Vue渐进式框架 2.0版本学习 框架和库很像

Vue标签：



API:

Vue 基本使用



<body>

    <div id="app">

    <div>{{msg}}</div>  <!--插值表达式 -->

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

    /\*vue的基本使用步骤

    1.需要提供标签用于填充数据

    2.引入vue.js库文件

    3.可以使用vue的语法做功能了

    4.把vue提供的数据填充到标签里面\*/

  var vm=new Vue({

      el:"#app",

      data:{

      msg:'hello vue'

      }

  })

El：元素的挂载位置（值可以是css 选择器 或者是dom元素）填充位置

Data：模型数据（值是一个对象） 填充需要的数据

{{}}: 插值表达式，可以将数据填充到html标签中，插值表达式支持基本的计算操作

计算操作举栗子：

  <div>{{1+2}}</div> 结果返回3

    <div id="app">

    <div>{{msg}}</div>  <!--插值表达式 -->

    <div>{{1+2}}</div>

    <div>{{msg+"----"+123 }}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

    /\*vue的基本使用步骤

    1.需要提供标签用于填充数据

    2.引入vue.js库文件

    3.可以使用vue的语法做功能了

    4.把vue提供的数据填充到标签里面\*/

  var vm=new Vue({

      el:"#app",

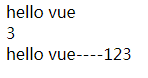
      data:{

      msg:'hello vue'

      }

  })

结果返回：



v码vue代码运行原理分析：vue代码-----vue框架----- 原生js代码 编译过程运行原

Vue的模板语法规则：

1. 前端渲染 （模板+数据（数据后端提供）=静态html内容）

模板语法概览 ：1.插值表达是 2.指令 3.事件绑定 4.属性绑定 5.样式绑定 6.分支循环结构

指令解释：指令的本质就是自定义属性，指定的格式：v-开始 （比如 v-cloak）

V-cloak指令语法：他可以解决插值的“闪动问题”，解决原理：先隐藏，替换好值后再显示最终的值

**指令如何使用：参考官网的API手册——指令 绑定指令一般用=**

1.提供样式  //方括号是标准的属性选择器

   [v-cloak]{

       display：none

   }

   2. 在插值表达式所在的标签中添加v-cloak指令

代码：div id="app">

        <div v-cloak>{{msg}}</div>

        <div  v-cloak>{{1+2}}</div>

        <div  v-cloak>{{msg+"\_\_\_\_"+123}}</div>

    </div>

    <style type="text/css">

   [v-cloak]{

       display: none;

   }

    </style>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"> </script>

   <script type="text/javascript">

   /\*v-cloak指令的用法

   1.提供样式  //方括号是标准的属性选择器

   [v-cloak]{

       display：none

   }

   2. 在插值表达式所在的标签中添加v-cloak指令

   背后原理：先通过样式隐藏内容，再内存中进行值的替换，替换后显示结果

   \*/

    var vm=new Vue({

     el:"#app",

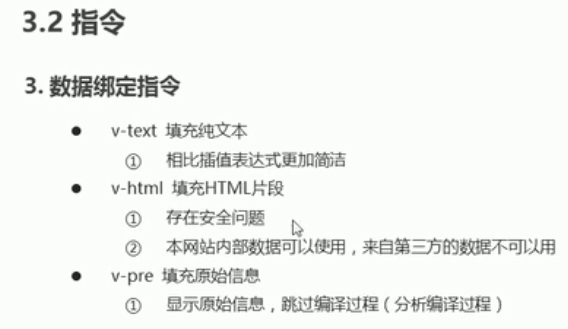
     data:{

        msg:"hello worder"

     }

    })

数据绑定指令



V-text 填充纯文本 没有“闪动”问题

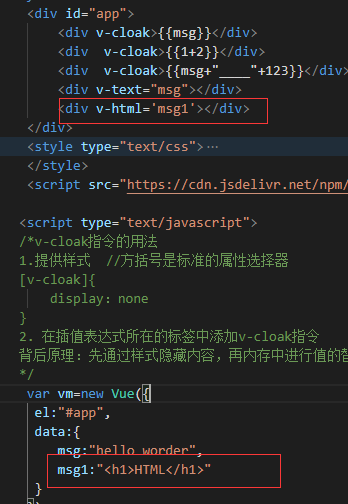
举栗子： <div v-text="msg"></div>

V-hml 填充html片段

**1.存在安全问题**

2. 本网站内部数据可以使用，来自第三方的数据不可以用

栗子：



V-pre 填充元原始信息

1. 显示原始信息，跳过编译过程（分析编译过程）

栗子：



显示结果：带上了花括号

Vue代码（网址 ）：

"https://cdn.jsdelivr.net/npm/vue/dist/vue.js"

Vue模板语法-数据响应式

1. Html5中的响应式（屏幕尺寸的变化导致样式的变化）屏幕适配
2. 数据的响应式（数据的变化导致页面内容的变化）
3. 数据绑定就是：将数据填充到标签中
4. V-once 只编译一次（显示内容后不再具有响应式功能） 应用场景：如果显示的信息后续不需要再修改，我们就可以使用v-once，这样可以提高性能

V-one案例：

 <style type="text/css">

[v-cloak]{

  display: none;

}

  </style>

</head>

<body>

<div id="app">

<div v-cloak>{{msg}}</div>

<div v-text="msg1"></div>

<div v-html="msg3"></div>

<div v-pre>{{我是原型代码}}</div>

 <div v-once>{{info}}</div> <!--这里使用v-one不再支持二次数据响应 -->

</div>

  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

  <script>

    var vm=new Vue({

      el:"#app",

      data:{

        msg:"hello vue",

        msg1:"你好，我是v-text",

        msg3:"<h1>你好我是html</h1>",

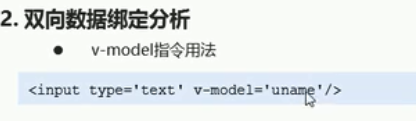
        info:"你好"

      }

    })

