1. vuejs开发环境搭建
2. 直接CND导入

可以选择开发版本还是生产版本。

|  |
| --- |
| <!—开发环境版本 -->  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  <!—生产环境版本 -->  <script src="https://cdn.jsdelivr.net/npm/vue"></script> |

1. 下载和引入



1. NPM安装

后续通过webpack和CLI的使用，以后都使用该方式。

1. Vue初体验

现在要显示如何展示列表数据，在html代码中使用v-for语法来遍历数据，整体输出。更加重要的时vue的代码是响应式的，也就是数据发生改变时，页面显示的数据也会实时改变。如下：

|  |
| --- |
| <div id="id1">  <ul>  <li v-for="item in movies">{{item}}</li>  </ul> </div> <script src="../js/vue.js"></script> <script>  const ***app*** = new Vue({  el: "#id1", // 挂在要声明的元素  data: { // 定义数据  message: "你好呀",  movies: ['星际创越', '大话西游', '笔仙','三国演义']  }  }) </script> |

1. 实现简单的计数器

现在来实现一个简单的计数器，点击加号数据加1，点击减号数据减一。这是使用新的属性methods，该属性用于在vue对象钟定一方法。@click用于监听单击事件。

|  |
| --- |
| <div id="id1">  <h2>当前计数:{{message}}</h2> <!-- <button v-on:click="message++">+</button>--> <!-- <button v-on:click="message&#45;&#45;">-</button>-->  <button v-on:click="add">+</button>  <button v-on:click="sub">-</button> </div> <script src="../js/vue.js"></script> <script>  const ***app*** = new Vue({  el: "#id1", // 挂在要声明的元素  data: { // 定义数据  message: 0  },  methods: {  add: function () {  this.message++  ***console***.log("成功加1");  },   sub: function () {  if (this.message > 0) {  this.message--  }else{  this.message = 0;  }  ***console***.log("成功加1");  }  }  }) </script> |

# 二. vue的基础语法

## 1. Mustache语法

|  |
| --- |
| <div id="id1">  <h2>{{message}}</h2>  <h2>{{firstname}},张夏琳</h2>  <!-- Mustache不仅可以写变量，还可以写表达式 -->  <h2>{{firstname + lastname}}</h2>  <h2>{{firstname}} {{lastname}}</h2>  <h2>counter:{{counter \* 2}}</h2> </div> <script src="../js/vue.js"></script> <script>  // let-变量 const-常亮  // 编程范式：声明式编程  const ***app*** = new Vue({  el: "#id1", // 挂在要声明的元素  data: { // 定义数据  message: "我是王雷，hello vuejs",  firstname: "王雷",  lastname: "wangiz",  counter: 33  }  }) </script> |

## 2. 插值操作-其他指令

* 1. v-once

在某些情况下，我们不希望页面的数据随意的发生改变，这个时候我们就可以使用v-once指令。该指令后面不需要跟任何的表达式，该指令表示元素和组件只会渲染一次。

|  |
| --- |
| <div id="id1">  <h2>{{message}}</h2>  <!-- v-once表示该标签只会渲染一次 -->  <h2 v-once>{{name}},张夏琳</h2> </div>  <script src="../js/vue.js"></script> <script>  const ***app*** = new Vue({  el: "#id1",  data: {  message: "我是豌豆荚",  name: "wl"  }  }) </script> |

* 1. v-html

在某些情况下，我们从服务器请求的数据就是HTML的代码，如果我们直接使用{{ }}来输出，会将HTML代码一块输出，但是我们可以按照html格式进行解析，并且显示对应的内容。

这个时候我们就可以使用v-html指令。该指令后面往往会跟一个string类型，然后将HTML解析出来进行渲染。

|  |
| --- |
| <div id="id1">  <h2>{{url}}</h2>  <h2 v-html="url"></h2> </div>  <script src="../js/vue.js"></script> <script>  const ***app*** = new Vue({  el: "#id1",  data: {  url: "<a href='https://www.baidu.com'>百度一下</a>"  }  }) </script> |

