# Vue.js

一、什么是Vue.js

Vue.js 是目前最火的一个前端框架，React是最流行的一个前端框架（React除了开发网站，还可以开发手机App， Vue语法也是可以用于进行手机App开发的，需要借助于Weex）

Vue.js 是前端的主流框架之一，和Angular.js、React.js 一起，并成为前端三大主流框架！

Vue.js 是一套构建用户界面的框架，只关注**视图层**，它不仅易于上手，还便于与第三方库或既有项目整合。（Vue有配套的第三方类库，可以整合起来做大型项目的开发）。

1、前端的主要工作：

主要负责MVC中的V这一层；主要工作就是和界面打交道，来制作前端页面效果；

二、提高开发效率的发展历程

1、原生JS

有兼容性问题，不同的浏览器有自己的实现。

2、jQuery之类的类库

把兼容性都给处理好了，假如要设置表格，就要拼接字符串，然后去操作dom，这样不方便，大家的拼接方式可能不一样。

1. 前端模板引擎

它提供一个模板，我们去填数据就行，解决了频繁操作dom问题。  
 [

{id:1, name:'hello'}

{id:3, name:'hello'}

{id:2, name:'hello'}

{id:4, name:'hello'}

]

点击id这一列，它会重新进行排序，然后模板引擎重新渲染，但是有个问题，1，4的顺序是不用改变的，但是它被重新渲染了，如果有上千百条的数据，性能问题就不得不考虑。

1. Vue.js

能够帮助我们减少不必要的DOM操作；提高渲染效率；双向数据绑定的概念：通过框架提供的指令，我们前端程序员只需要关心数据的业务逻辑，不再关心DOM是如何渲染的了。

在Vue中，一个核心的概念，就是让用户不再操作DOM元素，解放了用户的双手，让程序员可以更多的时间去关注业务逻辑。

三、框架和库的区别

1、框架：是一套完整的解决方案；对项目的侵入性较大，项目如果需要更换框架，则需要重新架构整个项目。

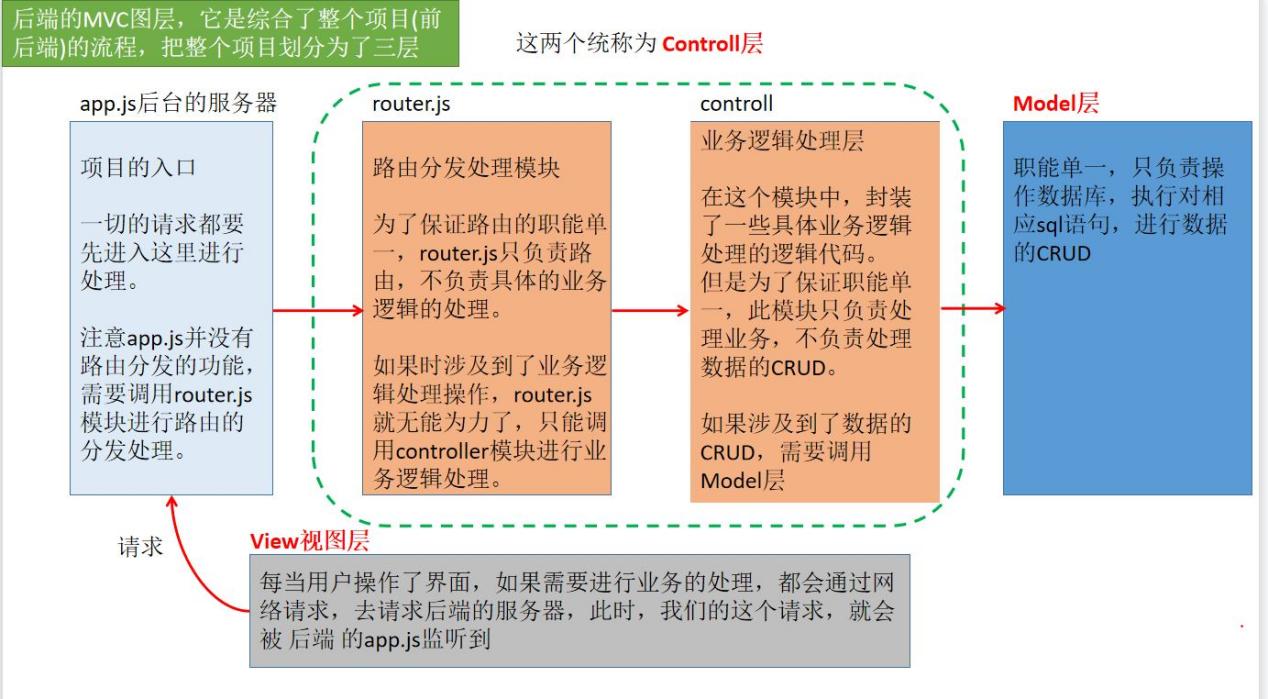
node 中的 express。

2、库（插件）：提供某一个小功能，对项目的侵入性较小，如果某个库无法完成某些需求，可以很容易切换到其它库实现需求。

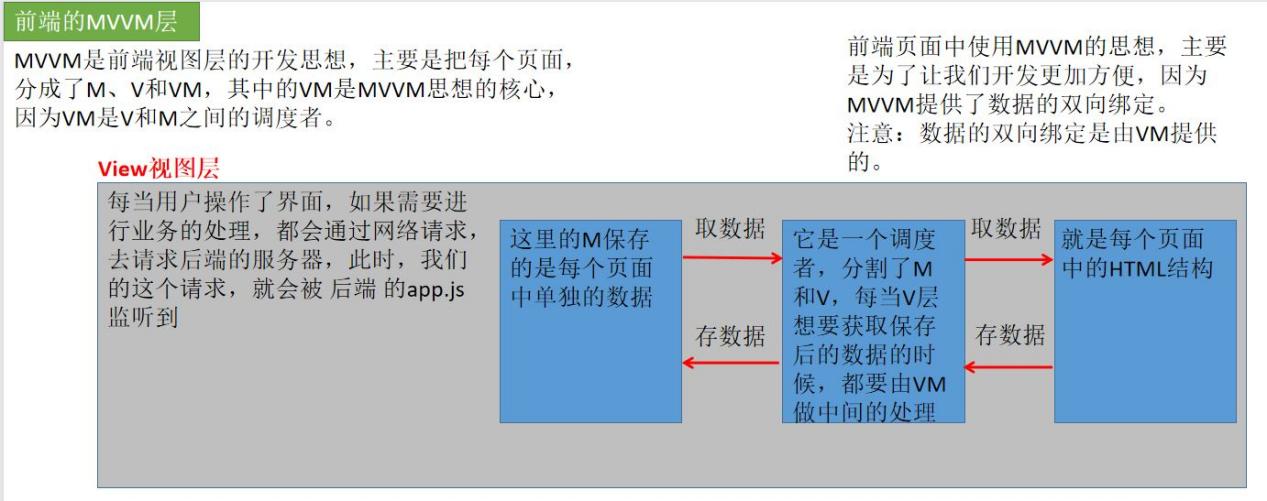
如从jQuery 切换到 Zepto，它们的API基本是大同小异的。

四、Node（后端）中的 MVC 与前端中的 MVVM 之间的区别

1. MVC 是后端（站在整个项目层面上）的分层开发概念。



2、MVVM是前端视图层的概念，主要关注于 视图层分离，也就是说：MVVM把前端的视图层，分为了 三部分 Model、View、VM（ViewModel）



CRUD是指在做计算处理时的增加(Create)、查询(Retrieve)（重新得到数据）、更新(Update)和删除(Delete)几个单词的首字母简写。主要被用在描述软件系统中数据库或者持久层的基本操作功能。

1. Vue.js 基本代码 和 MVVM 之间的对应关系
2. 第一个Vue实例

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Vue案例</title>  </head>  <body>  <!-- Vue 实例所控制的这个元素区域，就是我们的 V -->  <div class="box">  <p>{{ msg }}</p>  </div>  <!-- 当我们导入包之后，在浏览器的内存中，就多了一个 Vue 构造函数 -->  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <script type="text/javascript">  // 创建一个Vue的实例，我们创建出来的这个 vm 对象，就是我们 MVVM 中的 VM 调度者。  var vm = new Vue({  el: '.box', // 表示Vue的实例，要控制页面上的哪个区域，选择器。  // data 就是 MVVM 中的 M，专门用来保存每个页面的数据，它存放的是 el 中要用到的数据。  data: {  msg: '你好啊，我是vue',  // 通过 Vue 提供的指令，很方便就能把数据渲染到页面上，程序员不再手动操作DOM元素了，前端的 Vue 之类的框架，不提倡我们手动操作DOM元素了。  }  });  </script>  </body>  </html>  1 |

2、插值

（1）数据绑定最常见的形式就是使用“Mustache”语法 (双大括号) 的文本插值：双大括号会将数据解释为普通文本，而非 HTML 代码。

|  |
| --- |
| <span>Message: **{{ msg }}**</span> |

Mustache 标签将会被替代为对应数据对象上 msg 属性的值。无论何时，绑定的数据对象上 msg 属性发生了改变，插值处的内容都会更新。

（2）v-once

通过使用 v-once 指令，你也能执行一次性地插值，当数据改变时，插值处的内容不会更新。但请留心这会影响到该节点上的其它数据绑定：

|  |
| --- |
| <span v-once>这个将不会改变: {{ msg }}</span> |

（3）使用 JavaScript 表达式

这些表达式会在所属 Vue 实例的数据作用域下作为 JavaScript 被解析。有个限制就是，每个绑定都只能包含单个表达式，所以下面的例子都不会生效。

|  |
| --- |
| <!-- 这是语句，不是表达式 -->  {{ var a = 1 }}  <!-- 流控制也不会生效，请使用三元表达式 -->  {{ if (ok) { return message } }} |

