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：

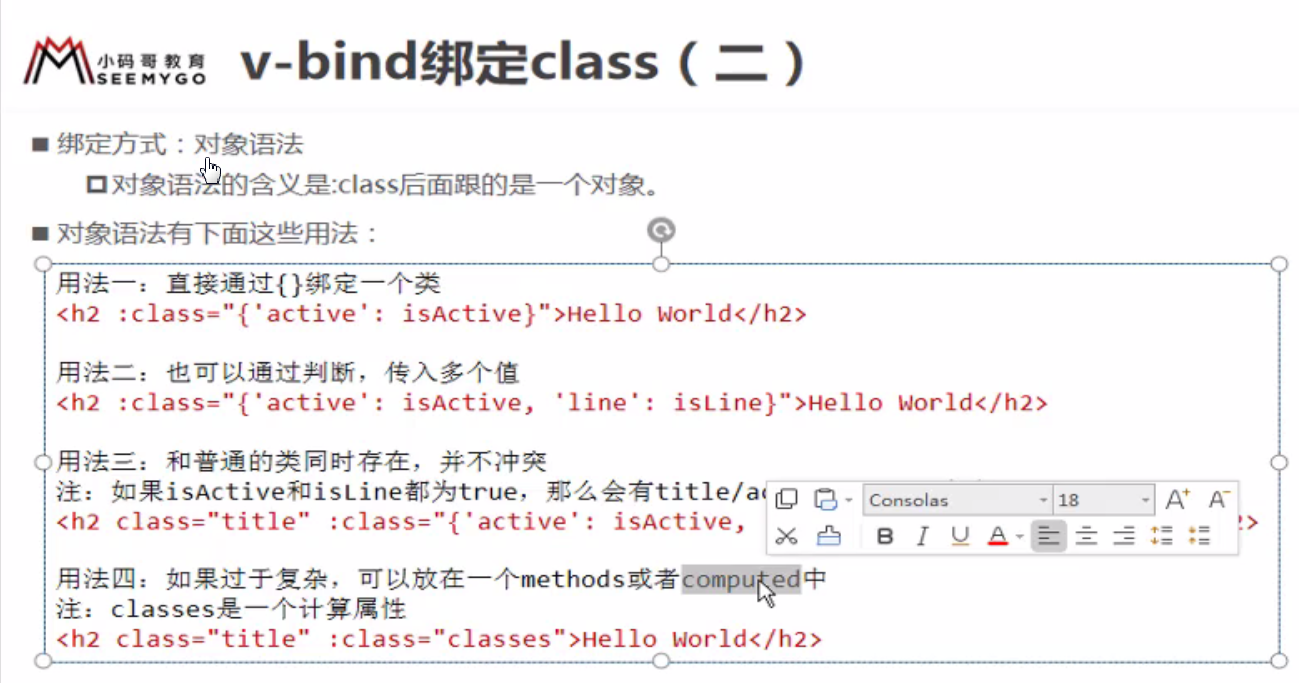
* 作用：动态的绑定属性。
* 缩写： : 语法糖式的写法，本质是v-bind。
* 预期：any object
* 参数：attrOrProp(属性)