* 1. v-text

该指令和Mustache的语法类似，都是将数据显示在界面中。该指令后面往往会跟一个string类型。

|  |
| --- |
| <div id="id1">  <h2>{{message}}</h2>  <h2 v-html="message"></h2> </div>  <script src="../js/vue.js"></script> <script>  const ***app*** = new Vue({  el: "#id1",  data: {  message: "你好啊，王雷"  }  }) </script> |

* 1. v-pre

该指令用于跳过该元素和子元素的编译过程，用于显示原本的Mustache语法。

|  |
| --- |
| <div id="id1">  <h2>{{message}}</h2>  <h2 v-pre>{{message}}</h2> </div> <script src="../js/vue.js"></script> <script>  const ***app*** = new Vue({  el: "#id1",  data: {  message: "ww ww"  }  }) </script> |

* 1. v-cloak

有些情况下，浏览器可能会显示未编译的Mustache标签，v-cloak指令和CSS 规则一起用的时候，能够解决差值表达式闪烁的问题（即：可以隐藏未编译的标签直到实例准备完毕）。。

|  |
| --- |
| <div id="id1">  <h2 v-cloak>hello {{message}}</h2> </div> <script src="../js/vue.js"></script> <script>  setTimeout(function () {  const app =new Vue({  el: "#id1",  data: {  message: "wwwwww"  }  })  },1000) </script> |

## 3. 属性绑定

* 1. v-bind介绍

前面学习的内容主要是将内容添加到模板的内容的当中，但是在开发的过程当中，除了内容要动态决定外，标签属性的内容也需要动态的来决定。

这个时候，就可以使用v-bind指令：

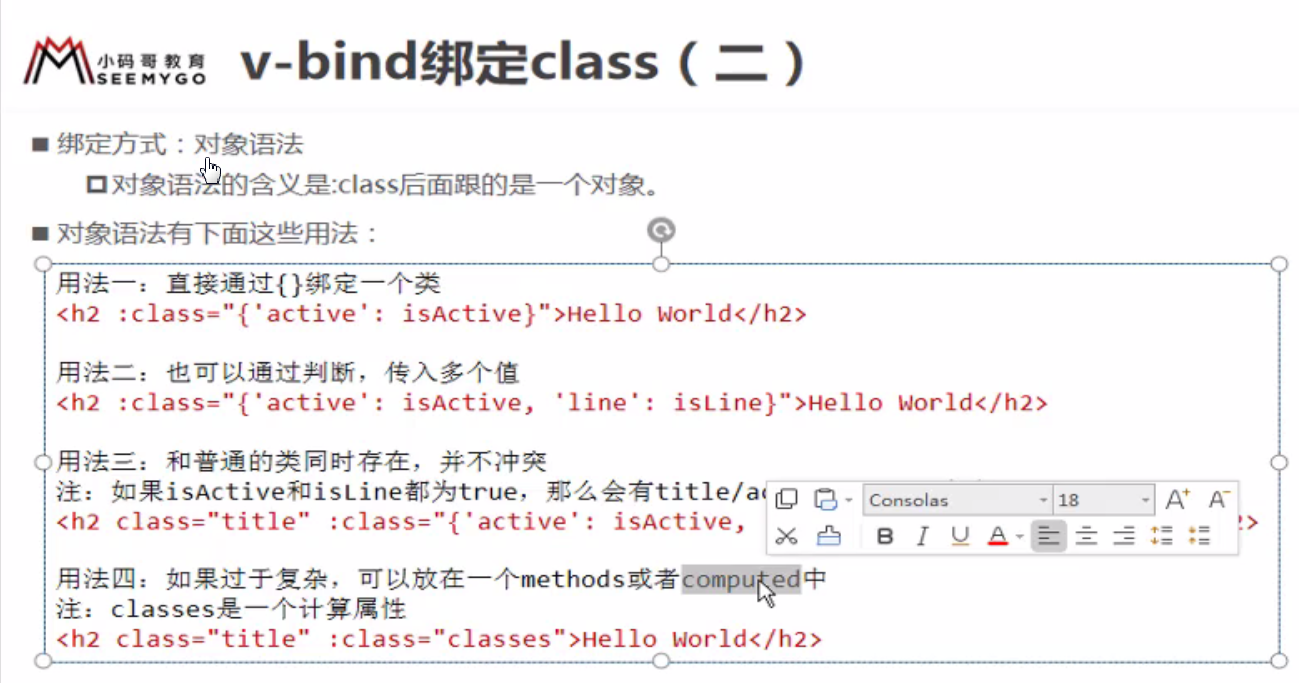
* 作用：动态的绑定属性。
* 缩写： : 语法糖式的写法，本质是v-bind。
* 预期：any object
* 参数：attrOrProp(属性)