3、指令

指令 (**Directives**) 是带有 v- 前缀的特殊 attribute。指令 attribute 的值预期是**单个 JavaScript 表达式** (v-for 是例外情况，稍后我们再讨论)。指令的职责是，当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM。

注意：在一些自定的指令中，v-color = “red”，跟 v-color = “**’**red**’**”，这两者是不一样的，前者会去找data中的red属性，而后者是一个字符串red来着。

（1）v-cloak：能解决 插值表达式 闪烁的问题

在慢速请求的情况下，{{ msg }}还没有渲染出来，它还在加载vue库。它在加载完毕之后，就会把这个指令移除了。刷新快点就能看到

（2）v-text

v-text是不会有闪烁的问题的。如果它标签里边原来有内容，那么原来的内容会闪烁的，但是一般不会有这个情况。加内容似乎没有意义。

v-text解析完毕后，会覆盖元素中原本的内容，**但是 插值表达式 只会替换**自己的这个占位符，不会把整个元素的内容清空的。

（3）v-html

{{ }}和v-text是不会解析标签的，只会把它们解析成普通的文本，它们在没有显示出来是不会占位置。

v-html也不会有闪烁问题。它会覆盖原有的内容

|  |
| --- |
| <style type="text/css">  [v-cloak] {  display: none;  }  </style>  <body>  <div class="box">  <p v-cloak>==========={{ msg }}=========</p>  <p v-text='msg'>啊啊啊啊啊啊啊啊啊</p>  <div v-html='msgWithHTML'>66666666666</div>  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <script type="text/javascript">  var vm = new Vue({  el: '.box',  data: {  msg: '你好啊，我是vue',  msgWithHTML: '<h1>我是大标题啊</h1>',  }  });  console.log(vm);  </script>  </body> |

（4）v-bind

是Vue中提供用于绑定属性的指令，把属性和变量绑定起来，它把属性值当成js代码来执行，可以写合法的js表达式。

v-bind指令可以被简写为 :要绑定的属性。

|  |
| --- |
| <input type="button" value="按钮" **v-bind**:title="**myTitle**">  var vm = new Vue({  el: '.box',  data: {  **myTitle**: '我是v-bind绑定的属性title',  }  });  如果不是合法的js表达式代码则会报错。 |

（5）v-on

v-on:click=”xxx”，xxx表示函数的名字。后边是能带小括号跟参数的

简写@click=

methods属性中定义了当前的Vue实例所有可用的方法。

|  |
| --- |
| <input type="button" value="按钮" v-bind:title="myTitle" v-on:click='show'>  var vm = new Vue({  el: '.box',  methods: {  show: function () {  console.log('hello i am v-on:click');  }  }  }); |

（6）this的使用，跑马灯小案例

在vm实例中，如果想要获取 data 上的数据或者想要调用 methods 中的方法，就必须通过 this 数据属性名或 this 方法名，来进行访问，这里的this 就表示我们new出来的vm实例对象，它在构造函数里边就用this

Vue它会监听自身data数据的改变，如果数据改变了，那么它会自动同步跟着刷新，从data上同步到页面中去，我们只需关心数据，不需要考虑如何重新渲染DOM页面。

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Vue案例</title>  <style type="text/css">  [v-cloak] {  display: none;  }  </style>  </head>  <body>  <div class="box">  <input type="button" value="跑起来" @click="run"/>  <input type="button" value="停止" @click="stop"/>  <h4 v-text="msg"></h4>  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <script type="text/javascript">  var vm = new Vue({  el: '.box',  data: {  msg: '跑起来，跑马灯，gogo',  timeId: null,  },  methods: {  run: function () {  // 为什么访问msg没有通过data，访问data却为undefined  // console.log(this.msg);  if (this.timeId) {  return;  // 这种处理方式只能在时间比较快执行下才起作用，400毫秒太慢了，一直点击的话，会出现停止现象  // clearInterval(this.timeId);  // this.timeId = null;  }  this.timeId = setInterval(() => {  var first = this.msg.substring(0, 1);  var end = this.msg.substring(1);  this.msg = end + first;  }, 400);  },  stop: function () {  clearInterval(this.timeId);  this.timeId = null;  }  }  });  </script>  </body>  </html>  这里定时函数，它那箭头函数是属于run的，把它当实参传递进去而已，并不是属于setInterval的。还可以通过bind，不用马上执行。  this.timeId = setInterval(function() {  var first = this.msg.substring(0, 1);  var end = this.msg.substring(1);  this.msg = end + first;  }**.bind(this)**, 400); |

4、事件修饰符

1. .stop：阻止冒泡，从里到外，想从谁开始阻止就给谁加

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>study Vue</title>  <style type="text/css">  .box1 {  padding: 50px;  background-color: pink;  }  .box2 {  width: 200px;  height: 200px;  display: table-cell;  vertical-align: middle;  text-align: center;  background-color: lightskyblue;  }  </style>  </head>  <body>  <div class="box1" @click="box1Click">box1  <div class="box2" @click="box2Click">box2  <input type="button" value="按钮" **v-on:click.stop**="btn">  </div>  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <script type="text/javascript">  var vm = new Vue({  el: '.box1',  data: {  },  methods: {  box1Click: function () {  console.log('box1 事件发生');  },  box2Click: function () {  console.log('box2 事件发生');  },  btn: function () {  console.log('btn 事件发生');  },  }  });  </script>  </body>  </html> |

（2）.prevent：阻止默认事件

|  |
| --- |
| 如果把a链接写到box1区域之后，那么vue就控制不到了  <div class="box1" @click="box1Click">box1  <div class="box2" @click="box2Click">box2  <input type="button" value="按钮" v-on:click.stop="btn">  </div>  <a href="http://www/baidu.com" **@click.prevent="linkClick"**>前去百度</a>  </div> |

（3）.capture：添加事件侦听器时使用事件捕获模式，从外到里（默认的是冒泡模式）

|  |
| --- |
| <div class="box1" **@click.capture="box1Click"**>box1  <div class="box2" @click="box2Click">box2  <input type="button" value="按钮" v-on:click="btn">  </div>  <a href="http://www/baidu.com" @click.prevent="linkClick">前去百度</a>  </div>  box1 事件发生  btn 事件发生  box2 事件发生 |

（4）.self：只当事件在该元素本身（比如不是子元素）触发时触发回调（如冒泡事件它是被动触发的，只有点击自己才执行）

|  |
| --- |
| <div class="box1" @click="box1Click">box1  <div class="box2" **@click.self="box2Click"**>box2  <input type="button" value="按钮" v-on:click="btn">  </div>  <a href="http://www/baidu.com" @click.prevent="linkClick">前去百度</a>  </div>  btn 事件发生  box1 事件发生 |

（5）.once：事件只触发一次

如果是被冒泡事件触发的，它也是照算执行了一次，后续就不会再触发了

|  |
| --- |
| <div class="box1" @click="box1Click">box1  <div class="box2" @click="box2Click">box2  <input type="button" value="按钮" **v-on:click.once="btn**">  </div>  <a href="http://www/baidu.com" @click.prevent="linkClick">前去百度</a>  </div> |

（6）.self和.stop的区别

.self只会阻止自己身上的冒泡行为的触发，并不会阻止冒泡行为的传递。

5、指令v-model实现双向数据绑定（唯一）

表单元素（input之类的）就能跟用户进行交互。

（1）v-bind 只能实现数据的单向绑定，从 M（数据） 自动绑定到 V（视图） ，无法实现数据的双向绑定。

|  |
| --- |
| 1  2 |

（2）v-model 指令，可以实现表单元素和 Model 中数据的双向绑定

注意：v-model 只能运用在表单元素中去。

input（radio、text、address、email....） select、checkbox、textarea

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>study Vue</title>  </head>  <body>  <div class="box1">  {{ msg }}  <input type="text" v-model="msg">  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <script type="text/javascript">  var vm = new Vue({  el: '.box1',  data: {  msg: '你好啊，我是Tom,找Jerry',  },  });  </script>  </body>  </html>  1 |

（3）计算器小案例

select的value属性它用于在选择的下拉列表中指定<option>元素value的值。

|  |
| --- |
| <body>  <div class="box1">  <input type="number" **v-model**="n1">  <select v-model="opt">  <option value='+'>+</option>  <option value='-'>-</option>  <option value='\*'>\*</option>  <option value='/'>/</option>  </select>  <input type="number" **v-model**="n2">  <input type="button" value="=" v-on:click="calc">  <input type="number" **v-model**="result">  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <script type="text/javascript">  var vm = new Vue({  el: '.box1',  data: {  n1: 0,  n2: 0,  result: 0,  opt: '+',  },  methods: {  calc: function () {  /\*switch (this.opt) {  case '+':  this.result = parseInt(this.n1) + parseInt(this.n2);  break;  case '-':  this.result = parseInt(this.n1) - parseInt(this.n2);  break;  case '\*':  this.result = parseInt(this.n1) \* parseInt(this.n2);  break;  case '/':  this.result = parseInt(this.n1) / parseInt(this.n2);  break;  }\*/  // 通过拼接字符串来处理结果，eval,在实际中不能常用，耗费性能  var str = 'parseInt(this.n1) '+ this.opt +' parseInt(this.n2)';  this.result = eval(str);  }  }  });  </script>  </body>  1 |

6、在Vue中使用class样式

（1）第一种，直接传递一个数组，**类名需要加引号**，这里的 class 需要使用 v-bind 做数据绑定，这种**不方便使用**，不如原来的方式。

|  |
| --- |
| <h3 v-bind:class="[**'**thin**'**, **'**italic**'**]">{{ msg }}</h3> |