Vue模板语法-双向数据绑定

V-model



 <div id="app">

  <input type="text" v-model="msg" >{{msg}}

</div>

  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

  <script type="text/javascript">

    var vm=new Vue({

      el:"#app",

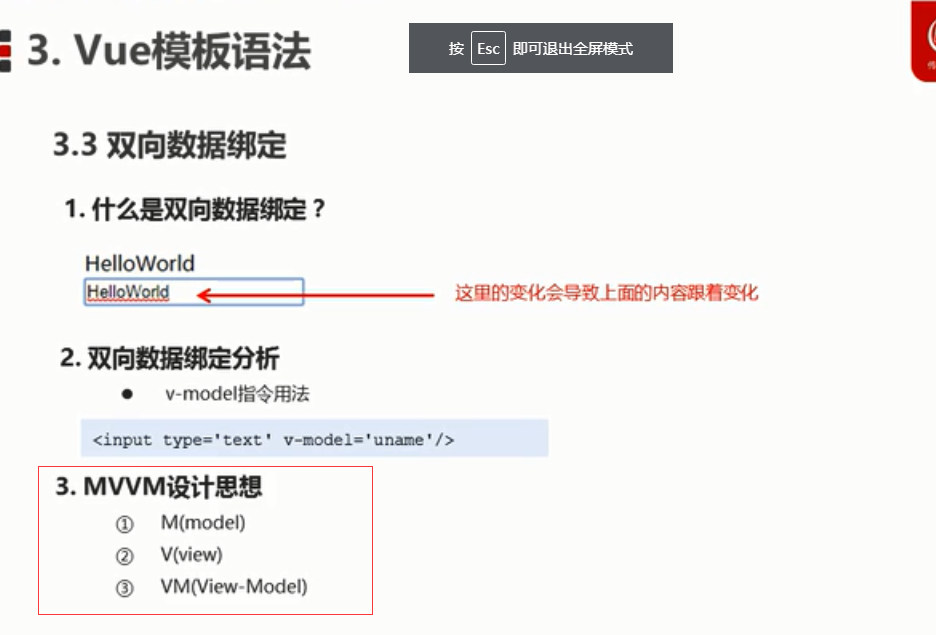
      data:{

        msg:'双向数据绑定'

      }

    })

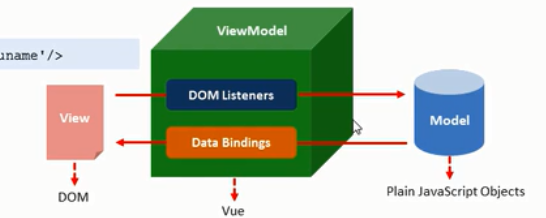
Mvvm设计思想



M：就是vue中的data数据

V：视图 本质上是dom元素

Vm：控制逻辑，结合m和v



3.4事件绑定（用户交互）



V-on指令：

不推荐

<div id="app">

        <div>{{msg}}</div>

        <button v-on:click='msg++'>点击相加</button>  <!--指令用法 -->

        <button @click='msg++'>点击相加</button>  <!--指令用法 -->

    </div>

      <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

      <script type="text/javascript">

      var vm=new Vue({

          el:"#app",

          data:{

              msg:0

          }

      })

Methods属性用来专门定义需要的方法

Methods：{

Handle：function（）{

语句块

}



注意事项：函数必须要做在methods

<body>

    <div id="app">

        <div>{{msg}}</div>

        <button v-on:click='msg++'>点击相加1</button>  <!--指令用法，不能在这里执行逻辑相加 -->

        <button @click='msg++'>点击相加2</button>  <!--简写用法,不能在这里执行逻辑相加 -->

        <button @click='msg++'>点击相加3</button>  <!--单独写methods+this调用 -->

        <button @click='xiangja()'>点击相加4</button>  <!--使用方法名+（）调用方法 -->

    </div>

      <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

      <script type="text/javascript">

      var vm=new Vue({

          el:"#app",

          data:{

              msg:0

          },

          methods:{

              xiangja:function(){

                  //这里的this是vue的实例对象msg

                  console.log(this===vm)//结果是true

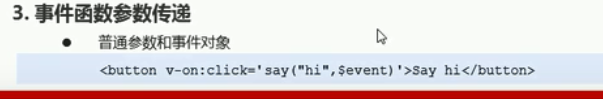
                  this.msg++

              }

          }

      })

事件函数参数传递



$sevent固定写法，传递事件对象（必须要放到参数的最后一个，并且名字是固定的）

tagName：获取标签的名称

10-11

    <div id="app">

        <div>{{msg}}</div>

        <button v-on:click='jj1'>点击相加</button>

        <button v-on:click='jj2(123,456,$event)'> 点击显示传参</button>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

       var vm=new Vue({

             /\*事件绑定-参数传递

             1.如果事件直接绑定函数名称，那么默认会传递事件对象作为事件函数的第一个参数

             2.如果事件绑定函数调用。那么事件对象必须作为最后一个参数显示传递，并且对象的名称必须是$event\*/

           el:"#app",

           data:{

                msg:0

           },

        methods:{

            jj2:function(p,p1,sevent) {

                console.log(p,p1)

                console.log(sevent.target.innerHTML)//tarNanme

                this.msg++

            },

             jj1:function(event){

                 console.log(event.target.innerHTML)

             }

            }

       })

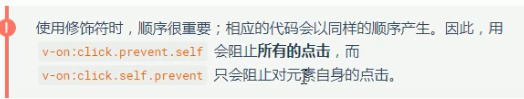
10-12事件修饰符

.stop阻止冒泡

.prevent 阻止默认行为

更多修饰符在官网 事件处理-拉下去 有个事件修饰符/或者在 api --v-on里面也有修饰符

注意事项：如果串联的话要注意先后顺序





键盘修饰符



.enter 回车键

.delete 删除键==

  <div id="app">

      <form action="">

       登录：<input type="text" v-on:keyup.delete="kon" v-model="uname"  > <br>

       密码：<input type="Password"  v-on:keyup.enter="click1" v-model="pwd" > <br>

             <input type="button"  v-on:click="click1" value="提交">

      </form>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

    /\*事件绑定-按键修饰符\*/

          var vm=new Vue({

              el:"#app",

              data:{

                uname:"",

                pwd:""