v-bind用于绑定一个或者多个属性值，或者向另外一个组件传递props值（组价相关的知识），在开发中一般有哪些属性需要动态的来决定呢？如：图片链接src、网址链接href、动态的绑定一些类和样式等、如下所示的代码就是通过vue中的data绑定元素的src和href。

|  |
| --- |
| <div id="id1">  <a v-bind:href="tesurl">  <img v-bind:src="message" alt="">  </a>  <!-- 语法糖式的写法 -->  <a :href="tesurl">  <img :src="message" alt="">  </a> </div> <script src="../js/vue.js"></script> <script>  const ***app*** =new Vue({  el: "#id1",  data: {  message: "https://f10.baidu.com/it/u=3142007949,1827689403&fm=173&app=49&f=JPEG?w=640&h=480&s=CCC27223C00A36EE1C18B8840100E091&access=215967316",  tesurl: "https://www.baidu.com"  }  }) </script> |

* 1. v-bind动态绑定class-对象语法

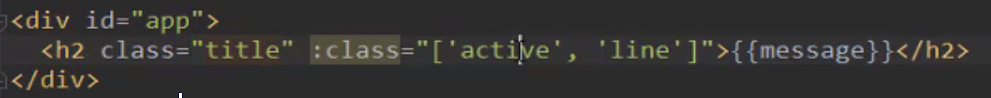
对象语法的含义：class后面跟的是一个对象。



|  |
| --- |
| <head>  <meta charset="UTF-8">  <title>对象语法</title>  <style>  .active {  color: red  }  </style> </head> <body> <div id="id1">  <h2 class="title" :class="{active: isActive, line: isLine}">{{message}}</h2>  <button v-on:click="btnclick">按钮</button> </div> <script src="../js/vue.js"></script> <script>  const ***app*** =new Vue({  el: "#id1",  data: {  message: "王雷来了",  isActive: true,  isLine: true  },  methods: {  btnclick: function () {  this.isActive = !this.isActive  }  }  }) </script> |

* 1. v-bind动态绑定class-数组语法

数组语法的含义：class后面跟的是一个数组。



* 1. v-bind和v-for结合使用

功能：点击那个那个就变红。

|  |
| --- |
| <head>  <meta charset="UTF-8">  <title>数组语法</title>  <style>  .active {  color: red  }  </style> </head> <body> <div id="id1">  <ul>  <li v-on:click="btnlick(index)" :key="index" v-bind:class="{active: index == isActive}" v-for="(movie,index) in movies">{{movie}}</li>  </ul> </div> <script src="../js/vue.js"></script> <script>  const ***app*** =new Vue({  el: "#id1",  data: {  movies: ['大鱼海棠','火影忍者', '中华小当家', '海尔兄弟'],  isActive: false  },  methods: {  btnlick: function (index) {  if (index) {  this.isActive = index  }  }  }  }) </script> |