（2）第二种，在数组中使用三元表达式

|  |
| --- |
| <h3 v-bind:class="['thin', 'italic', **flag?'active':''**]" v-cloak>{{ msg }}</h3> |

（3）在数组中使用对象来代替三元表达式，提高代码的可读性

|  |
| --- |
| <h3 v-bind:class="['thin', 'italic', **{'active': flag}**]" v-cloak>{{ msg }}</h3> |

（4）在为 class 使用v-bind 绑定对象的时候，对象的属性是类名，由于 对象的属性可带引号，也可不带引号，属性值 是一个标识符

|  |
| --- |
| <h3 v-bind:class="**classObj**" v-cloak>{{ msg }}</h3>  data: {  msg: 'vue的类样式操作。',  flag: true,  classObj: {  thin: true,  italic: true,  active: true,  }  }, |

7、在Vue使用内联样式

（1）直接在元素上通过 :style 的形式，书写样式对象。

|  |
| --- |
| <h3 :style="{color: 'red', **'font-weight'**: '200',}" v-cloak>{{ msg }}</h3> |

（2）将样式对象，定义到 data 中，并直接引用到 :style 中，引用对象就不用加引号了，相当于是使用变量。

如果对象属性有横线，就要加上单引号，如font-size。

|  |
| --- |
| <h3 :style="styleObj" v-cloak>{{ msg }}</h3>  var vm = new Vue({  el: '.box',  data: {  msg: '你好啊，我是h3',  styleObj: {  color: 'red',  'font-size': '18px',  },  },  }); |

（3）在 :style 中通过数组，引用多个 data 上的样式对象

|  |
| --- |
| <h3 :style="[styleObj, styleObj1]" v-cloak>{{ msg }}</h3>  data: {  msg: '你好啊，我是h3',  styleObj: {  color: 'red',  'font-size': '18px',  },  styleObj1: {  'font-style': 'italic',  }  }, |

8、v-for

迭代跟循环是一样的，说法不同。

（1）迭代数组，如果是数组形式的，那么能第一个参数是它的值，第二个参数是它的索引。

|  |
| --- |
| <div class="box">  <ul>  <li v-for="**item** in **arr**" v-text="**item**"></li>  </ul>  </div>  var vm = new Vue({  el: '.box',  data: {  arr: [1,2,3,4,5,6,7],  },  methods: {  }  });  1 |

（2）迭代对象数组，如果是数组形式的，那么能第一个参数是它的值，第二个参数是它的索引。

|  |
| --- |
| <li v-for="(val, index) in objArr">索引：{{ index }}---------属性值：{{ val.name }}----{{ val.color }}</li>  data: {  objArr: [{ name: 'Tom', color: 'blue', },  { name: 'Jerry', color: 'yellow', },  { name: 'Jon', color: 'sky', }],  },  1 |

（3）迭代对象属性，对象的话v-for有三个属性。

注意，在遍历对象身上的键值对的时候，除了有 val(第一个是属性值)、key(第二个是属性)，在第三个位置还有一个索引（这个索引基本上不会用到）。

|  |
| --- |
| <li v-for="(val, key, index) in objArr">索引index：{{ index }}---------属性key：{{ key }}----属性值val：{{ val }}</li>  1 |

（4）迭代数字

in 后面我们放过 普通数组、对象数组、对象、还可以是数字。

注意：如果使用 v-for 迭代数字的话，前面的 count 值从 1 开始

|  |
| --- |
| <li v-for="count in 10">count: {{ count }}</li>  1 |

1. key

在组件中使用v-for时，要注意key是必须使用的，每次for循环的时候，通过指定key来标识当前循环的这一项的唯一身份。

v-for 循环的时候，key 属性只能使用number，或者 string ，在使用的时候，必须使用 v-bind 属性绑定的形式（因为它的值是变化，不加v-bind就变成了字符串，固定了），指定 key 的值。

在组件中使用 v-for 循环的时候，或者在一些特殊的情况的中，如果 v-for 有问题，就必须指定唯一的**字符串、数字类型**的 key 值。

|  |
| --- |
| 初步简单理解，不加key，你选第五个checkbox，在渲染的时候checkbox只记得是第五个被选，只要是第五就行。  加了key，就记得是item.id为5的那个。保证我们的data数据和页面上的数据，让它强制关联起来，加了key(一定要具有唯一性) id的checkbox跟内容进行了一个关联。是我们想达到的效果  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Vue study</title>  </head>  <body>  <div class="box">  id:<input type="text" v-model="id">  name:<input type="text" v-model="name">  <input type="button" value="添加" @click="add">  <p v-for="item in list" **v-bind:key="item.id"**>  <input type="checkbox">  id:{{item.id}}-----name: {{item.name}}  </p>  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <script type="text/javascript">  var vm = new Vue({  el: '.box',  data: {  list: [  {id: 1, name: 'Tom'},  {id: 2, name: 'Jerry'},  {id: 3, name: 'Jon'},  {id: 4, name: 'Mike'},  ],  id: 0,  name: '',  },  methods: {  add: function () {  // this.list.push({id: this.id, name: this.name});  this.list.unshift({id: this.id, name: this.name});  }  },  });  </script>  </body>  </html>  不加key的情况  1 2、 |

1. v-for 也可以去迭代一个函数中的数据，这样能动态切换数据，见品牌案例实例运用

v-for中的数据都是直接从 data 上的list中直接诶渲染过来的，这样是固定的了。

现在，我们自定义一个 search 的方法，同时把所有的关键字，通过传参的形式，传递给了 search 方法。

在 search 方法内部，通过 执行 for循环，把所有符合搜索关键字的数据，保存到一个新数组中，再返回。

9、v-if和v-show

（1）v-if，每次都会重新删除，或创建元素

有较高的切换性能消耗。

如果元素涉及到频繁的切换，最好是不要使用 v-if，推荐使用v-show。

（2）v-show，每次不会重新进行DOM操作，只是切换了元素的 display:none样式

有较高的初始渲染消耗（就是flag为false，它还是会display渲染了，if什么都不会去搞）。

如果元素可能永远也不会被显示出来被用户看到，则推荐使用 v-if，而且不会被频繁切换。

|  |
| --- |
| <div class="box">  <input type="button" @click="toggle" value="切换">  <p v-if="flag">我是v-if控制的</p>  <h3 v-show="flag">我是v-show控制的</h3>  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <script type="text/javascript">  var vm = new Vue({  el: '.box',  data: {  flag: true,  },  methods: {  toggle: function () {  this.flag = !this.flag;  }  },  });  </script>  1  2 |

10、小总结：

Vue控制的区域不能是body（1.0版本是可以的）

1. 品牌案例

分析：

获取到 id 和 name ，直接从 data 上面获取

组织出一个对象

把这个对象，调用数组的相关方法，添加到当前的data上的list中

在Vue中，已经实现了数据的双向绑定，每当我们修改了 data 中的数据，Vue会默认监听到数据的改动，自动把最新的数据，应用到页面上。

我们更多的是在进行 VM 中的Model 数据的操作，同时，在操作Model数据时候，指定业务逻辑操作

看不到vue data上的数据，可以通过装vue-devtools，在F12中可以看到，数据的变化。

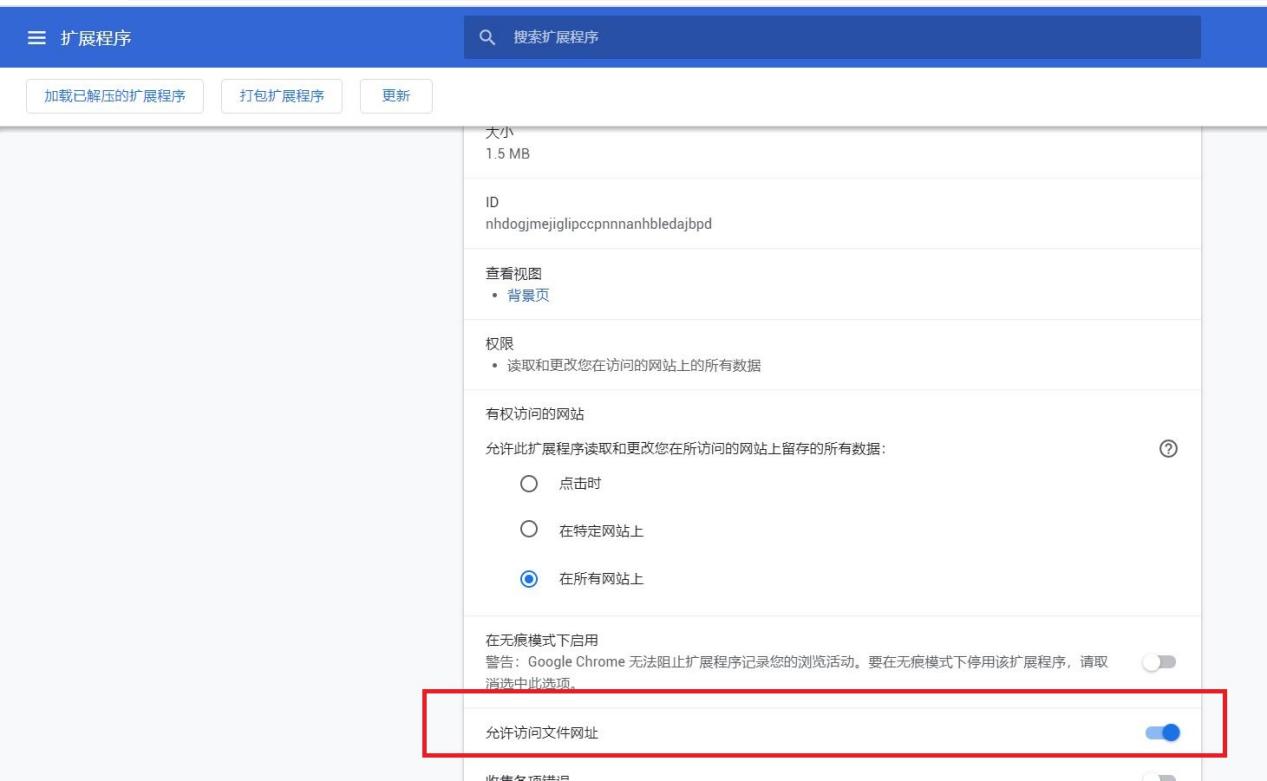
要允许访问文件网址，才能在控制台查看

在写函数的时候，要考虑参数没有传递的情况下，会不会报错的问题，一般给默认值，可以采用ES6的方式加

七、Vue调试工具vue-devtools的安装步骤和使用，它是一个chrome的插件

1、Vue.js devtools ，翻墙安装方式，安装后重启就行。

https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd?hl=zh-CN



八、过滤器

1. 概念：

Vue.js 允许你自定义过滤器，可被用作一些常见的**文本格式化**。

过滤器可以用在**两个地方（其他不能用）**：mustache 插值（插值表达式：{{ xxxx }}）和 v-bind 表达式。过滤器应该被添加在 JavaScript 表达式的尾部，通过“管道符”（ | ）形式。