              },

              methods:{

                click1:function(){

                   console.log(this.uname,this.pwd)

                  },

                  kon:function(){

                      //按delete键清空用户名

                    this.uname=""

                  }

                }

          })

在事件修饰符下面 主要用于事件

Model在input相当于value可以显示用户输入的值

自定义按键修饰符

config.keyCodes



案例： <title>自定义修饰符</title>

</head>

<body>

    <div id="app">

  <input type="text" v-on:keyup.aaa="ss" v-model="kecod">

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script  type="text/javascript">

    /\*事件绑定-自定义按键修饰符\*/

    //规则：自定义按键修饰符：名字是自定义的对应的值必须是对应event.keyCode值

    Vue.CONFIG.keyCode.aaa=65

   var  vm=new Vue({

         el:"#app",

         data:{

               kecod:""

         },

         methods:{

             ss:function(event){

                 console.log(event.keyCode)

             }

         }

   })

案例：简单计算器



代码： <title>计算机</title>

</head>

<body>

    <div id="app">

 <h1>简单计算器</h1><br>

数值A:<input type="text" v-model="a"><br>

数值B:<input type="text" v-model="b"><br>

      <button v-on:click="jisuan">计算</button><br>

      <span>计算结果:{{all}}</span>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script  type="text/javascript">

   var vm=new Vue({

       el:"#app",

       data:{

        a:"",

        b:"",

        all:""

       },

       methods: {

           jisuan:function(){

               this.all=parseInt(this.a)+parseInt(this.b)

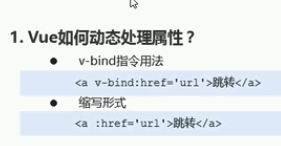
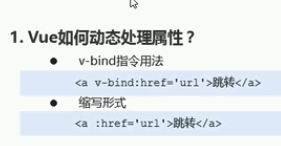
           }

       },

   })

Vue模板语法 之属性绑定

V-bind指令：动态修改属性值



案例：

 <title>动态修改属性</title>

</head>

<body>

    <div id="app">

        <a :href="url">百度一下</a>

        <button v-on:click="qie">切换</button>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

     var vm=new Vue({

        el:"#app",

        data:{

            url:"https://www.baidu.com/"

        },

        methods:{

            qie:function(){

               this.url="https://www.bilibili.com/"

            }

        }

     })

V-model的底层实现原理分析：

手工方式实现双向绑定



或者：



三种方式实现双向绑定

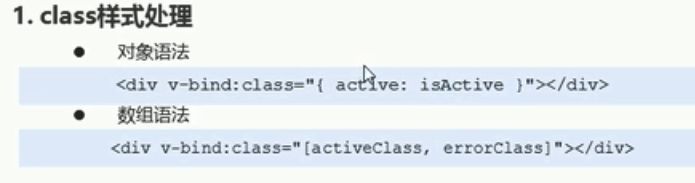


手工双向绑定太麻烦：只是作为了解

19.Vue模板语法 之 样式绑定

is开头的属性名：他的值，不是true就是false





关于class对象样式案例：

 <title>Vue之动态绑定样式</title>

    <style type="text/css">

    .yang{

        border: 1px solid red;

        height: 150px;

        width: 150px;

    }

    .error{

        background-color: orange;

    }

    </style>

</head>

<body>

    <div id="app">

        <div v-bind:class="{yang:isYang,error:isError}">样式测试</div>

        <button v-on:click="ceshi">切换</button>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

        /\*事件绑定-按键修饰符\*/

        var vm = new Vue({

            el: "#app",

            data: {

                isYang:true,

                isError:true

            },

            methods: {

                //控制ceshi的值在true和false之间

              ceshi:function(){

                    this.isYang= !this.isYang

                    this.isError= !this.isError

              }

            }

        })

Class样式数组语法：

 <title>动态绑定样式之数组</title>

    <style type="text/css">

        .yang{

            border: 1px solid red;

            height: 150px;

            width: 150px;

        }

        .error{

            background-color: orange;

        }

        </style>

</head>

<body>

    <div id="app">

        <div v-bind:class="[yangclass,errorclass]">样式测试</div>

        <button v-on:click="shuzu">切换</button>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

        var vm = new Vue({

            el: "#app",

            data: {

              yangclass:"yang",

              errorclass:"error"

            },

            methods: {

                //控制样式绑定数组

              shuzu:function(){

                  this.yangclass=""

                  this.errorclass=""

              }

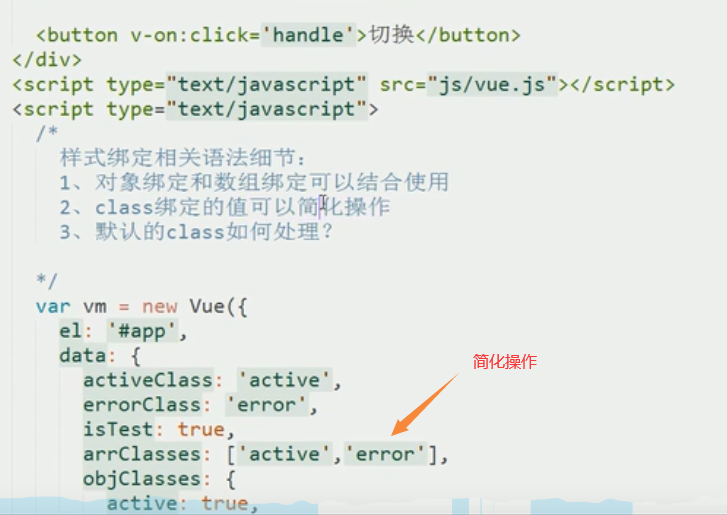
            }

        })

样式绑定相关语法细节

1.对象绑定和数组绑定可以结合使用

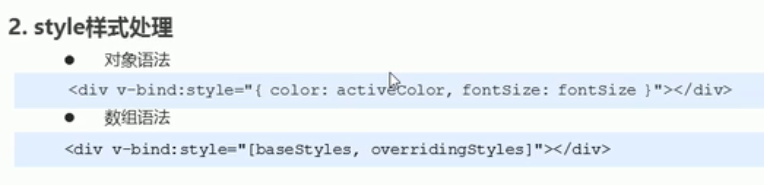
1. class绑定的值可以简化操作
2. 默认的class如何处理 默认的class会保留，可以再通过对象或数组进行动态添加