## 4. 计算属性

4.1 简介

我们知道，在模板中我们可以通过一些差值语法显示data数据。但是有的时候。我们可能要对乙烯数据进行转化后才进行显示、或者将多个数据结合起来进行显示。例如我们有firstname和lastname这两个变量，而我们需要显示完整的变量可以这样写{{firstname lastname}}，但是有多个地方的话，就需要写多次。

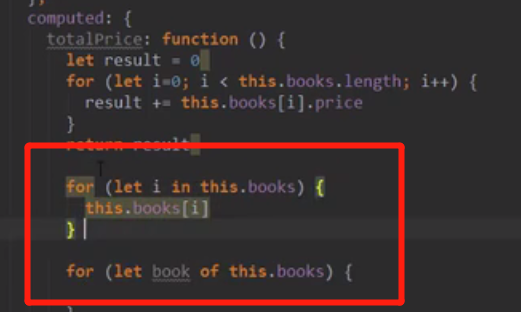
我们可以将上面的功能换成计算属性，放在computed中。。

（1）计算属性的简单操作

|  |
| --- |
| <**div id="id1"**>  <**h2**>{{firstname + " " + lastname}}</**h2**>  <**h2**>{{getfullname()}}</**h2**>  <**h2**>{{fullname + "调用时不需要加（）"}}</**h2**> </**div**> <**script src="../js/vue.js"**></**script**> <**script**>  **const *app*** = **new** Vue({  **el**: **"#id1"**,  **data**: {  **firstname**: **"tom"**,  **lastname**: **"jack"** },  *// 计算属性 调用时不需要加（）* **computed**: {  fullname: **function** () {  **return this**.**firstname** + **this**.**lastname** }  },  **methods**: {  getfullname: **function** () {  **return this**.**firstname** + **this**.**lastname** }  }  }) </**script**> |

（1）计算属性的复杂操作

计算书的总价格



|  |
| --- |
| <**div id="id1"**>  <**h2**>总价格：{{totalPrice}}</**h2**> </**div**> <**script src="../js/vue.js"**></**script**> <**script**>  **const *app*** = **new** Vue({  **el**: **"#id1"**,  **data**: {  **books**: [  {**id**: 100,**name**:**""**,**price**: 112},  {**id**: 101,**name**:**""**,**price**: 33},  {**id**: 102,**name**:**""**,**price**: 1333},  {**id**: 103,**name**:**""**,**price**: 145},  {**id**: 104,**name**:**""**,**price**: 88},  ]  },  *// 计算属性 调用时不需要加（）* **computed**: {  totalPrice: **function** () {  **let** result = 0;  **for** (**let** i = 0;i < **this**.**books**.**length**;i++){  result += **this**.**books**[i].**price** *alert*(**"价格为："** + result)  }  **return** result  }  }  }) </**script**> |

4.1 计算属性的set和get

每一个计算属性都包含一个set和get，在上面的例子中，我们都是使用get来获取数据的，但有时还可以使用set方法。但是要记住计算属性一般没有set方法，是只读属性的、99%的情况下不用。

|  |
| --- |
| <**div id="id1"**>  <**h2**>{{fullname}}</**h2**> </**div**> <**script src="../js/vue.js"**></**script**> <**script**>  **const *app*** = **new** Vue({  **el**: **"#id1"**,  **data**: {  **firstname**: **"tom"**,  **lastname**: **"jack"** },  *// 计算属性 调用时不需要加（）* **computed**: {  **fullname**: {  *// 计算属性一般没有set方法，是只读属性的、99%的情况下不用。* set: **function** (newv) {  *alert*(**"----------"**+newv)  },  get: **function** () {  **return this**.**firstname** + **this**.**lastname** }  }  }  }) </**script**> |