2、过滤器的格式

**{{ name | nameOpt }}**，在输出name值之前先调用 nameOpt 进行一下处理，并把处理的结果当做内容进行渲染。nameOpt是过滤器的名称。原来的数据并不会被修改

3、过滤器的定义语法

Vue.filter(‘过滤器的名称’, function (data) {....});

过滤器中的function，第一个参数已经被规定死了，永远都是过滤器-->管道符前面-->传递过来的数据。

疑问：如果要过滤的对象是一个对象呢。

4、使用多个过滤器，先执行第一个过滤器，然后把第一个过滤器的结果，给第二个过滤器，这样子来处理。

|  |
| --- |
| <body>  <div class="box">  {{ msg **|** translate('u') **|** test }}  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <script type="text/javascript">  Vue.filter('translate', function (data, str) {  var result = data.replace(/\w/g, str);  return result;  });  Vue.filter('test', function (data) {  return data + 'test double filter go go go go go';  });  var vm = new Vue({  el: '.box',  data: {  msg: 'hello 我是Tom，我喜欢Jerry',  },  methods: {    },  });  </script>  </body> |

5、全局过滤器，就是所有的 VM 的实例都能进行共享的。

|  |
| --- |
| Vue.**filter**('dateFormate', function (data, format = 'yyyy-mm-dd hh:mm:ss') {  ...................  });  filter是不带s的，它实现了一个过滤器功能。 |

6、自定义一个私有的过滤器（局部）

过滤器调用的时候，如果私有过滤器和全局过滤器名称一致，这时候，优先调用私有过滤器。

它是在Vue构造函数的属性 fliters，它可以定义多个过滤器

|  |
| --- |
| var vm = new Vue({  el: '.box',  data: {...},  methods: {...},  // 私有过滤器，过滤器的名称和处理函数  filters: {  dateFormate: function (data, format = 'yyyy-mm-dd hh:mm:ss') {  ...............  },  dateFormate1: function (data, format = 'yyyy-mm-dd hh:mm:ss') {  ...............  },  }  }); |

7、ES6中字符串新方法，头部填充String.prototype.padStart(maxlength, fillstring = ‘ ’)或者尾部填充String.prototype.padEnd(maxlength, fillstring = ‘ ’)，来填充字符串。

|  |
| --- |
| var h = date.getHours().toString().padStart(2, '0');  var m = date.getMinutes().toString().padStart(2, '0');  var s = date.getSeconds().toString().padStart(2, '0'); |

1. 键盘修饰符以及自定义键盘修饰符

在一些网站输入完信息之后，直接回车就可以提交信息了，这个回车，是输入框的一个键盘事件。

如果只是绑定键盘事件，那么在输入信息的时候，也会触发，这个时候，就需要键盘修饰符，来指定你敲了回车键才触发。

1、键盘修饰符

为了在必要的情况下支持旧浏览器，Vue 提供了绝大多数常用的按键码的别名：

|  |
| --- |
| .enter  .tab  .delete (捕获“删除”和“退格”键)  .esc  .space  .up  .down  .left  .right |

2、自定义全局键盘修饰符

（1）通过**Vue.config.keyCodes.**名称 = 按键值。来自定义按键修饰符的别名：

|  |
| --- |
| **Vue.config.keyCodes.f2** = 113; |

（2）使用自定义的按键修饰符：

|  |
| --- |
| <input type="text" class="form-control" v-model="name" **@keyup.f2="add"**> |

3、系统修饰键（来自vue文档，还有其他的组合按键，可以看文档，这里就不复制了）

|  |
| --- |
| 2.1.0 新增  可以用如下修饰符来实现仅在按下相应按键时才触发鼠标或键盘事件的监听器。  **.ctrl**  **.alt**  **.shift**  **.meta**  注意：在 Mac 系统键盘上，meta 对应 command 键 (⌘)。在 Windows 系统键盘 meta 对应 Windows 徽标键 (⊞)。在 Sun 操作系统键盘上，meta 对应实心宝石键 (◆)。在其他特定键盘上，尤其在 MIT 和 Lisp 机器的键盘、以及其后继产品，比如 Knight 键盘、space-cadet 键盘，meta 被标记为“META”。在 Symbolics 键盘上，meta 被标记为“META”或者“Meta”。 |
| 例如：  <!-- Alt + C -->  <input v-on:keyup.alt.67="clear">  <!-- Ctrl + Click -->  <div v-on:click.ctrl="doSomething">Do something</div>  请注意修饰键与常规按键不同，在和 keyup 事件一起用时，事件触发时修饰键必须处于按下状态。换句话说，只有在按住 ctrl 的情况下释放其它按键，才能触发 keyup.ctrl。而单单释放 ctrl 也不会触发事件。如果你想要这样的行为，请为 ctrl 换用 keyCode：keyup.17。 |

4、js 里面的键盘事件对应的键码--网址

http://www.cnblogs.com/wuhua1/p/6686237.html

http://www.cnblogs.com/kidney/p/6052935.html

注意：Vue中所有的指令，在调用的时候，都以 v- 开头

十、自定义指令

**涉及一些组件，跟钩子函数的知识？？？**

1、使用 Vue.directive()定义全局指令

参数1：指令的名称，在定义的时候，指令的名称前面，**不需要加 v- 前缀**，在调用的时候，必须在指令名称前加上 v- 前缀来进行调用。

参数2：是一个对象，这个对象身上，有一些指令相关的函数，这些函数可以在特定的阶段，执行相关的操作。

**和样式相关的可以在bind中，和行为相关的可以在inserted中。只要通过指令绑定给了元素，不管这个元素有没有被插入到页面中去，这个元素肯定有了一个内联的样式**

**将来元素肯定会显示到页面中，这时候，浏览器的渲染引擎必然会解析样式，应用给这个元素。**

|  |
| --- |
| // 自定义全局指令  // 在每个函数中，第一个参数永远是 el ，表示被绑定了指令的那个元素，el 参数是一个原生的js对象，那我们就可以使用DOM对象的那些方法和属性了  Vue.directive('focus', {  // 每当指令绑定到元素上的时候，会立即执行这个bind函数，只执行一次。  // 在元素绑定指令的时候，还没有插入到DOM中，这时候，调用focus，方法没有作用，因为元素只有插入DOM之后，才能获取焦点  bind: function (el) {  // el.focus();  },  // inserted 表示元素插入到DOM中的时候，会执行 inserted 函数，只触发一次  inserted: function (el) {  el.focus();  },  // 当VNode(理解成节点值，Vue 编译生成的虚拟节点)更新的时候，会执行updated，可能会触发多次。  updated: function () {  }  }); |

在bind发生的时候，我们的元素还没有加入到DOM树中，这时候还在内存中，获取焦点是获取不到的，还没有进行渲染

2、自定义私有指令（局部指令）

跟filters的定义类似，指令名称：处理函数

|  |
| --- |
| <label for="exampleInputName2" v-color=**"'green'"**>搜索名称关键字:</label>  它加上单引号，表示是一个字符串。不加话，它会去找data里边的green属性，是找不到的。  var vm = new Vue({  el: '.box',  data: {....},  methods: {....},  // 私有过滤器  filters: {....},  // 私有自定义指令  directives: {  color : {  bind: function (el, binding) {  el.style.color = binding.value;  },  },  }, |

3、钩子函数参数

指令钩子函数会被传入以下参数：

|  |
| --- |
| el：指令所绑定的元素，可以用来直接操作 DOM。  binding：一个对象，包含以下 property：  name：指令名，不包括 v- 前缀。  value：指令的绑定值，例如：v-my-directive="1 + 1" 中，绑定值为 2。  oldValue：指令绑定的前一个值，仅在 update 和 componentUpdated 钩子中可用。无论值是否改变都可用。  expression：字符串形式的指令表达式。例如 v-my-directive="1 + 1" 中，表达式为 "1 + 1"。  arg：传给指令的参数，可选。例如 v-my-directive:foo 中，参数为 "foo"。  modifiers：一个包含修饰符的对象。例如：v-my-directive.foo.bar 中，修饰符对象为 { foo: true, bar: true }。  vnode：Vue 编译生成的虚拟节点。移步 VNode API 来了解更多详情。  oldVnode：上一个虚拟节点，仅在 update 和 componentUpdated 钩子中可用。 |