Vue模板语法之style样式处理  
对象语法，数组语法



Style绑定之 对象绑定方式和数组绑定方式：

 <title>style绑定</title>

 <title>style绑定</title>

</head>

<body>

  <div id="app">                  <!--  style样式处理之对象案例 -->

            <div v-bind:style="{border:borderStyle,width:widthStyle, height:heightStyle}"></div>

            <button v-on:click="hen">切换</button>

            <div v-bind:style="objStyle">第二个div</div>

                                  <!-- styley样式 之数组对象案例： -->

            <div v-bind:style="[objStyle,overrideStyles]">第三个div</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

        var vm=new Vue({

            el:"#app",

            data:{

                borderStyle:"1px solid blue",

                widthStyle:"100px",

                heightStyle:"200px",

                objStyle:{

                    border:"1px solid green",

                    height:"100px",

                    width:"100px"

                    /\*对象简写方式\*/

                },

                overrideStyles: {

                    border:"5px solid orange",

                    backgroundColor:"blue"

                }

            },

            methods:{

                hen:function(){

                    this.heightStyle="100px"

                    this.objStyle.width="200px"

                }

            }

        })

    </script>

1. 分支循环结构
2. V-if
3. V-else
4. V-else-if
5. V-show



V-if v-else-if （概念：控制元素是否渲染到页面，适用于不经常改动显示元素）

 <title>vue之ifelse</title>

</head>

<body>

    <div id="app">

        <div v-if="score>=99">优秀 </div>

        <div v-else-if='score<90 &&score>=80'>良好 </div>

      <div v-else-if='score<80 &&score>=60'>一般 </div>

        <div v-else>比较差 </div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

      var vm=new Vue({

          el:"#app",

          data:{

            score:35

          } ,

          methods: {

          },

      })

渲染出来console.log界面只会出现匹配的单个值

V-show案例（概念：控制元素是否显示通过display属性（已静渲染到页面，只不过是通过display属性是否显示）适用于：频繁显示或隐藏）：

V-show值是：true或false

案例：

 <title>vue之ifelse</title>

</head>

<body>

    <div id="app">

        <div v-if="score>=99">优秀 </div>

        <div v-else-if='score<90 &&score>=80'>良好 </div>

      <div v-else-if='score<80 &&score>=60'>一般 </div>

        <div v-else>比较差 </div>

        <div v-show="flag">测试 v-show</div>

        <button v-on:click="hen">切换</button>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

    /\*分支结构

    v-show的原理：控制元素样式是否显示 display：none\*/

      var vm=new Vue({

          el:"#app",

          data:{

            score:35,

            flag:false

          } ,

          methods: {

              hen:function(){

                  this.flag=!this.flag

              }

          },

      })

Vue模板语法之循环结构：

V-for遍历数组

Key的作用：帮助区分不同的元素，从而提高性能



V-for遍历案例 和 V-for增加索引（下标）案例：

 <title>vue之ifelse</title>

</head>

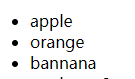
<body>

    <div id="app">

        <div>水果列表</div>

        <ul>  <!--这里的item是自定义的，代表的是遍历数组得到的某一个数组，in是关键字（不可少） fruits是数组名 -->

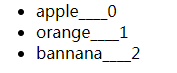
                <!-- {{intem}}这里是将v-for遍历到的结果放进插入表达式 -->

            <li v-for='item in  fruits'>{{item}}</li> 显示效果：

          <!-- 增加访问：数据下标（索引） -->

       <!-- 这里的intem(得到遍历数组的数据)和 index（得到遍历数组的下标）名都是自定义名称 -->

        <li v-for='(item,index) in  fruits'>{{item+'\_\_\_\_'+index}}</li>

显示效果：

        </ul>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

      var vm=new Vue({

          el:"#app",

          data:{

              /\*数组\*/

            fruits:['apple','orange','bannana']

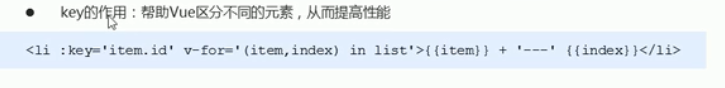
          } ,

          methods: {

          },

      })

Key的作用：帮助vue不同元素，从而提高性能



案例：区分兄弟节点，给每一个兄弟节点标注一个唯一的标识，一般使用id

遍历时都加上key

复杂案例2：key案例

  <title>vue之ifelse</title>

</head>

<body>

    <div id="app">

        <div>水果列表</div>

        <ul>  <!--这里的item是自定义的，代表的是遍历数组得到的某一个数组，in是关键字（不可少） fruits是数组名 -->

                <!-- {{intem}}这里是将v-for遍历到的结果放进插入表达式 -->

            <li v-for='item in  fruits'>{{item}}</li>

          <!-- 增加访问：数据下标（索引） -->

       <!-- 这里的intem(得到遍历数组的数据)和 index（得到遍历数组的下标）名都是自定义名称 -->

          <li v-for='(item,index) in  fruits'>{{item+'\_\_\_\_'+index}}</li>

  <!-- 加上key(区分兄弟节点，给每一个兄弟节点标注一个唯一的标识,从而提高性能)标识： -->

          <li :key='index' v-for='(item,index) in myfruits'>

              <span>{{item.ename}}</span>

              <span>-----------</span>

              <span>{{item.cname}}</span>

          </li>0

        </ul>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

      var vm=new Vue({

          el:"#app",

          data:{

              /\*数组\*/

            fruits:['apple','orange','bannana'],

            myfruits:[{

                ename:'apple',

                cname:'苹果',

              },

              {

                ename:'orange',

                cname:'橘子',

              },

              {

                  ename:'bannana',

                  cname:'香蕉'

              }

            ]

          } ,

          methods: {

          },

      })

