script>  </head>  <body>  <div id="app">    <h1>性别: {{sex | sexFilter}}</h1>    <h1>性别: {{sex | sexFilter("en")}}</h1>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      sex:1    }  })  </script> |

9. 过滤器可以连用:



(1). 绑定语法中: {{变量 | 过滤器1 |过滤器2 | ... }}

(2). 强调:

a. 后一个过滤器2，进入的旧值，已经不是变量的原始值了，而是前一个过滤器加工后的中间值。

b. 只有最后一个过滤器的返回值，才会显示到页面上。如果希望前几个过滤器的返回值也能一起显示到页面上，只能在最后一个过滤器中将新值拼接到上一步过滤器传入的旧值上。

(3). 示例：为性别再额外添加图标

|  |
| --- |
| <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>      //定义一个新的过滤器专门为性别添加图标      Vue.filter("sexIcon",function(oldVal){        //oldVal可能有六种：1  0  男  女   Male   Female        if(oldVal==1||oldVal==0){          return oldVal==1?"♂":"♀";        }else{          return oldVal=="男"||oldVal=="Male"?oldVal+"♂":oldVal+"♀";        }      });      //sexFilter=function(){ ... }      Vue.filter("sexFilter",function(oldVal,language){        //性别接住的旧值可能是1或0        //language参数可能接住cn或en,其中，默认是cn        if(language=="en"){          return oldVal==1?"Male":"Female"        }else{          return oldVal==1?"男":"女"        }      })    </script>  </head>  <body>  <div id="app">    <h1>性别: {{sex | sexFilter | sexIcon }}</h1>    <h1>性别: {{sex | sexFilter("en") | sexIcon }}</h1>    <h1>性别: {{sex | sexIcon }}</h1>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      sex:1    }  })  </script>  </body> |

五. axios:

1. 什么是axios: 基于Promise的专门发送ajax请求的函数库

2. 为什么: 总结发送ajax请求:

(1). xhr4步/6步

(2). 自己封装函数，考虑不全面

(3). jQuery中$.ajax: 问题，在vue中几乎不再使用DOM操作，几乎不用jQuery了。如果单是为了引入$.ajax函数而引入整个jQuery库，有点儿小题大做。

(4). Vue官方提供了一套发送ajax请求的组件: vue-resource，后来，Vue发现哪个框架都有自己的发送ajax请求就得函数，而且都大同小异，所以，Vue认为自己没有必要再重新开放按一套ajax函数库，所以vue-resource已经不再维护。

(5). Vue官方帮我们选了一个时髦好用的ajax函数库: axios，所以将来在框架中发送ajax请求，几乎都用axios。

3. 何时: 只要在Vue框架中，发送ajax请求服务器端数据，都用axios

4. 如何:

(1). 准备: 在项目中引入axios.js，才能引入axios函数库

<script src="js/axios.js"> 引入的顺序和vue.js无关

(2). 设置所有服务器端接口的公共域名部分

axios.defaults.baseURL="服务器端域名地址部分"

(3). 发送get请求:

axios.get("url",{ //向服务器端接口地址url发送get请求

params:{ //携带参数

参数1: 值1,

... : ...

}

//自动翻译为"?参数1=值1&参数2=值2&..."

//}).then(function(result){

}).then(res=>{

res.data才是服务器端返回的结果

})

(4). 发送post请求:

axios.post("url", "变量1=值1&变量2=值2&...")

.then(res=>{

res.data时服务器端返回的结果

})

(5). 示例: 使用axios向新浪云上的接口地址发送get和post请求，传参，并接受响应结果

|  |
| --- |
| <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/axios.min.js">      //axios={      //  get(){ ... },      //  post(){ ... }      //}    </script>    <script src="js/qs.min.js">      //专门就将对象语法转为查询字符串语法      //{ uname:"dingding", upwd:"123456"}      // ↓ Qs.stringify(对象)      //"uname=dingding&upwd=123456"    </script>  </head>  <body>    <script>    //先定义所有接口统一的服务器端域名部分    axios.defaults.baseURL="http://xzserver.applinzi.com"    //向服务器端请求学子商城首页商品数组，包含6个商品对象的信息    axios.get("/index")        .then(res=>{          console.log(res.data);        })    //想获取5号商品的详细信息: 要求: 携带一个lid参数，值为要查询的商品编号    axios.get("/details",{      params:{lid:5}    }).then(res=>{      console.log(res.data) //返回5号商品的详细信息    })    //用用户名dingding，密码123456，调用登录接口    axios.post("/users/signin",      //"uname=dingding&upwd=123456"      Qs.stringify({ uname:"dingding", upwd:"123456"})    ).then(res=>{      console.log(res.data);    })    </script>  </body> |

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-cloak]{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">

作业：看微信小程序中，"在线"栏目中的视频，预习和复习：

1. 预习Vue day03下: 1. 创建一个vue全局组件

2. 预习Vue day03下

2.1 Vue组件化开发todo ...

2.2 Vue组件化开发todo ...

2.3 Vue组件化开发todo ...

3. 看Vue day02下: 作业: 模块化1至7