除了 el 之外，其它参数都应该是只读的，切勿进行修改。如果需要在钩子之间共享数据，建议通过元素的 dataset 来进行。

此外还有动态指令参数，通过vue文档查看。

十一、vue实例的生命周期

1、什么是生命周期

从Vue实例创建、运行、到销毁期间，它会创建各种各样的事件，这些事件，统称为生命周期！说白了就是一个事件。

每个 Vue 实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫做****生命周期钩子****的函数，这给了用户在不同阶段添加自己的代码的机会。

1. 生命周期钩子：就是生命周期**事件**的别名而已。等价于生命周期函数。

2、主要的生命周期函数分类

（1）创建期间的生命周期函数：

beforeCreate：实例刚在内存中被创建出来，此时，还没有初始化好 data 和 methods 属性。

created：实例已经在内存中创建OK，此时 data 和 methods 已经创建OK，此时还没有开始 编译模板。

beforeMount：此时已经完成了模板的编译，但是还没有挂载到页面中。

mounted：此时，已经将编译好的模板，挂载到了页面指定的容器中显示。

（2）运行期间的生命周期函数：

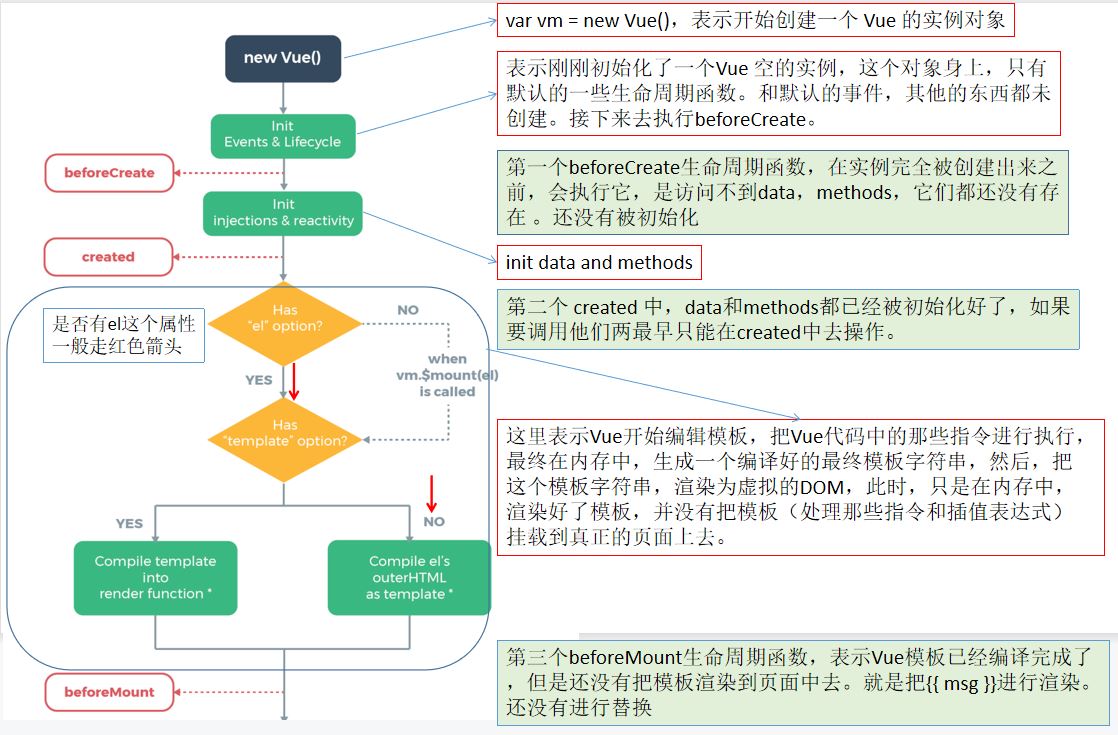
beforeUpdate：状态更新之前执行此函数， 此时 data 中的状态值是最新的，但是界面上显示的 数据还是旧的，因为此时还没有开始重新渲染DOM节点。

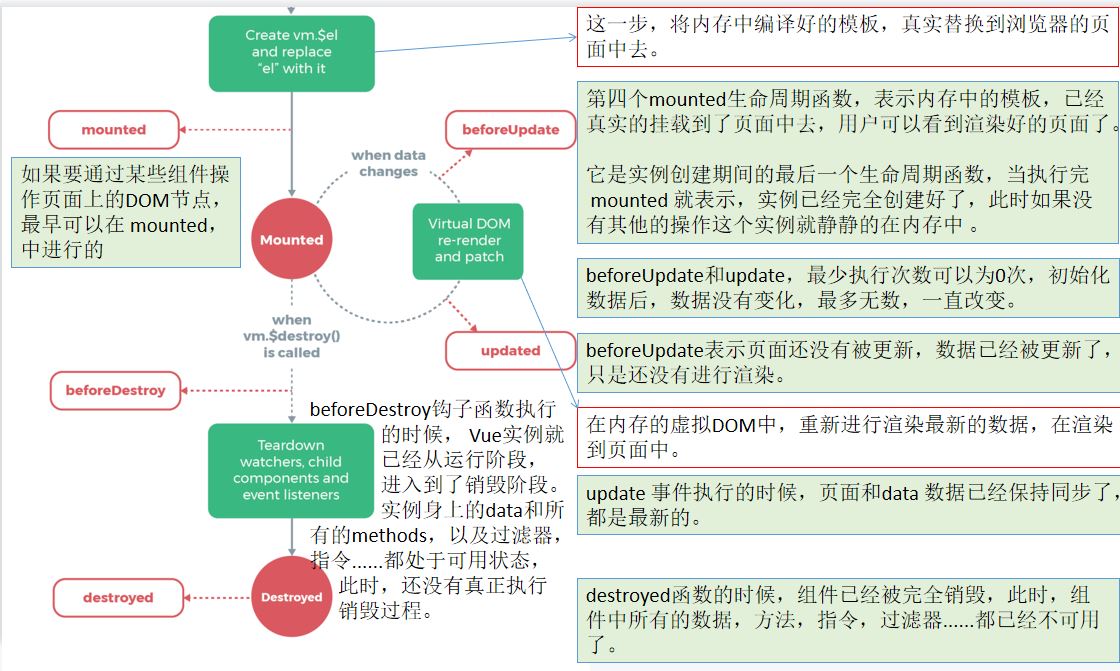
updated：实例更新完毕之后调用此函数，此时 data 中的状态值 和 界面上显示的数据，都已经完成了更新，界面已经被重新渲染好了！

（3）销毁期间的生命周期函数：

beforeDestroy：实例销毁之前调用。在这一步，实例仍然完全可用。

destroyed：Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。





|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Vue study</title>  <style type="text/css">  [v-cloak] {  display: none;  }  </style>  </head>  <body>  <div class="box" v-cloak>  <input type="button" value="改变内容" @click='change'>  {{ msg }}  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <script type="text/javascript">  var vm = new Vue({  el: '.box',  data: {  msg: 'I am Vue',  },  methods: {  change: function () {  this.msg = 'click change';  }  },  **beforeCreate**: function () {  console.log('beforeCreate', this.msg);  },  **created**: function () {  console.log('created' ,this.msg);  },  **beforeMount**: function () {  var box = document.querySelector('.box');  console.log('beforeMount: box.innerText' ,box.innerText);  },  **mounted**: function () {  var box = document.querySelector('.box');  console.log('mounted: box.innerText' ,box.innerText);  },  **beforeUpdate**: function () {  console.log('beforeUpdate msg = ', this.msg);  var box = document.querySelector('.box');  console.log('beforeUpdate: box.innerText' ,box.innerText);  },  **updated**: function () {  console.log('updated msg = ', this.msg);  var box = document.querySelector('.box');  console.log('updated: box.innerText' ,box.innerText);  }  });  </script>  </body>  </html>  1 |

十二、[vue-resource 实现 get，post，jsonp请求](https://github.com/pagekit/vue-resource)

https://github.com/pagekit/vue-resource

Vue向window挂载Vue，vue-resource向Vue的$http。它是ajax的封装。

在Vue开发中（不提倡去操作DOM），我们不需要jQuery（它主要去操作DOM），它们俩是有一些冲突的，vue-resource是官方开发出来请求。**后来推荐是用 axios 的第三方包来实现数据的请求。好像没人去维护了。**

它用的是ajax请求，可以在F12 network中看请求的类型，xhr。Ajax是原生的自带的，vue-resource只是对它的封装，不单单是jQuery有。一开始以为ajax是jQuery的。

测试的URL请求资源地址：（测试接口用wamp和php也能实现，设置一下php的响应头解决ajax跨域的问题）

get请求地址： [http://www.liulongbin.top:3005/api/getlunbo](http://vue.studyit.io/api/getlunbo)

post请求地址：http://www.liulongbin.top:3005/api/post

jsonp请求地址：http://www.liulongbin.top:3005/api/jsonp

1. get请求使用

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Vue</title>  </head>  <body>  <div class="box">  <input type="button" value="get请求" @click="getRequest">  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <!-- vue-resource是依赖于vue的，vue要先引入 -->  <script type="text/javascript" src="lib/vue-resource-1.5.3/dist/vue-resource.js"></script>  <script type="text/javascript">  var vm = new Vue({  el: '.box',  data: {  },  methods: {  getRequest: function () {  **this.$http.get**('http://www.myWebServer.com/index.php?username=lxf&password=123')**.then**(function (result) {  console.log(result);  });  },  },  });  </script>  </body>  </html> |
| 允许所有的源访问，方便测试，这样在本地也能访问到  <?php  header("Access-Control-Allow-Origin:\*");  $username = $\_GET['username'];  $password = $\_GET['password'];  // $username = $\_POST['username'];  // $password = $\_POST['password'];  $arr = array('username'=>$username, 'password'=>$password);  echo json\_encode($arr);  ?>  1  2 |

|  |
| --- |
| getRequest: function () { this.$http.get('http://www.liulongbin.top:3005/api/getlunbo').then(function (result) {  console.log(result);  }); |
| 12 |