4.1 计算属性和methods的区别

计算属性的值计算过一次之后，如果值不发生改变，会有缓存存储。而methods则会每次都执行计算函数。所以说起来计算属性的性能要比methods高很多。

## 5. 事件监听

在前端开发中，经常会和后端进行交互，这个时候我们就必须监听用户操作的时间、类型（单击、拖拽、键盘事件等）。在Vue中使用v-on进行监听。

5.1 v-on的简单介绍

作用是“绑定事件监听器”，简写是“@”。下面的例子中使用v-on来监听一个按钮的点击事件，还有就是将事件指定向methods中的一个函数。



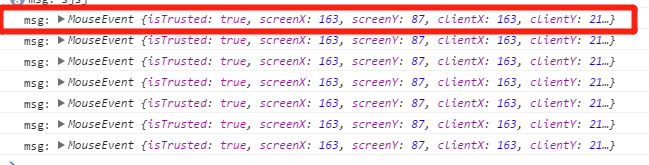
5.2 v-on的参数传递

当通过methods中定义方法，供@click调用时，需要注意的还有参数的问题。

* 方法不需要参数：那么方法后面的()可以不写，但是如果方法本身有一个参数时，那么默认会将原生事件event参数传递进去。

方法需要参数：同时需要参数和event时，可以通过$event传递参数。如果函数需要传递参数，但是没有传，这个时候Vue会默认将浏览器生成的event对象传递过来；如果函数需要传递参数和event，event可以使用$event传递。

|  |
| --- |
| <**div id="id1"**>  *<!-- 事件调用时没有参数，（）可以省略 -->* <**button v-on:click="btn1click()"**>按钮1</**button**>  <**button @click="btn1click"**>按钮2</**button**>  *<!-- 事件调用时有参数 如果函数需要传递参数，但是没有传，  这个时候Vue会默认将浏览器生成的event对象传递过来-->* <**button @click="btn2click"**>按钮3</**button**>  <**button @click="btn2click('sjsj')"**>按钮4</**button**>  *<!-- 事件调用时有参数 如果函数需要传递参数和event，event可以使用$event传递-->* <**button @click="btn3click(123,$event)"**>按钮5</**button**> </**div**> <**script src="../js/vue.js"**></**script**> <**script**>  **const *app*** = **new** Vue({  **el**: **"#id1"**, *// 挂在要声明的元素* **data**: { *// 定义数据* **message**: 0  },  **methods**: {  btn1click: **function** ($event) {  ***console***.log(**'btn1click'**)  },  btn2click(event){  ***console***.log(**"msg:"**,event)  },  btn3click(msg,event){  ***console***.log(**"++++++"**,msg,event)  },  }  }) </**script**> |



5.3 v-on的修饰符

在某些情况下，我们拿到event的目的可能是对一些事件进行处理。VUE提供了修饰符来帮助我们方便的处理一下事情

* .stop修饰符的作用：调用event.stopPropagation()，停止事件冒泡
* .prevent修饰符的作用：调用event.preventDefault()，阻止默认事件
* .once修饰符，按钮只会被点击一次（触发一次回调）。
* .native-监听组件根元素的原生事件



|  |
| --- |
| <**div id="id1"**>  *<!-- 1. .stop修饰符的作用：调用event.stopPropagation()，停止事件冒泡 -->* <**div @click="divclick"**>  jfdfghjkl  <**button @click.stop="btn1click()"**>按钮1</**button**>  </**div**>  *<!-- 2. .prevent修饰符的作用：调用event.preventDefault()，阻止默认事件 -->* <**br**>  <**form action="baidu"**>  <**input type="submit" value="提交" @click.prevent="subclick"**>  </**form**>  *<!-- 3. 监听摸个键盘的键帽 -->* <**input type="text" @keyup.enter="keyUp"**>  *<!-- 4. .once修饰符，按钮只会被点击一次 -->* <**button @click.once="btn1once()"**>按钮</**button**>  *<!-- -->  <!-- -->* </**div**> <**script src="../js/vue.js"**></**script**> <**script**>  **const *app*** = **new** Vue({  **el**: **"#id1"**, *// 挂在要声明的元素* **data**: { *// 定义数据* **message**: 0  },  **methods**: {  btn1click: **function** ($event) {  ***console***.log(**'btn1click'**)  ***event***.preventDefault()  },  divclick: **function** () {  ***console***.log(**'divclick'**)  },  subclick: **function** () {  ***console***.log(**'subclick'**)  },  keyUp: **function** () {  ***console***.log(**'keyUp'**)  },  btn1once: **function** () {  ***console***.log(**'btn1once'**)  }   }  }) </**script**> |