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

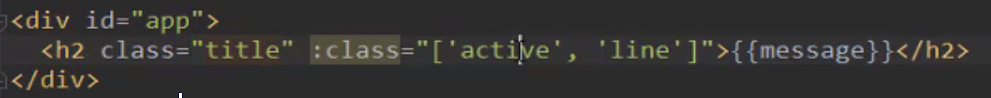
对象语法的含义：class后面跟的是一个对象。



|  |
| --- |
| <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。

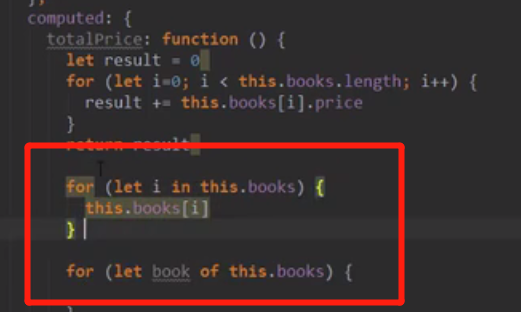
我们可以将上面的功能换成计算属性，放在computed中。。

（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。



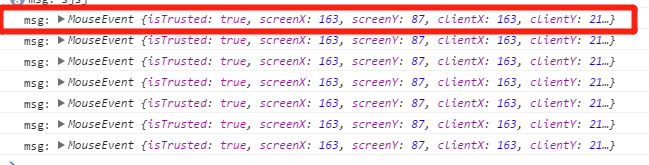
5.2 v-on的参数传递

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



6.5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**> |

6.6购物车小案例

|  |
| --- |
| Style.css  table {  border: 1px solid #e9e9e9;  border-collapse: collapse;  border-spacing: 0; } th, td {  padding: 8px 16px;  border: 1px solid #e9e9e9;  text-align: left; } th {  background-color: #f7f7f7;  color: chartreuse;  font-weight: 600; }  main.js  const ***app*** = new Vue({  el: "#app",  data: {  books: [  {id:1,name:"ww1",date:"wwww1",price:12.1,count:1},  {id:2,name:"ww2",date:"wwww2",price:12.2,count:2},  {id:3,name:"ww3",date:"wwww3",price:12.3,count:3},  {id:4,name:"ww4",date:"wwww4",price:12.4,count:4},  {id:5,name:"ww5",date:"wwww5",price:12.5,count:5}  ]  },  methods: {  increment: function (index) { // 加加  this.books[index].count++  },  oncrement: function (index) { // 减减  if (this.books[index].count > 0) {  this.books[index].count--  } else {  this.books[index].count = 0  }  },  btnRemove: function (index) {  this.books.splice(index,1)  }  },  computed:{  totalPrice: function () {  let totalPrice = 0  for (let i = 0;i < this.books.length;i++){  totalPrice += this.books[i].count \* this.books[i].price  }  return totalPrice  }  },  filters: {  showPrice: function(price) {  return '¥ ' + price.toFixed(2)  }  } })  index.html  <head>  <meta charset="UTF-8">  <title>购物车案例</title>  <link rel="stylesheet" href="style.css"> </head> <body> <div id="app">  <div v-if="books.length">  <table>  <thead>  <tr>  <th></th>  <th>书籍名称</th>  <th>出版日期</th>  <th>价格</th>  <th>购买数量</th>  <th>操作</th>  </tr>  </thead>  <tbody>  <tr v-for="(items, index) in books">  <!-- 不建议这样做，拓展起来不方便 -->  <!-- <td v-for="item in items">{{item}}</td>-->  <td>{{items.id}}</td>  <td>{{items.name}}</td>  <td>{{items.date}}</td>  <td>{{items.price | showPrice}}</td>  <td>  <button @click="oncrement(index)" v-bind:disabled="items.count <= 1">-</button>  {{items.count}}  <button @click="increment(index)">+</button>  </td>  <td>  <button @click="btnRemove(index)">移除</button>  </td>  </tr>  </tbody>  </table>  <h2>总价格： {{totalPrice | showPrice}}</h2>  </div>  <h2 v-else>购物车为空</h2> </div> <script src="../js/vue.js"></script> <script src="main.js"></script> </body> |

## 7. 表单绑定

7.1 v-model的基本使用

表单控件在实际的开发中是非常常见的，特别是对于用户信息的提交需要用到大量的表单。在vue中使用v-model指令来实现表单控件和数据的双向绑定。

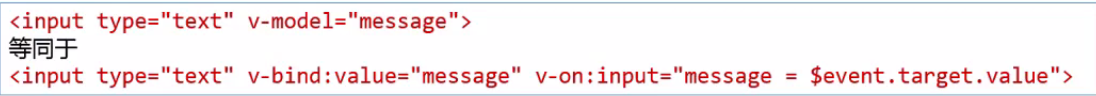
|  |
| --- |
| <div id="app">  <input type="text" v-model="message">  <h2>{{message}}</h2> </div>  <script src="../js/vue.js"></script> <script>  const ***app*** = new Vue({  el: "#app",  data: {  message: "wls"  }  }) </script> |