2、\*支持promise情况下，async和await修改实现数据请求的异步方法，这样可以不要写.then方法

|  |
| --- |
| // 加上async，它从普通的方法修饰为 异步的方法，就可以使用await。再加上解构赋值。  async getRequest() {  const {body, data} = await this.$http.get('http://www.liulongbin.top:3005/api/getlunbo');  console.log(body);  console.log(data);  },  1 |

3、post请求

手动发起的 post 请求，默认没有表单格式，有的服务器可能处理不了，**request body是请求体**的意思。

post都是表单类型的提交，application/x-www-form-urlencoded，提交格式。

|  |
| --- |
| async postRequest() {  const {body} = await this.$http.post('http://www.myWebServer.com/index.php', {username: 'lxf', password: '123'}, **{emulateJSON: true}**);  console.log(body);  }  如果没有提交格式，那么它会出错。  1 |
| async postRequest() {  const {body} = **await this.$http.post**('http://www.liulongbin.top:3005/api/post', **{username: 'lxf', password: '123'}**, {**emulateJSON**: true});  console.log(body);  }  1 |

4、jsonp请求

|  |
| --- |
| async jsonpRequest() {  const { body } = await this.$http.**jsonp**('http://www.liulongbin.top:3005/api/jsonp');  console.log(body);  }  1 |

5、axios请求

支持promise语法，它和vue并没有强制的关系，不用指定引入顺序，但是它不支持发送jsonp的请求，它的数据格式跟vue-resource有点区别

|  |
| --- |
| getRequest: function () {  **axios**.get('http://www.liulongbin.top:3005/api/getlunbo').then(function (result) {  console.log(result);  });  }, |
| 1 |

（1）axios可以挂载到Vue的构造函数的原型上，这样能跟vue-resource无缝衔接

|  |
| --- |
| async postRequest() {  var formData = new FormData();  formData.append('username', 'lxf');  formData.append('password', '123');  // 注意这个不能在全局设置，它会影响get。  const {data} = await this.$http.post('http://www.myWebServer.com/index.php',**formData**,{  headers: {  'Content-Type': 'application/x-www-form-urlencoded',  }  });  console.log(data);  },  1  如果数据放在对象中，它是以对象的形式直接传过去  const {data} = await this.$http.post('http://www.myWebServer.com/index.php', {username: 'lxf', password: '123'},{  headers: {  'Content-Type': 'application/x-www-form-urlencoded',  }  });  1  测试使用的是php做的后台接口  <?php  header("Access-Control-Allow-Origin:\*");  $username = $\_POST['username'];  $password = $\_POST['password'];  $arr = array('username'=>$username, 'password'=>$password);  echo json\_encode($arr);  ?> |

6、jsonp的实现原理

由于浏览器的安全性限制，不允许AJAX访问 协议不同、域名不同、端口号不同的 数据接口，浏览器认为这种访问不安全；

可以通过动态创建script标签的形式，把script标签的src属性，指向数据接口的地址，因为script标签不存在跨域限制，这种数据获取方式，称作JSONP（注意：根据JSONP的实现原理，知晓，JSONP只支持Get请求）；

（1）具体实现过程：

先在客户端定义一个回调方法，预定义对数据的操作；

再把这个回调方法的名称，通过URL传参的形式，提交到服务器的数据接口；

服务器数据接口组织好要发送给客户端的数据，再拿着客户端传递过来的回调方法名称，拼接出一个调用这个方法的字符串，发送给客户端去解析执行；

客户端拿到服务器返回的字符串之后，当作Script脚本去解析执行，这样就能够拿到JSONP的数据

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Jsonp模仿</title>  </head>  <body>  <script type="text/javascript">  function success(data) {  console.log(data, typeof data);  }  </script>  <script type="text/javascript" **src="http://192.168.43.153:3000/?callback=success"**></script>  </body>  </html> |
| Node.js服务器下  const http = require('http');  http.createServer((request, response) => {  const {searchParams} = new URL(request.url, `http://${request.headers.host}`);  const callback = searchParams.get('callback');  console.log(callback);  let people = {  name:'Tom',  age: '43',  };  response.writeHead(200, {  'content-Type': 'text/plain; charset=utf8',  });  **const data = `${callback}(${JSON.stringify(people)})`;**  console.log(data);  response.end(data);  }).listen(3000,'192.168.43.153', (err)=>{  if (err) {  console.log(500);  return;  }  console.log('server running');  }); |

（2）注意点，回传的字符串

`${callback}(${JSON.stringify(people)})`结果是字符串success({"name":"Tom","age":"43"})

这是node服务器返回的结果，people对象转化为字符串之后还跟callback拼接了，它们是整体是一个字符串。 客户端获取，执行时，它实际上就是一个对象。

如果在script中直接写test(JSON.stringify(people));，那么people对象，它是一个字符串传进去，打印也是字符串

如果涉及到了跨域，最好是在服务端的Api接口只中CORS就行了

十三、品牌管理结合axios

1. input应该把数据前后的空格处理一下。

前边我们都是直接把请求的网址写死了，这样后期修改很麻烦，可以同axios提供

axios.defaults.baseURL

axios.create()方法，调用的返回值，是一个新的 axios 实例，在调用 create 函数的时候，可以初始化一些默认配置项，比如，请求的baseURL地址。

这样的话每个html页面都要写一个axios的实例，都是一样的代码，可以把这段代码提取出来，放到**js文件**中，需要的就引入。

2、接口说明

Vue.js中所有接口更改为：http://www.liulongbin.top:3005/

（1）品牌数据列表

|  |  |
| --- | --- |
| **地址** | http://www.liulongbin.top:3005/api/getprodlist |
| **作用描述** | 获取品牌数据json数据 |
| **请求方式** | Get |
| **传入api的参数** | 无 |
| **返回数据格式** | Json |
| **返回数据格式样例** |  |

（2）删除品牌数据

|  |  |
| --- | --- |
| **地址** | http://www.liulongbin.top:3005/api/delproduct/:id |
| **作用描述** | 删除品牌数据 |
| **请求方式** | Get |
| **传入api的参数** | Id：传入品牌数据id  举例：  vueapi.ittun.com/api/delproduct/8 |
| **返回数据格式** | Json |
| **返回数据格式样例** |  |

（3）添加品牌数据

|  |  |
| --- | --- |
| **地址** | http://www.liulongbin.top:3005/api/addproduct |
| **作用描述** | 添加品牌数据 |
| **请求方式** | Post |
| **传入api的参数** | name : ’品牌名称’ |
| **返回数据格式** | Json |
| **返回数据格式样例** |  |

十四、Vue中的动画与过渡

CSS3可以做的比较炫酷，而Vue中的动画、过渡是为了让客户去了解页面。能够提高用户的体验，帮助用户更好的理解页面中的功能。基本是过渡的效果，渐渐显示出来，隐藏等等。

动画与过渡看具体的使用，它们在表现形式上跟动画是类似的。以下一些描述就暂时不做区分。

1、单元素/组件的过渡，**进入和离开就是显示跟隐藏**

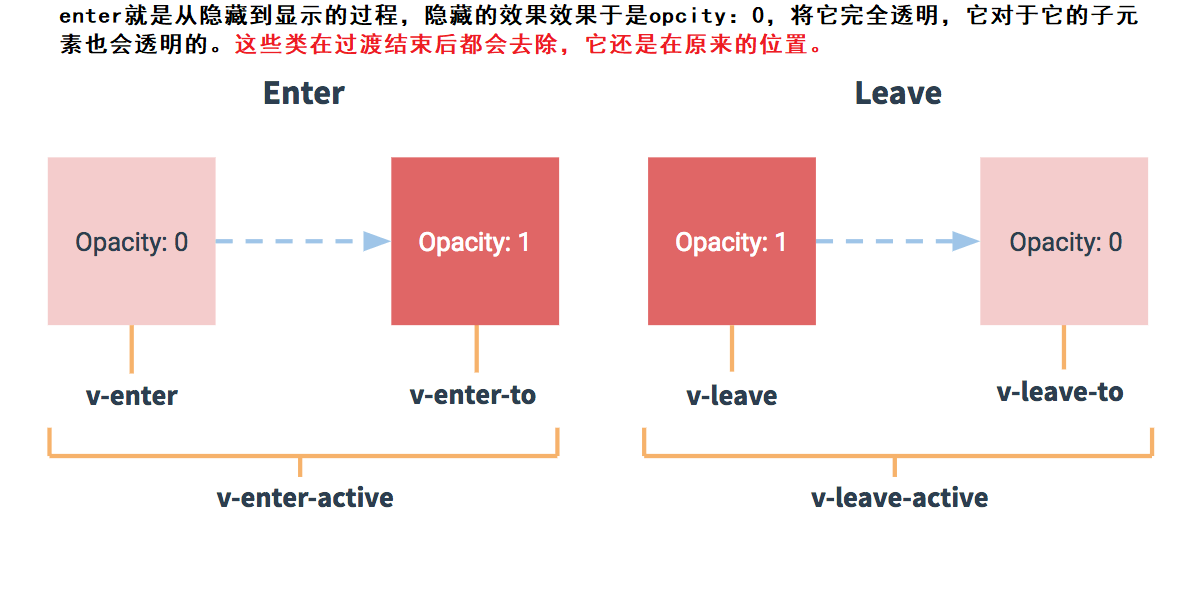
Vue 提供了 transition 的封装组件，**在下列情形中**，可以给任何元素和组件添加**进入**/**离开**过渡。过渡结束的时候，是会移除它所有的样式的。

（1）条件渲染 (使用 v-if，可以理解成增加 / 删除，它的本质是添加 / 删除dom，列表过渡时候，可以这么理解)