## 6. 条件和循环

6.1 v-if的简单实用



### 6.2 登录小案例

实现用户登录和邮箱登录功能切换。

|  |
| --- |
| <**div id="id1"**>  <**span v-if="isUser"**>  <**label for="username"**>用户账号</**label**>  <**input type="text" id="username" key="user-input" placeholder="用户账号"**>  </**span**>  <**span v-else**>  <**label for="email"**>用户邮箱</**label**>  <**input type="text" id="email" key="email-input" placeholder="用户邮箱"**>  </**span**>  *<!-- 如果我们不希望出现重复利用的情况，那么就要给其赋key属性，且key不同 -->* <**button @click="butCLick12"**>切换登录</**button**> </**div**> <**script src="../js/vue.js"**></**script**> <**script**>  **const *app*** = **new** Vue({  **el**: **"#id1"**,  **data**: {  **message**: 0,  **isUser**: **false** },  **methods**: {  butCLick12: **function** () {  **this**.**isUser** = !**this**.**isUser** }  }  }) </**script**> |

6.3 v-show简介

v-show和v-if的使用非常相似，也是决定一个元素是否渲染。

v-show和v-if的对比:

* v-if指令当条件为false时，在dom中不会存在对应的元素。
* v-show指令当条件为false时，在dom中的标签元素存在，只是将display属性设置为none

在开发的过程中如何选择呢？

* 当需要在显示和隐藏之间切换很频繁时，使用v-show。
* 当只有一次切换时，使用v-if。

|  |
| --- |
| <**div id="id1"**>  <**h2 v-if="isShow"**>  <**div**>刘嚎嚎</**div**>  <**div**>王雷</**div**>  <**div**>张晓林</**div**>  </**h2**>  <**h2 v-show="isShow"**>  <**div**>刘嚎嚎</**div**>  <**div**>王雷</**div**>  <**div**>张晓林</**div**>  </**h2**> </**div**> <**script src="../js/vue.js"**></**script**> <**script**>  **const *app*** = **new** Vue({  **el**: **"#id1"**,  **data**: {  **message**: 0,  **isShow**: **false** }  }) </**script**> |

6.4 v-for的使用

当我们有一组数据需要进行渲染时，我们就可以使用v-for来渲染。v-for的用法类似于js中的for循环，使用的格式如下：（v-for=”item in items”）,在使用时需要注意以下的几点。

* 在遍历的过程中，获取value："value in names"
* 在遍历的过程中，获取key value:"(value,key) in names"
* 在遍历的过程中，获取key value index:"(value,key,index) in names"