1. V-for 遍历对象 v-if和v-for结合使用（ ）



Div v-for‘（value（值），key（键盘/关键字），index（下标））in object’> in object’> </div>

案例：

 <title>vue之ifelse</title>

</head>

<body>

    <div id="app">

<div v-if='v==13'   v-for="(v,k,i) in obj">{{v+k+i}}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

       /\* 原生js方式

var obj = {

            uname: 'lisi',

            age: 12,

            gender: 'male'

        }

       /\* for (var key in obj) {

            console.log(key, obj[key])

        }\*/

        var vm = new Vue({

            el:"#app",

            data: {

                obj:{

                    uname:'lisi',

                    age:13,

                    gender:'male'

                }

            },

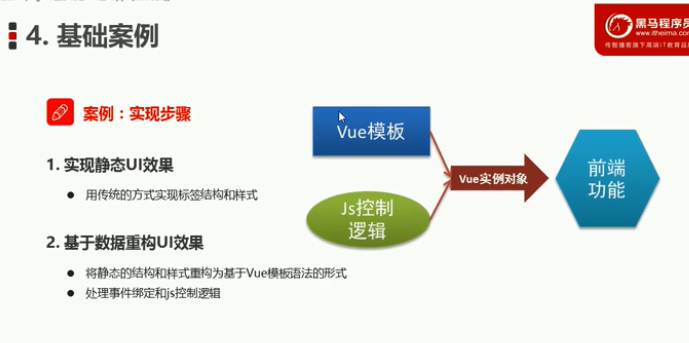
            methods: {

            },

        })

案例：







  <title>vue之ifelse</title>

    <style type="text/css">

        table tr td {

            border:1px solid blue;

            width: 120px;

            height: 60px;

            text-align: center;

        }

        table tr td.xuan{

            background-color: orange;

        }

        ul li{

            list-style: none;

        }

       .div1{

            display: none;

        }

    </style>

</head>

<body>

    <div id="app">

        <table>

            <tr >

                <td v-on:click="click1(index)" :class='xuan==index?"xuan":""'  :key="item.id" v-for='(item,index) in list'>{{item.title}}</td>

            </tr>

        </table>

        <div >

            <div :class='xuan==index?"":"div1"' :key="item.id" v-for='(item,index) in list' >

                <img :src="item.path" >

            </div>

        </div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

      var vm=new Vue({

          el:"#app",

          data:{

              xuan:0,//当前选中选项卡的索引

            list:  [{id:1,

                  title:'apple',

                  path:'img/apple.png'},

                 {

                  id:2,

                  title:'orange',

                  path:'img/orange.png'

                  },

                 {id:3,

                  title:'lemon',

                  path:'img/lemon.png'}],

      },

          methods: {

              click1:function(index){

                  //在这里实现选项卡切换操作：本质就是操作类名

                  //如何操作类名？就是通过索引（xuan）来操作

                  this.xuan=index

              }

          },

      })

P226 01 vue的常用特性（）



1.表单操作

2.自定义指令

3.计算属性

4.过滤器

5.侦听器

6.生命周期

●表单操作：通过表单进行用户数据交互



Input 单行文本

Textarea多行文本

Select 下拉多选

Radio 单选框

Checkbox 多选框

注意：提交表单不要用默认的提交，一定要通过js方式来控制手动提交

  <title>vue之ifelse</title>

</head>

<body>

    <div id="app">

        <form action="https://www.w3school.com.cn/">

            <div>

                <span>姓名：</span>

                <span>

                    <input type="text" v-model="uname">

                </span>

            </div>

            <!-- 单选框 -->

             <div>

                 <span>性别：</span>

                 <span>

                     <input type="radio" id="male" value="1" v-model="gender"></span>

                <label for="male">男</label>

                <span><input type="radio" id="female" value="2" v-model="gender"></span>

                <label for="female">女</label>

                </div>

                <div>

            <!-- 复选框 -->

            <span>爱好：</span>

            <span><input type="checkbox" id="ball" value="1" v-model="hob"></span>

            <label for="ball">篮球</label>

            <span><input type="checkbox" id="sing" value="2" v-model="hob"></span>

            <label for="sing">唱歌</label>

            <span><input type="checkbox" id="prm" value="3 " v-model="hob"></span>

            <label for="prm">敲代码</label>

        </div>

                <div>

            <!-- 下拉列表 -->

            职业：<select>

                <option value="mo">请选择职业</option>

                <option value="ofe">文员</option>

                <option value="prm">程序员</option>

                <option value="sel">推销员</option>

            </select>

        </div>

            <!-- 文本域 -->

            <pre></pre>

            <span>个人简介：</span>

            <textarea name="jian" cols="30" rows="3">请输入简介.....

            </textarea>

            <pre></pre>

            <!--这里的prevent是事件修饰符，阻止了form表单中的默认提交功能 -->

            <input type="submit" value="提交" @click.prevent='handle'>

        </form>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

        var vm = new Vue({

            el: "#app",

            data: {

                uname: "单红艳",

                gender:'1',

                hob:[2,1]

            },

            methods: {

                handle: function () {

                    console.log(this.uname)

                    console.log(this.gender)

                    console.log(this.hob.toString())

                }

            },

        })

    </script>

 <title>vue之ifelse</title>

</head>

<body>

    <div id="app">

        <form action="https://www.w3school.com.cn/">

            <div>

                <span>姓名：</span>

                <span>

                    <input type="text" v-model="uname">

                </span>

            </div>

            <!-- 单选框 -->

             <div>

                 <span>性别：</span>

                 <span>

                     <input type="radio" id="male" value="1" v-model="gender"></span>

                <label for="male">男</label>

                <span><input type="radio" id="female" value="2" v-model="gender"></span>

                <label for="female">女</label>

                </div>

                <div>

            <!-- 复选框 -->

            <span>爱好：</span>

            <span><input type="checkbox" id="ball" value="1" v-model="hob"></span>

            <label for="ball">篮球</label>

            <span><input type="checkbox" id="sing" value="2" v-model="hob"></span>

            <label for="sing">唱歌</label>

            <span><input type="checkbox" id="prm" value="3 " v-model="hob"></span>

            <label for="prm">敲代码</label>

        </div>

                <div>

            <!-- 下拉列表 -->

            职业：<select v-model="work">

                <option value="0">请选择职业</option>

                <option value="1">文员</option>

                <option value="2">程序员</option>

                <option value="3">推销员</option>

            </select>

        </div>

            <!-- 文本域 -->

            <pre></pre>

            <span>个人简介：</span>

            <textarea name="jian" cols="30" rows="3" v-model="text"> <!-- 这里不能写值，要在v-model里面写 -->