（2）条件展示 (使用 v-show)

（3）动态组件

（4）组件根节点



2、Vue把一个完整的过渡，拆分成两个部分（你把它隐藏了，肯定可以让它显示，所有可以分成两部分），入场（从隐藏到显示）过渡和离场（从显示到隐藏）过渡。

（1）入场过渡中，包含了两个状态，是起始状态（v-enter），结束状态（v-enter-to）。

（2）v-enter表示入场过渡的起始状态，如透明度为0，横向偏移量为50px。

（3）v-enter-to表示入场过渡的结束状态，如透明度为1，横向偏移量为 0px。

（4）v-enter-active表示入场过渡的时间段，在这里，可以规定过渡的时长，还有一些过渡相关的效果，如ease等等

v-leave和v-enter是类似的。

3、过渡的类名

在进入/离开的过渡中，会有 6 个 class 切换。

（1）进入

|  |
| --- |
| v-enter：定义进入过渡的开始状态。在元素被插入之前生效，在元素被插入之后的**下一帧移除（它的样式也就失效了）**。  v-enter-active：定义进入过渡生效时的状态。在整个进入过渡的阶段中应用，在元素被插入之前生效，在过渡/动画完成之后**移除**。这个类可以被用来定义进入过渡的过程时间，延迟和曲线函数。  v-enter-to：2.1.8 版及以上定义进入过渡的结束状态。在元素被插入之后下一帧生效 (与此同时 v-enter 被移除)，在过渡/动画完成之后**移除**。  简单理解：先v-enter，然后v-enter-active，再v-enter-to同移除v-enter，结束所有类移除。 |

（2）离开

|  |
| --- |
| v-leave：定义离开过渡的开始状态。在离开过渡被触发时立刻生效，**下一帧被移除（它的样式就失效了）**。  v-leave-active：定义离开过渡生效时的状态。在整个离开过渡的阶段中应用，在离开过渡被触发时立刻生效，在过渡/动画完成之后**移除**。这个类可以被用来定义离开过渡的过程时间，延迟和曲线函数。  v-leave-to：2.1.8 版及以上定义离开过渡的结束状态。在离开过渡被触发之后下一帧生效 (与此同时 v-leave 被删除)，在过渡/动画完成之后**移除**。 |

（3）进入和离开，之间没有任何关系。因为它们最后都被删除了，执行某个过程，才会去添加对应的类

4、过渡用法以及详细分析（重要）

（1）先用transition标签包裹需要有动画的元素

（2）设置v-enter、v-leave-to样式，跟v-enter-active

过渡知识的补充，例如原本width 200px，给一个按钮 width 300px，那么它由200px变为300px这个过程会有一个过渡

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Vue动画</title>  <style type="text/css">  .v-enter,  .v-leave-to {  opacity: 0;  transform: translateX(200px);  }  .v-enter-active,  .v-leave-active {  transition: all 3s linear;  }  </style>  </head>  <body>  <div class="box">  <input type="button" value="toggle" v-on:click="flag=!flag"><br />  <transition>  <div v-show="flag">使用Vue动画的h3元素</div>  </transition>  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <script type="text/javascript">  var vm = new Vue({  el:'.box',  data: {  flag: **true**,  },  methods: {  },  });  </script>  </body>  </html>  分析如下：  flag为true，h3显示，先分析离开。  1、离开。  首先是先加上v-leave起始状态这个类（在离开时候，一般不添加，因为起始状态就是它原来的位置，假如你设置了left，它突然跳一下，这样体验不好），这个时间非常短。**------->** 然后加上v-leave-to类和v-leave-active。-------> 然后同时把 v-leave移除了。------->接下来opacity：0慢慢透明化，以及transform:translate(100px)慢慢向左移动，-------> 最后把v-leave-to和v-leave-active移除，-------> 然后display：none。  2、进入。  首先是先加上v-enter起始状态这个类，这个时间非常短。**------->** 然后加上v-enter-to类（在进入时候，一般不设置这个类，无论你去到哪里，它最后都会被移除，回到原来的位置）和v-enter-active。-------> 然后同时把v-enter 移除了。------->接下来在初始的条件下opacity：0，transform:translate(100px)，它们已经被移除了，所以会表现为慢慢不透明化，慢慢向右移动，就是恢复原来的样子，-------> 最后把v-leave-to和v-leave-active移除，-------> 然后display：none移除。 |
| 把类移除后，会慢慢恢复原状的测试代码  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <style type="text/css">  .box {  width: 200px;  height: 200px;  background-color: red;  transition: all 3s linear;  }  .add {  transform: translateX(100px);  }  </style>  </head>  <body>  <input type="button" id="btn" value="增加transform">  <input type="button" id="btn1" value="去除trans">  <div class="box" id = 'box'></div>  <script type="text/javascript">  var btn = document.getElementById('btn');  var btn1 = document.getElementById('btn1');  var box = document.getElementById('box');  btn.onclick = function () {  box.className = 'box add';  }  btn1.onclick = function () {  box.className = 'box';  }  </script>  </body>  </html> |

5、使用第三方 CSS 动画库，需要加强学习

自定义过渡的类名

我们可以通过以下来自定义过渡类名：

|  |
| --- |
| enter-class  enter-active-class  enter-to-class (2.1.8+)  leave-class  leave-active-class  leave-to-class (2.1.8+) |

他们的优先级高于普通的类名，这对于 Vue 的过渡系统和其他第三方 CSS 动画库，如 Animate.css 结合使用十分有用。

（1）导入动画类库：

（2）定义 transition 及属性：

|  |
| --- |
| <link rel="stylesheet" href="lib/animate.css-4.1.1/animate.css">  <div class="box">  <input type="button" value="toggle" v-on:click="flag=!flag"><br />  <transition enter-active-class="animate\_\_animated animate\_\_bounceInDown" leave-active-class="animate\_\_animated animate\_\_bounceOutDown">  <div v-show="flag" class="animated">使用第三方动画的h3元素</div>  </transition>  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script> |

6、使用javascript钩子函数

商品加入购物车的动作，就是只有离开的过渡。只有半场的过渡，过渡类名和动画库都无法实现。

（1）钩子就是transition的生命周期函数，有两组，一组是进入，一组离开。

|  |
| --- |
| <transition  v-on:before-enter="beforeEnter" // 在开始过渡之前的起始状态。  v-on:enter="enter"// 过渡完成之后的结束状态。如果这些样式没有在after-enter中进行清理，那么它会一直存在。  v-on:after-enter="afterEnter" // 过渡完成之后的清理工作，相当于是回调函数，动画完成之后调用这个函数。（这个说法并不是准确的，它源码分了自动，手动区别，看下面分析）  v-on:enter-cancelled="enterCancelled" // 取消入场过渡，这个一般用不到，一般只用前边的三个。  v-on:before-leave="beforeLeave"  v-on:leave="leave"  v-on:after-leave="afterLeave"  v-on:leave-cancelled="leaveCancelled"  >  .........  </transition> |

（2）元素的transitionend事件，<transition>元素</transition>

Vue 为了知道过渡的完成，必须设置相应的事件监听器。它可以是 transitionend 或 animationend。

（3）\*钩子函数的执行时机（个人推测，有初步验证）

点击按钮flag由false变为true，当v-show条件为true的时候，先执行before-enter，初始化过渡的起始状态。再执行enter事件，设置过渡的效果，以及元素移动到哪个位置。

此时如果enter带done参数（2个参数），done（after-enter函数）的时机由用户决定；

如果enter不带done参数（只有el一个参数），那么after-enter函数会在动画结束（transitionend事件发生的时候）被Vue自动调用。

（4）enter钩子函数带done参数，与不带done参数分析，有代码

注意：那个239集视频中讲解是有误的，对done的执行时机是错的。done并不是在动画过渡结束执行的。

看源代码发现，如果形参**不指定 done** ，则表明 用户不会手动控制 动画（过渡）的结束，而转由节点的**transitionEnd控制**或者animationEnd 来标识动画（过渡）结束，开始回调 afterEnter。

具体参考源码:

var expectsCSS = css !== false && !isIE9;

var userWantsControl = getHookArgumentsLength(enterHook);

console.log('enterHook arguments length , ' + userWantsControl);

getHookArgumentsLength 用于判断 enterHook 也就是你配置的

enter(el,done) {

el.offsetWidth

el.style.transform = 'translate(200px,150px)'

el.style.transition = 'all 1s ease'

}

钩子函数的形参的个数。

当形参个数大于1，表示形参中有done, 也就是说明 用户需要手动控制动画何时结束。也就是说一旦你配置了done形参，则转由你告诉框架，动画合适结束。所以你需要在合适的时机调用done，否则 afterEnter接口就没法被调用了。