7.2 v-model的原理

v-model的实质是一个语法糖，它的背后实质上是两个操作合起来的。

* v-bind绑定一个value属性
* v-on给当前元素绑定一个input事件（获取input中输入了什么内容）。

也就是说下面的代码其实现的效果是一样的。



7.3 v-model和redio（单选框）

|  |
| --- |
| <div id="app">  <label><input type="radio" value="男" v-model="sex">男</label>  <label><input type="radio" value="女" v-model="sex">女</label>  <h2>您选择的性别是:{{sex}}</h2> </div> <script src="../js/vue.js"></script> <script>  const ***app*** = new Vue({  el: "#app",  data: {  message: "wls",  sex: ""  }  }) </script> |

7.4 v-model和checkbox（多选框）

复选框分为两种情况：单个勾选框和多个勾选框。

* 单个勾选框：v-model为布尔值，此时input的值并不会影响v-model的值。
* 多个勾选框：对应的data中的数据是一个数组，当选中其中一个或者多个数据时，都会将数据加入到data中的数组中。

代码如下：

|  |
| --- |
| <div id="app">  <!-- 单选框 -->  <h2>单选框</h2>  <label for="agreeId">  <input type="checkbox" id="agreeId" v-model="isAgree">同意协议  </label>  <h2>您选择的是:{{isAgree}}</h2>  <button v-bind:disabled="!isAgree">下一步</button>  <!-- 多选框 -->  <h2>多选框</h2>  <label>  <input type="checkbox" value="篮球" v-model="loveList">篮球  </label>  <label>  <input type="checkbox" value="足球" v-model="loveList">足球  </label>  <label>  <input type="checkbox" value="lol" v-model="loveList">lol  </label>  <label>  <input type="checkbox" value="DNF" v-model="loveList">DNF  </label>  <h2>您的爱好是:{{loveList}}</h2> </div> <script src="../js/vue.js"></script> <script>  const ***app*** = new Vue({  el: "#app",  data: {  message: "wls",  isAgree: false,  loveList: []  }  }) </script> |

7.5 v-model和select（选择框）

select和checkbox一样，也分为单选框和多选框。



|  |
| --- |
| <div id="app">  <h2>选择单个</h2>  <select v-model="fruit">  <option value="apple">apple</option>  <option value="香蕉">香蕉</option>  <option value="栗子">栗子</option>  <option value="芒果">芒果</option>  </select>  <h2>您选择的水果是:{{fruit}}</h2>   <h2>选择多个</h2>  <select v-model="fruits" multiple>  <option value="apple">apple</option>  <option value="香蕉">香蕉</option>  <option value="栗子">栗子</option>  <option value="芒果">芒果</option>  </select>  <h2>您选择的水果是:{{fruits}}</h2> </div> <script src="../js/vue.js"></script> <script>  const ***app*** = new Vue({  el: "#app",  data: {  message: "wls",  fruit: "芒果",  fruits: []  }  }) </script> |

7.6 v-model和值绑定

值绑定，使用v-bind:value=””语法，实质就是动态的给value赋值。因为在实质的开发过程中，表单中的值可能不是定死的，而是通过解析后台返回的数据使用v-bind:value=””或者:value=””来进行操作赋值。

这就是v-bind指令在input框中的操作，示例代码如下：

|  |
| --- |
| <div id="app">  <!-- 多选框值绑定 -->  <h2>多选框值绑定</h2>  <label v-for="orangeLove in orangeLoveList">  <input type="checkbox" :value="orangeLove" v-model="loveList">{{orangeLove}}  </label>  <h2>您的爱好是（多选框值绑定）:{{loveList}}</h2> </div> <script src="../js/vue.js"></script> <script>  const ***app*** = new Vue({  el: "#app",  data: {  message: "wls",  isAgree: false,  loveList: [],  orangeLoveList: ["篮球","足球","lol","DNF"]  }  }) </script> |