            </textarea>

            <pre></pre>

            <!--这里的prevent是事件修饰符，阻止了form表单中的默认提交功能 -->

            <input type="submit" value="提交" @click.prevent='handle'>

        </form>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

        var vm = new Vue({

            el: "#app",

            data: {

                uname: "单红艳",

                gender:'1',

                hob:[2,1],

                work:2,

                text:'你好'

            },

            methods: {

                handle:function () {

                   /\*/ console.log(this.uname)

                    console.log(this.gender)

                    console.log(this.hob.toString())\*/

                    console.log(this.work)//单选方式

                }

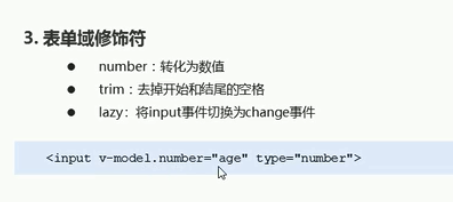
            },

        })

    </script>



Vue 常用特性---表单操作-表单域修饰符



Number：一般默认是字符串，可通过number修饰符转化为数值

Trim：去掉开始和结尾的空格

Lazy：将input事件切换为change事件 1. change是失去焦点的时候触发

2. input事件是实时触发

<title>vue之ifelse</title>

</head>

<body>

    <div id="app">

       <input type="text" v-model.number='age'>

       <input type="text" v-model.lazy="msg">

       <div>{{msg}}</div>

       <button @click="handle"> 点击一下</button>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

      var vm=new Vue({

          el:"#app",

          data:{

                 age:"",

                 msg:''

          } ,

          methods: {

            handle:function(){

                alert(this.age+10) // 结果返回20 通过v-model.number将默认字符改为number类型

                alert(this.msg)

            }

          },

      })

Vue常用特性---自定义指令：

内置指令官网一共有13个，在api哪里



需求：给input获取焦点 （directive英文意思：指令）

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>vue之自定义指令</title>

</head>

<body>

    <div id="app">

        <input type="text" v-focus>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

        /\*

    自定义全局指令

    \*/

        /\*括号内的第一个参数就是指令名是自定义（focus）的，第二个参数对象inserted

         用来实现自定义指令的业务逻辑，是固定的名字\*/

        Vue.directive("focus", {

            inserted: function (el) {

                //el 表示指令所绑定的元素

                el.focus()

            }

        });

        var vm = new Vue({

            el: "#app",

            data: {

            },

            methods: {

            },

        })

    </script>

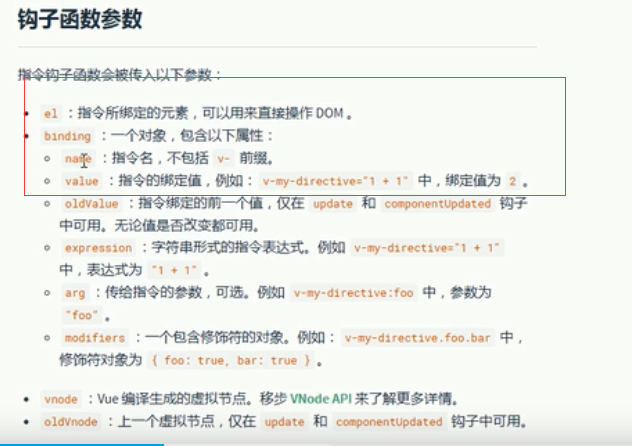
</body>

</html>

Vue常用特性----自定指令携带参数，使用和定义



案例：



<title>vue之ifelse</title>

</head>

<body>

    <div id="app">

       <input type="text" v-color="msg">

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

    Vue.directive('color',{

        bind:function(el,binding){

            //根据指令的参数设置背景色

           // console.log(binding.value.color)

           el.style.backgroundColor = binding.value.color

        }

    })

      var vm=new Vue({

          el:"#app",

          data:{

              msg:{

                  color:'blue'

              }

          } ,

          methods: {

          },

      })

常用特性 之 指令之局部指令



  <title>vue之自定义指令携带参数的定义和使用</title>

</head>

<body>

    <div id="app">

       <input type="text" v-color='msg'>

       <input type="text" v-focus>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

    /\*

    自定义指令----带参数

     Vue.directive('color',{

        bind:function(el,binding){

            //根据指令的参数设置背景色

           // console.log(binding.value.color)

           el.style.backgroundColor = binding.value.color

        }

    })\*/

      var vm=new Vue({

          el:"#app",

          data:{

            msg:{

                color:'red'

            }

          } ,

          methods: {

          },

          directives:{

              //局部指令

            color:{

                bind:function(el,binding){

            //根据指令的参数设置背景色

           // console.log(binding.value.color)

           el.style.backgroundColor = binding.value.color

        }

            },

            //多个指令

            focus:{

                inserted:function(el){

                    el.focus();

                }

            }

          }

    })

常用特性---局部指令只能限制在对应的组件内使用！

Vue常用特性之 ------计算属性（可以使得模板内容更加简洁）

案例：



Computed计算方法：{

}

案例：   <title>vue计算属性</title>

</head>

<body>

    <div id="app">

       <div>{{msg}}</div>

因为是计算属性调时reveseSting 方法不用带（括号）

       <div>{{reveseSting}}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

      var vm=new Vue({

          el:"#app",

          data:{

            msg:'你好'

          } ,

          methods: {

          },

          computed:{

              reveseSting:function(){

                  //定义计算方法

                  //计算属性基于data内的内容去改变

                  //反转字符串 必须使用return 将结果返回

 //split（）分割字符串的方法，reverse（）颠倒字符串的方法，join（）

                return this.msg.split('').reverse().join('');