|  |
| --- |
| <**div id="id1"**>  *<!-- 1.在遍历的过程中，整体输出 -->* <**ul v-for="names in nameList"**>  <**li**>{{names.name}}</**li**>  <**li**>{{names.age}}</**li**>  <**li**>{{names.height}}</**li**>  </**ul**>  *<!-- 2.在遍历的过程中，如果只取一个值，那么我们取得是value -->* <**ul v-for="names in nameList"**>  <**li v-for="value in names"**>{{value}}</**li**>  </**ul**>  *<!-- 3.在遍历的过程中，获取key value:"(value,key) in names" -->* <**ul v-for="names in nameList"**>  <**li v-for="(value,key) in names"**>{{key}} {{value}}</**li**>  </**ul**>  *<!-- 4.在遍历的过程中，获取key value index:"(value,key,index) in names"-->* <**ul v-for="names in nameList"**>  <**li v-for="(value,key,index) in names"**>{{key}}+{{index}}+{{value}}</**li**>  </**ul**> </**div**> <**script src="../js/vue.js"**></**script**> <**script**>  **const *app*** = **new** Vue({  **el**: **"#id1"**,  **data**: {  **nameList**: [  {**name**:**"tom"**,**age**:12,**height**: 123},  {**name**:**"jack"**,**age**:22,**height**: 12233},  {**name**:**"wl"**,**age**:33,**height**: 1233},  {**name**:**"rose"**,**age**:55,**height**: 1233}]  }  }) </**script**> |

6.4 v-for使用Key属性



6.4 数组的相应式方法

* push方法：向最后插入一条数据,可以插入多个值  
  // this.leaters.push("last")
* pop方法：删除数组中的最后一个元素  
  // this.leaters.pop()
* shift方法：删除数组中的最开始一个元素  
  // this.leaters.shift()
* unshift方法：向数组前面添加多个元素，可一次添加多个元素  
  // this.leaters.unshift("21","222",{1:2})  
  // this.leaters.age = "21"
* splice:插入，替换，删除元素  
  // 5.1 删除-1-操作元素的位置；2-元素的个数，不传，删除所有  
  // this.leaters.splice(1,2)  
  // 5.2 替换-1-操作元素的位置；2-元素的个数，不传，删除所有；3后-替换的元素  
  // this.leaters.splice(1,2,'22','ww','awss')  
  // 5.3 插入-1-操作元素的位置；2-元素的个数:0，不传，删除所有；3后-替换的元素  
  // this.leaters.splice(1,0,12,3,4,4)
* 元素翻转  
  // this.leaters.reverse()
* 通过索引修改元素的值，但是页面不会发生响应式的修改  
  // this.leaters[1] = "wanglei"
* 通过vue方法set(修改的对象，索引值，修改后的值)来修改。  
  Vue.set(this.leaters,2,"王雷")

|  |
| --- |
| <**div id="id1"**>  <**ul**>  <**li v-for="lets in leaters"**>{{lets}}</**li**>  </**ul**>  <**button @click="btnClick"**>按钮</**button**> </**div**> <**script src="../js/vue.js"**></**script**> <**script**>  **const *app*** = **new** Vue({  **el**: **"#id1"**,  **data**: {  **leaters**: [1,2,3,4,5,6]  },  **methods**: {  btnClick: **function** () {  *// 1.push方法：向最后插入一条数据,可以插入多个值  // this.leaters.push("last")  // 2.pop方法：删除数组中的最后一个元素  // this.leaters.pop()  // 3.shift方法：删除数组中的最开始一个元素  // this.leaters.shift()  // 4.unshift方法：向数组前面添加多个元素，可一次添加多个元素  // this.leaters.unshift("21","222",{1:2})  // this.leaters.age = "21"  // 5.splice:插入，替换，删除元素  // 5.1 删除-1-操作元素的位置；2-元素的个数，不传，删除所有  // this.leaters.splice(1,2)  // 5.2 替换-1-操作元素的位置；2-元素的个数，不传，删除所有；3后-替换的元素  // this.leaters.splice(1,2,'22','ww','awss')  // 5.3 插入-1-操作元素的位置；2-元素的个数:0，不传，删除所有；3后-替换的元  // this.leaters.splice(1,0,12,3,4,4)  // 6. 元素翻转  // this.leaters.reverse()  // 7.通过索引修改元素的值，但是页面不会发生响应式的修改  // this.leaters[1] = "wanglei"  // 8.通过vue方法set(修改的对象，索引值，修改后的值)来修改。* Vue.set(**this**.**leaters**,2,**"王雷"**)  }  }  }) </**script**> |