7.7 v-model和修饰符

* lazy修饰符：在默认的情况下，v-model默认是在input时间中同步输入框的数据的，也即是说，一旦数据发生改变的时候对应的data中的数据就会自动发生改变。而lazy修饰符可以让数据在失去焦点或者回车的时候才会更新。
* number修饰符：默认情况下，在输入框中我们无论输入的是字母或者是数字，都会被当成字符串类型进行处理。但是如果我们希望处理的是数字类型，那么最好直接将内容当做数字进行处理。number修饰符可以将输入框中的内容转化为数字类型。
* trim修饰符:如果输入的内容由很多的空格，通常我们希望将其去除。trim修饰符可以过滤内容左右两边的空格。

|  |
| --- |
| <div id="app">  <!-- 修饰符lazy -->  <input type="text" v-model.lazy="message">  <h2>{{message}}</h2>  <!-- 修饰符number -->  <input type="number" v-model.number="age">  <h2>{{age}}-{{typeof age}}</h2>  <!-- 修饰符trim -->  <input type="text" v-model.trim="name">  <h2>您的名字是：-{{name}}</h2> </div>  <script src="../js/vue.js"></script> <script>  const ***app*** = new Vue({  el: "#app",  data: {  message: "wls",  age: 8,  name: "w"  }  }) </script> |

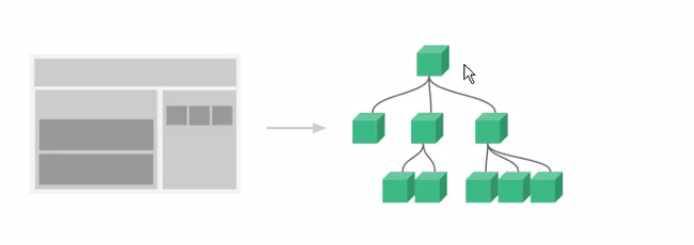
# 三. 组件化开发

## 1. 认识组件化

人面对复杂问题的处理方式：任何一个人处理信息的能力是有限的，所以面对一个大的问题时，不可能一次性搞定一大堆的问题，但是我们可以将问题进行拆解，将复杂的问题不断的拆解成简单的小问题，那么大问题最终也会迎刃而解。

组件化的思想也是类似的，如果我们将一个页面所有的处理逻辑都放在一起，代码内容肯定非常庞大、臃肿，处理起来非常的麻烦，而且不利于后去的维护、管理、拓展。但是我们将一个大的页面拆解成一个个小的功能块，每个功能块只完成自己的独立的功能，这样不仅简化了开发，而且后期的维护也会变得非常的方便。

所以总体来说，组件化开发的思想就是：我们将一个完整的页面拆分成非常多的组件；每个组件都用于实现页面的一个功能模块；而每一个小的组件又可以进行更小粒度的细分。如下图



## 2. 组件注册

2.1 注册组件的步骤

* 创建组件构造器

使用Vue.extend()函数创建组件构造器，在创建构造器时传入的是我们创建的template模板，该模板就是在使用到组件的时候要显示的HTML代码。但是事实上，从vue2.0开始已经基本上看不到使用该写法来注册组件的构造器了，它会使用我们后面我们学习到的语法糖来注册，但是核心的本质是不变的，源码中还是调用该方法来注册的。

* 注册组件

使用Vue.component()将刚才的组件构造器注册为一个组件，并且给他起一个组建的标签名称。所以这个时候需要传递两个参数：注册组件的标签名、创建好的组件构造器。

* 使用组件

组件必须挂在在某个vue的实例下，否则它是不会生效的。

整个组件化注册的流程单间代码如下所示。

|  |
| --- |
| <div id="app">  <!-- 使用vue的组件 -->  <my-cpn></my-cpn> </div>  <script src="../../js/vue.js"></script> <script>  // 1. 创建组件构造器  const ***cpn*** = Vue.extend({  template: '' +  '<div>' +  '<h2>组件标题</h2>' +  '<p>组件内容</p>' +  '</div>'  });  // 2. 注册组件，定义组件的标签名称  Vue.component('my-cpn',***cpn***);  // 3.1 创建vue实例，否则组件是不会生效的  const ***app*** = new Vue({  el: "#app"  }) </script> |