              }

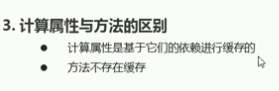
          }

      })

    </script>

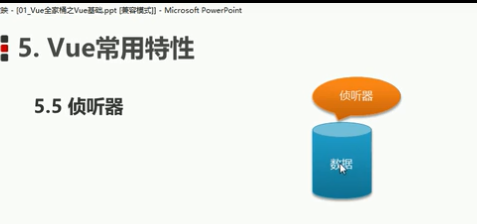
计算属性与方法的区别

1. 计算属性是基于他们的依赖进行缓存（数据依赖data数据）
2. 方法不存在缓存



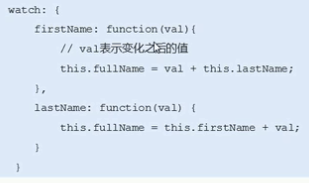


Vue常用特性 --侦听器





语法规则：



  <title>vue之ifelse</title>

</head>

<body>

    <div id="app">

       <span>姓：</span>

       <input type="text" v-model="firstName"><br>

       <span>名：</span>

       <input type="text" v-model="laseName">

       <div>{{allname}}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

      var vm=new Vue({

          el:"#app",

          data:{

            firstName:"",

            laseName:"",

            allname:""

          } ,

          methods:{

          },

          //侦听器方法

          watch:{

这里的firstName和laseName必须和需要data内的命名一致，才能被监听

            firstName:function(val){

                this.firstName=val+""+this.laseName

            },

            laseName:function(val){

                this.allname=this.firstName+""+val

            }

          }

        /\*计算属性 实现效果的一种方式

         computed:{

              allname:function(){

                  return this.first+this.name

                      }

          }\*/

      })

Vue常用特性之-----侦听器异步用法：





 <title>异步</title>

</head>

<body>

    <div id="app">

        <span>用户名：</span>

      <span>

          <input type="text" v-model.lazy="uname">

      </span>

       <div>{{tip}}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

    /\*侦听器

              1. 采用侦听器监听用户名的变化

              2.调用后台接口进行验证

              3.根据验证结果调试结果提示信息\*/

      var vm=new Vue({

          el:"#app",

          data:{

                uname:"",

                tip:""

          } ,

          methods: {

              checkName:function(uname){

                  var that = this;

                  //调用后台接口，但是可以使用定时任务的方式模拟接口调用

                  setTimeout(function(){

                      //模拟接口调用

                    if(uname=='admin'){

                         that.tip='用户已经存在，请更换一个'

                    }else{

                        that.tip = '用户名可以使用'

                    }

                } ,2000)

              }

          },

          watch:{

              uname:function(val){

                  //调用后台接口验证用户的合法性

                this.checkName(val);

                //修改提示信息

                this.tip='正在验证'

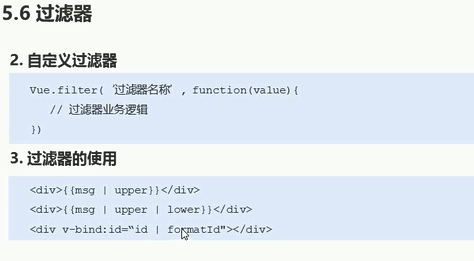
              }

          }

      })

Vue常用特性之 过滤器







案例：

 <title>过滤器</title>

</head>

<body>

    <div id="app">

        <input type="text" v-model="msg">

        <!-- 过滤器引用 -->

        <div>{{msg | upper}}</div>

        <!-- 过滤器连级引用 -->

        <div>{{msg | upper | lower}}</div>

             <!-- 这里使用upper过滤器 -->

        <div :abc='msg|upper '>测试数据</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

    /\*过滤器  全局过滤器

             这里的upper是过滤器名称自定义，这里的val是表示要过滤的数据                \*/

   /\* Vue.filter('upper',function(val){

         //charAt（）方法可以返回指定位置的字符

         //toUpperCase（）方法用于把字符串转为大写

         //slice（）方法可从已有的数组中返回选定的元素

      return  val.charAt(0).toUpperCase() +val.slice(1)

    })

    Vue.filter('lower',function(val){

         //charAt（）方法可以返回指定位置的字符

         //toLowerCase()可以将字符串转为小写

         //slice（）方法可从已有的数组中返回选定的元素

      return  val.charAt(0).toLowerCase() +val.slice(1)

    })\*/

      var vm=new Vue({

          el:"#app",

          data:{

               msg:""

          } ,

          methods: {

          },

          //局部过滤器 只能在本组件中使用

        filters:{

          upper:function(val){

      return  val.charAt(0).toUpperCase() +val.slice(1)

          }

        }

      })

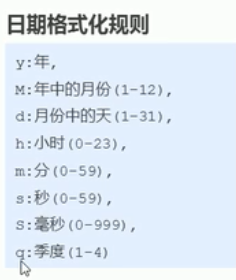
Vue常用特性之 带参数的过滤器



备注，参数是从第二个参数开始执行

需求：





 <title>参数过滤器</title>

</head>

<body>

    <div id="app">

        <div>{{date | format('yyyy-mm-dd')}}</div>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

    /\*过滤器案例：格式化日期\*/

    Vue.filter('format',function(value,arg){

console.log(arg)

if(arg=='yyyy-mm-dd'){

    var ret="";

    //                       get.Month()方法是0-11

    ret+= value.getFullYear()+"-"+(value.getMonth()+1 )+'-'+value.getDate();

    return ret

}

            return value;

    })

      var vm=new Vue({

          el:"#app",

          data:{

            date: new Date()

          } ,

          methods: {

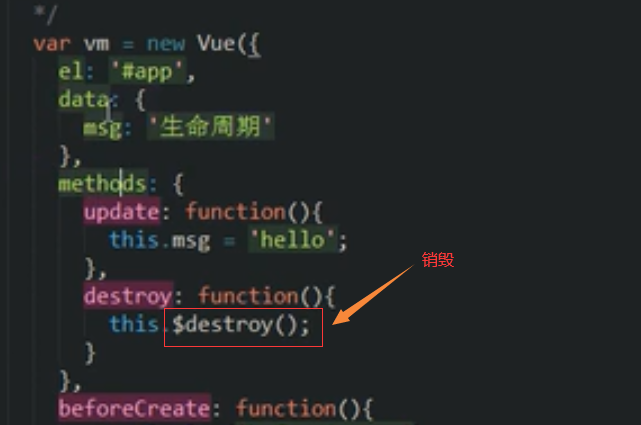
          },

      })