2.2 全局组件和局部组件

当我们通过Vue.component()注册组件时，注册的组件是全局属性的，这就意味着该组件可以在任意的一个VUE实例中使用。

但是在实际的开发过程中，我们一般不会注册太多或者只会注册一个VUE的示例，这样一来，就没必要编写太多的全局组件。这个时候，我们可以通过编写局部的组件来进行相应模块的开发。

那么什么是局部组件呢？答：如果我们注册的组件挂在到某一个VUE实例中，那么这个组件就是一个局部组件。使用属性components来注册局部组件。示例代码如下所示。

|  |
| --- |
| <div id="app1">  <!-- 使用vue的组件 -->  <cpn></cpn>  <ctn12></ctn12> </div> <div id="app2">  <cpn></cpn> </div> <script src="../../js/vue.js"></script> <script>  // 1. 创建组件构造器  const ***cpn*** = Vue.extend({  template: `  <div>  <h2>组件标题</h2>  <p>组件内容</p>  </div>  `  });  // 2. 注册组件-全局组件:意味着可以在多个VUE实例下使用该组件  Vue.component('cpn',***cpn***);  // 3.1 创建vue实例，否则组件是不会生效的  let ***app1*** = new Vue({  el: "#app1",  // 这个时候思考一下什么是局部组件？  // 答案：在一个vue实例中注册的组件就是局部组件，只能当前的示例中对应的标签模块中使用。  // 使用的关键字是：components-注册局部组件  components: {  // ctn12-标签名，cpn-组件构造器  ctn12: ***cpn*** }  })  let ***app2*** = new Vue({  el: "#app2"  }) </script> |

2.3 父组件和子组件



|  |
| --- |
| <div id="app1">  <!-- 使用vue的组件 -->  <cpn2></cpn2> </div> <script src="../../js/vue.js"></script> <script>  // 创建组件构造器1（子组件）  const ***cpn1*** = Vue.extend({  template: `  <div>  <h2>组件标题1</h2>  <p>组件内容1</p>  </div>  `  });  // 创建组件构造器1（父组件）  const ***cpn2*** = Vue.extend({  template: `  <div>  <h2>组件标题2</h2>  <p>组件内容2</p>  <cpn1></cpn1>  </div>  `,  components: {  cpn1: ***cpn1*** }  });  let ***app1*** = new Vue({  el: "#app1",  components: {  cpn2: ***cpn2*** }  }) </script> |