Vue常用特性--生命周期：



mounted

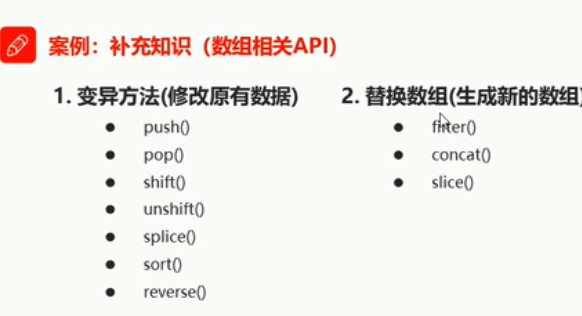


Mounted 可以调用后台接口，

Vue常用特性 综合案例

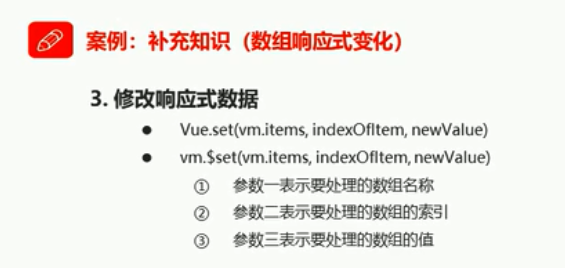


变异方法

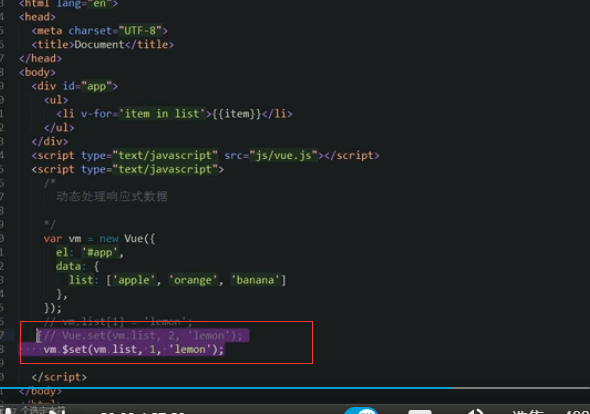


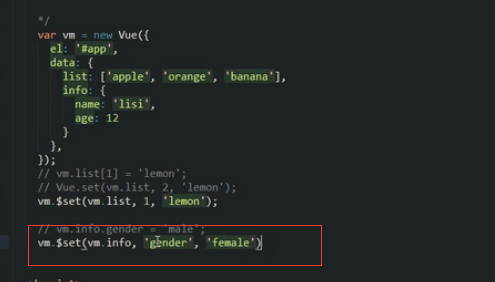






数组和对象响应式处理





图书馆







业务逻辑：先获取id信息，然后填充到data的数据内

Filter() 方法

filter() 方法创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素。

**注意：** filter() 不会对空数组进行检测。

**注意：** filter() 不会改变原始数组。

返回数组 ages 中所有元素都大于 18 的元素:

var ages = [32, 33, 16, 40];  
function checkAdult(age) {  
    return age >= 18;  
}  
function myFunction() {  
    document.getElementById("demo").innerHTML = ages.filter(checkAdult);  
}

输出结果为:

32,33,40

## 定义和用法

findIndex() 方法返回传入一个测试条件（函数）符合条件的数组第一个元素位置。

findIndex() 方法为数组中的每个元素都调用一次函数执行：

* 当数组中的元素在测试条件时返回 true 时, findIndex() 返回符合条件的元素的索引位置，之后的值不会再调用执行函数。
* 如果没有符合条件的元素返回 -1

**注意:** findIndex() 对于空数组，函数是不会执行的。

**注意:** findIndex() 并没有改变数组的原始值。

## 定义和用法

findIndex() 方法返回传入一个测试条件（函数）符合条件的数组第一个元素位置。

findIndex() 方法为数组中的每个元素都调用一次函数执行：

* 当数组中的元素在测试条件时返回 true 时, findIndex() 返回符合条件的元素的索引位置，之后的值不会再调用执行函数。
* 如果没有符合条件的元素返回 -1

**注意:** findIndex() 对于空数组，函数是不会执行的。

**注意:** findIndex() 并没有改变数组的原始值。

图书馆案例：

 methods: {

                handle: function () {

                    //修改原始数据

                    if (this.flag) {

                        //编辑操作

                        //就是根据当前的id去更新数组中对应的数据

                        this.books.some((bbb) => {

                            if (bbb.id == this.id) {

                                bbb.name = this.name

                                //完成更新操之后，需要终止循环

                                return true

                                s

                            }

                        });

                        this.flag = false;

                    } else {

                        //添加图书

                        var book = {}

                        book.id = this.id

                        book.name = this.name

                        book.date = ""

                        this.books.push(book)

                        //清空表单

                        this.id = ""

                        this.name = ""

                    }

                    this.id = ""

                    this.name = ""

                },

                toEdit: function (id) {

                    //禁止修改id

                    this.flag = true

                    // console.log(id)

                    //根据id查询出要编辑的数据 filter方法英文是过滤器的意思，它会返回一个新的符合条件的数组

                    var book = this.books.filter(function (item) {

                        return item.id == id

                    })

                    console.log(book)

                    //把获取到的信息填充到表单

                    this.id = book[0].id

                    this.name = book[0].name

                },

                deletBook: function (id) {

                    //删除图书

                    //根据id查找元素的索引

                    var index = this.books.findIndex(function (aaa) {

                        return aaa.id == id

                    })

                    //根据索引删除数据元素

                    // splice() 方法向/从数组中添加 /删除项目，然后返回被删除的项目。

                    this.books.splice(index, 1)

                }

            },

        })