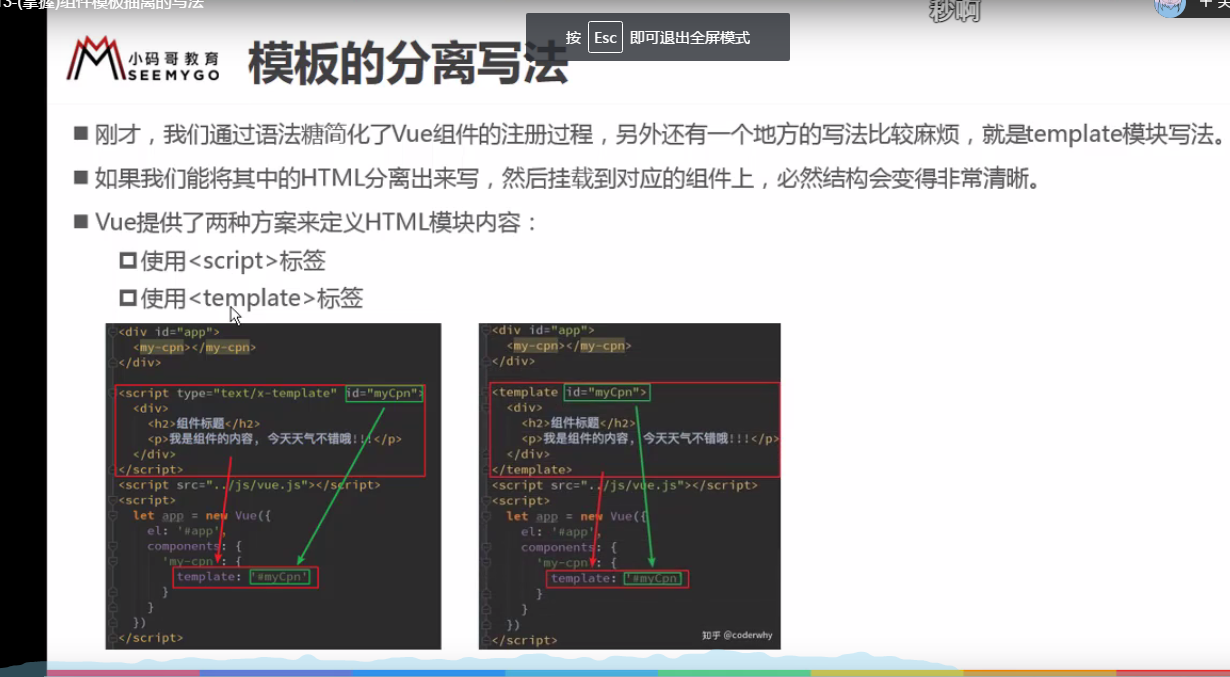
2.4 注册组件的语法糖写法

以上注册组件的方式看起来还是有点繁琐，VUE为了简化这个过程，提供了专门的语法糖式的写法，主要是省去了第一步Vue.extend()的步骤，在第二部中直接使用了一个对象来代替。

语法糖注册全局和局部组件的代码如下：

|  |
| --- |
| <div id="app">  <!-- 使用vue的组件 -->  <overralltag></overralltag>  <parttag></parttag> </div> <script src="../../js/vue.js"></script> <script>  // 2. 注册全局组件，语法糖写法  Vue.component('overralltag',{  template: `  <div>  <h2>我是全局组件</h2>  </div>`  });  const ***app*** = new Vue({  el: "#app",  // 注册局部组件，语法糖写法  components: {  'parttag': {  template: `<div> 我是局部组件</h2></div>`  }  }  }) </script> |

2.5 组件模板分离



刚才，通过语法糖简化了Vue组件的注册过程，另外还有一个地方的写法比较麻烦，就是template模块的写法。但是我们可以将其中的HTML代码单独分离出来，然后挂载到对应的组件上，这时候代码的结构看起来会清晰好多。在VUE中提供了两种方案来定义HTML模块的内容。

* 组件分离方式1：使用<template>标签
* 组件分离方式2：使用<script>，注意：类型必须是text/x-handlebars-template.

示例的代码如下：

|  |
| --- |
| <div id="app">  <!-- 使用vue的组件 -->  <overralltag></overralltag>  <parttag></parttag> </div> <!-- 组件分离方式1：使用template标签 --> <template id="parttagID">  <div><h2>我是局部组件</h2></div> </template> <!-- 组件分离方式2：使用script，注意：类型必须是text/x-handlebars-template --> <script type="text/x-template" id="overralltagID">  <div><h2>我是全局组件</h2></div> </script> <script src="../../js/vue.js"></script> <script>  Vue.component('overralltag',{  template: "#overralltagID"  });  const ***app*** = new Vue({  el: "#app",  components: {  parttag: {  template: "#parttagID"  }  }  }) </script> |

2.6 组件的data数据

在说明本章节内容之前，我们想一下组件可以访问Vue实例的数据吗？答案是不能访问，而且即使可以访问，我们也不应该访问，因为如果将所有的访问放在vue实例中的话，VUE的实例将会变得非常的臃肿，所以结果就是VUE的实例应该有保存自身数据的地方

组件是一个单独模块的封装，这个模块有属于自身的HTML语法，那么也理应有保存自己数据的地方。组件对象自身也有一个data属性（也有methods属性），但是要注意的是这个data必须是一个函数，并且函数返回一个对象，在返回的对象中存储的就是需要保存的组件数据。

|  |
| --- |
| <div id="app">  <!-- 使用vue的组件 -->  <overralltag></overralltag>  <parttag></parttag> </div> <!-- 组件分离方式1：使用template标签 --> <template id="parttagID">  <div><h2>{{parttagMessage}}</h2></div> </template> <!-- 组件分离方式2：使用script，注意：类型必须是text/x-handlebars-template --> <script type="text/x-template" id="overralltagID">  <div><h2>{{overralltagMessage}}</h2></div> </script> <script src="../../js/vue.js"></script> <script>  Vue.component('overralltag',{  template: "#overralltagID",  data:function () {  return {  overralltagMessage: "我是全局组件"  }  }  });  const ***app*** = new Vue({  el: "#app",  components: {  parttag: {  template: "#parttagID",  data: function () {  return {  parttagMessage: "我是局部组件"  }  }  }  }  }) </script> |

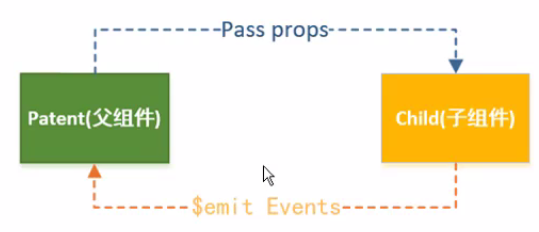
## 3. 组件数据传递

3.1 父子组件的通信

在上一章节中，我们学习到子组件是不能引用父组件或者Vue实例的数据的。但是在实际的开发中，一些数据往往需要从上层传递到下层。例如我们在一个页面中，从服务器端请求到了非常多的数据，但是这些数据中有一部分不是当前页面需要展示的，而是需要下面的子组件页面进行展示，这个时候，并不会让子组件再发送一次网络请求，而是让父组件将数据发送给子组件。

如何进行父子组件之间的通信？

* 在Vue中 父组件通过props向子组件传递数据；
* 子组件通过自定义事件向父组件传递数据。



（1）父传子：props的用法

在组件中，使用props属性来声明需要从父极接收到的数据，它有两种使用方式

* 字符串数组、数组中的字符串就是传递时的名称
* 对象，对象可以设置传递时的类型，也可以设置默认值等。

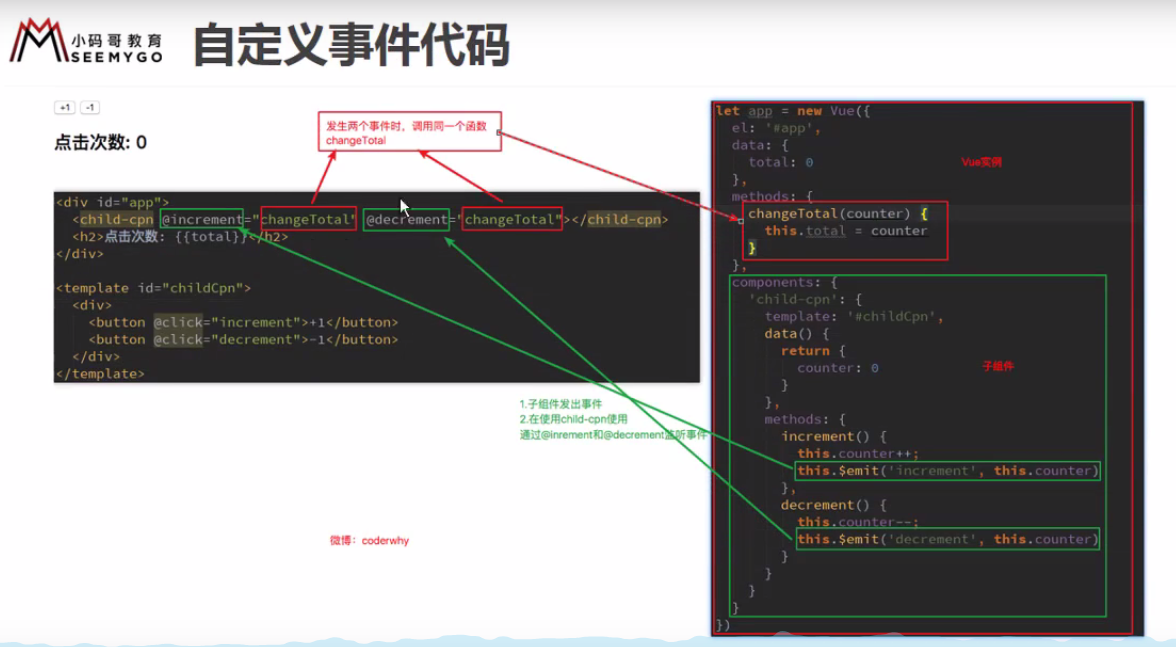
|  |
| --- |
| <div id="app">  <!-- 使用vue的组件 -->  <cpn v-bind:cmovies="movies" :cmessage="message"></cpn> </div> <template id="cpn">  <div>  <h2>{{cmovies}}</h2>  <h2>{{cmessage}}</h2>  </div> </template> <script src="../../js/vue.js"></script> <script>  // 父组件向子组件传递数据：props  // cpn为创建的组件  const ***ccpn*** = {  template: "#cpn",  data: function () {  return {}  },  // 1. 父组件向子组件传递数据，子组件中使用props属性进行接收  // props: ["cmovies","cmovies"]  // 2. 传递对象  props: {  // 1. 基础类型检查（null匹配任何类型）  // cmovies: Array,  // cmessage: String   // 2. d多个可能的类型  // cmessage: [String,Array]   // 3. 提供默认值，必须传的值  cmessage: {  type: ***String***,  default: "晚上连接",  required: true // 该项必传递  },  // 4. default:带有默认值的数据  cmovies: {  type: ***Array***,  // vue2.5.17，default写法正确，以上default必须的是一个函数  // default: [],  default(){  return [] // 现在推荐这么写 v2.6.11  }  },  // 5. 自定义的验证函数  // c: {  // validator: function (value) {  // return ["釜山行","生化危机","行尸走肉","明日传奇","映照万界"].indexOf(value) !== -1  // }  // }  }  }  const ***app*** = new Vue({  el: "#app",  data: {  movies: ["釜山行","生化危机","行尸走肉","明日传奇","映照万界"],  message: "我是呃呃呃呃呃呃呃呃呃呃呃呃传奇"  },  components: {  cpn: ***ccpn*** }  }) </script> |

（2）子传父：自定义事件

props用于父组件向子组件传递数据，还有一种比较常见的是子组件传递数据或者事件到父组件中，这种情况该如何来处理呢？这个时候我们就需要用到自定义事件了。

自定义事件一般当子组件向父组件传递数据时，就要用到；还有就是v-on不仅可以用于监听DOM事件，还可以用于监听组件间的自定义事件。

其一般的通信流程为：在子组件中，通过this.$emit()函数来发射、触发事件；在父组件中，通过v-on来监听子组件的事件。



|  |
| --- |
| <div id="app">  <cpn @itemclick="cpnclick"></cpn> </div> <template id="cpn">  <div>  <button v-for="item in categories"  v-on:click="btnClick(item)">  {{item.name}}  </button>  </div>  </template> <script src="../../js/vue.js"></script> <script>  // 子组件  const ***ccpn*** = {  template: "#cpn",  data: function () {  return {  categories: [  {id: 111,name: "热门推荐"},  {id: 112,name: "手机数码"},  {id: 113,name: "家用电器"},  {id: 114,name: "数码办公"},  ]  }  },  methods: {  btnClick: function (item) {  // 发射自定义事件  this.$emit('itemclick',item)  }  }  }  const ***app*** = new Vue({  el: "#app",  data: {  },  methods: {  cpnclick: function (item) {  ***console***.log('您点击的是：',item)  }  },  components: {  cpn: ***ccpn*** }  }) </script> |