PS: 函数有几个形参，可以通过 functionName.length 得到。

|  |
| --- |
| 1、带done参数，用户需要手动控制动画何时结束。也就是说一旦你配置了done形参，则转由你告诉框架，动画合适结束。  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Vue动画钩子</title>  <link rel="stylesheet" href="lib/animate.css-4.1.1/animate.css">  <style type="text/css">  .circle {  width: 20px;  height: 20px;  background-color: red;  border-radius: 50%;  }  </style>  </head>  <body>  <div class="box">  <input type="button" value="toggle" v-on:click="flag=!flag"><br />  <transition  v-on:before-enter="beforeEnter"  v-on:enter="enter"  @after-enter="afterEnter"  >  <div v-show="flag" class="circle"></div>  </transition>  </div>  <script type="text/javascript" src="lib/vue-2.4.0.js"></script>  <script type="text/javascript">  var vm = new Vue({  el:'.box',  data: {  flag: false,  },  methods: {  beforeEnter (el) {  // el 就是transition包括的元素,就是一个原生的DOM对象  el.style.transform = 'translate(0,0)';  console.log('1beforeEnter');  },  enter(el, done) {  el.offsetLeft;  el.style.transform = 'translate(300px,300px)';  el.style.transition = 'all 3s ease';  **done();**  **// 在这里可以添加el的监听事件transitionEnd，再去执行done，但是这样不如直接不带done参数来得简单。**  console.log('2enter');  },  afterEnter(el) {  this.flag = !this.flag;  console.log('3afterEnter');  }  },  });  </script>  </body>  </html>  1  整个效果就是。  （1）点击按钮v-show：true触发显示小球。  （2）before-enter初始化小球的样式。  （3）enter加上过渡效果，确认小球最终的样式，同时调用了done，执行了after-enter，flag取反，设置为false，display会隐藏，但它是过渡的效果，到过渡结束的时候，才会马上消失。 |
| 2、不带done参数，由Vue帮我们去搞  enter(el) {  el.offsetLeft;  el.style.transform = 'translate(300px,300px)';  el.style.transition = 'all 3s ease';  console.log('2enter');  },  afterEnter(el) {  el.style.transition = 'none';  this.flag = !this.flag;  console.log('3afterEnter');  }  整体效果：  （1）3不会马上打印，会等到小球过渡结束的时候才打印。  （2）然后等待3秒后才消失，因为过渡效果并没有被处理掉，v-show为false它也需要3秒时间从显示过渡到消失。  （3）如果想马上消失，需要把过渡效果处理掉。  1 |

（5）补充事项

动画钩子函数的第一个参数：el，表示 要执行动画的那个DOM元素，是个原生的 JS DOM对象。

等价于 el 通过 document.getElementById('xxx') 方式获取到的原生JS DOM对象。

enter参数done相当于是afterEnter函数，这个done 就是afterEnter 函数的引用。

这是固定的写法，如果不写el.offsetXXX就无法实现动画效果。这个现象视频没有进行解释

可认为。写这个是为了保证Dom实时刷新，随便写什么都行，只要能保证Dom实时刷新，enter只执行了一遍，比较难解释实时刷新。

7、v-for的列表过渡

使用 v-for这种场景中，使用 <transition-group> 组件，它会以一个真实元素呈现：默认为一个 <span>（transition-group渲染成了span标签的意思）。你也可以通过 tag 属性更换为其他元素（tag = “ul”）。span里边可能是li标签，这样就不合理，span行内元素不能包括li块级元素。

过渡模式（待学习）不可用，因为我们不再相互切换特有的元素。

内部元素总是需要提供唯一的 key attribute 值。

**CSS 过渡的类**将会应用在内部的元素中，而不是这个组/容器本身。

（1）v-move

<transition-group> 组件还有一个特殊之处。不仅可以进入和离开动画，还可以改变定位。要使用这个新功能只需了解新增的 **v-move**class****，它会在元素的**改变定位**的过程中应用。像之前的类名一样，可以通过 name attribute 来自定义前缀，也可以通过 move-class attribute 手动设置。

v-move 对于设置过渡的切换时机和过渡曲线非常有用。

v-move一般与v-leave-active { position:absolute }一起使用。

内部的实现，Vue 使用了一个叫 FLIP 简单的动画队列，使用 transforms 将元素从之前的位置平滑过渡新的位置。

注意：需要注意的是使用 FLIP 过渡的元素不能设置为 display: inline 。作为替代方案，可以设置为 display: inline-block 或者放置于 flex 中。

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Vue 列表过渡</title>  <style type="text/css">  \* {  margin: 0;  padding: 0;  }  ul {  list-style: none;  margin: 100px;  }  .box li {  width: 500px;  height: 30px;  line-height: 30px;  border: 1px dashed #000;  margin: 5px;  }  .box li:hover {  background-color: yellow;  }  .v-enter,  .v-leave-to {  opacity: 0;  transform: translateX(600px);  }  .v-enter-active,  .v-leave-active {  transition: all 2s ease;  }  .v-move {  transition: all 2s ease;  }  .v-leave-active {  position: absolute;  }  </style>  </head>  <body>  <div class="box">  id: <input type="text" v-model="id" />  name: <input type="text" v-model="name" />  <input type="button" value="首行添加" @click="firstAdd">  <input type="button" value="末尾添加" @click="lastAdd">  <transition-group tag="ul">  <li v-for="(item, index) in list" v-bind:key="item.id" @click='del(index)'>{{ item.id }}----{{ item.name }}</li>  </transition-group>  </div>  <script type="text/javascript" src="./lib/vue-2.4.0.js"></script>  <script type="text/javascript">  var vm = new Vue({  el: '.box',  data: {  list: [  {id: 1, name: 'Tom'},  {id: 2, name: 'Mike'},  {id: 3, name: 'Jon'},  {id: 4, name: 'Jerry'},  ],  id: '',  name: '',  },  methods: {  del(index) {  // 删除数据，在Vue渲染时候相当于删除了DOM对象，操作了v-if，产生过渡效果  this.list.splice(index, 1);  },  firstAdd() {  var o = {  id: this.id,  name: this.name,  }  this.list.unshift(o);  },  lastAdd() {  var o = {  id: this.id,  name: this.name,  }  this.list.push(o);  }  },  });  </script>  </body>  </html> |

将要被删除的元素，在执行过渡效果的时候，它原本的位置还是占着的，直到过渡效果结束之后，它才完全被删除。

如果在v-enter-active，加了absolute，那么它一过渡马上就脱离文档流，不占据原来的位置。v-move就能渐渐移动上去。

8、transition name属性必要性

如果只使用v-，那么在html文件中有多个需transition的元素，所有的transition的效果的一样的，可以通过指定name，来指定不一样的过渡效果。

十五、定义全局Vue组件

1、组件化应用构建

组件系统是 Vue 的另一个重要概念，因为它是一种抽象，允许我们使用小型、独立和通常可复用的组件构建大型应用。仔细想想，几乎任意类型的应用界面都可以抽象为一个组件树：



在 Vue 里，一个组件本质上是一个拥有预定义选项的一个 Vue 实例。在 Vue 中注册组件很简单：

（1）\*它是一个Vue实例的理解

|  |
| --- |
| // 先得到组件的构造函数          const com = Vue.extend({              template: '<div><h2>我是div中的h2,我是一个组件</h2></div>',          });          // 全局注册组件          Vue.component('my-com', com);  1  在html中使用my-com的时候，就相当于是去调用了com这个构造函数，得到一个Vue的实例，组件的实例。  Vue官网，组件的复用有解释 |

（2）全局注册组件要在new Vue实例之前

2、组件化和模块化的不同：

（1）模块化： 是从代码逻辑的角度进行划分的；方便代码分层开发，保证每个功能模块的职能单一；为了保证大家以相同的方式去封装模块，于是我们就创造了模块化的规范（CommonJS规范）。

模块化的好处，方便项目的开发，和后期的维护与扩展，今后如果需要某些固定的功能模块，则直接拿来引用就行，提高了项目的开发效率。

（2）组件化： 是从UI界面的角度进行划分的；把页面上有重用性的UI结构和样式，单独 的抽离粗来，封装为单独的组件，前端的组件化，方便UI组件的重用；

组件化的好处：随着项目规模的发展，我们手里的组件，会越来越多。这样，我们今后一个页面中的UI，几乎都可以从手中拿现成的组件拼接出来，方便项目的开发和维护。

3、created()和mounted()补充

（1）created()表示data数据和 methods 方法都已经可用了，这时候就看可以去初始化一些数据，例如axios去请求数据，等等渲染页面。

（2）mounted()表示内存中渲染好的DOM树，已经挂载到了真实的页面中。

在网页中，有一些第三方的插件需要初始化，插件封装一般分为两部分，UI结构，JS代码。初始化插件的js代码就能先初始化。

一些第三方的插件，如果要结合Vue使用，而且，这个插件需要手动做一些js的初始化，此时，必须放到mounted，否则，可能无法出现预期的效果。

4、全局组件定义的三种方式

定义之后无论在哪个Vue的实例中都可以去使用。

（1）Vue.extend()返回的结果是组件构造函数。

（2）template属性，表示这个组件UI代码结构。

5、第一种方式Vue.extend 配合 Vue.component （麻烦点）

（1）先调用 Vue.extend()得到组件的构造函数。

1. 通过Vue.component(‘组件的名称’, 组件的构造函数)，来注册全局组件。
2. 把注册好的全局组件名称，以标签的形式引入到页面中即可。

（4）注意，全局组件的命名有以下注意事项。

|  |
| --- |
| 1、组件的命名以小写为主，一般不会出问题  2、如果组件的名称以驼峰式命名，在标签引用的时候，需要在大写字母前加 -小写字母。否则出错。  <mycom></mycom> 正确的写法 <my-com></my-com>  Vue.component('myCom', com);  1  3、官方推荐小写加 - 。  <my-com></my-com>  Vue.component('my-com', com); |

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta http-equiv="X-UA-Compatible" content="IE=edge">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body>      <div class="box">  **<my-com></my-com>**      </div>      <script type="text/javascript" src="./lib/vue-2.4.0.js"></script>      <script type="text/javascript">  **// 先得到组件的构造函数**  **const com = Vue.extend({**  **template: '<div><h2>我是div中的h2,我是一个组件</h2></div>',**  **});**  **// 全局注册组件**  **Vue.component('my-com', com);**          var vm = new Vue({              el: '.box',              data: {              },              methods: {              },          });      </script>  </body>  </html>  2 |

6、第二种，单Vue.component()

直接使用 Vue.component('组件名称', { 组件模板对象 })

Vue.component的第二个参数，即接收一个组件的构造函数，同时，也接收一个对象。

|  |
| --- |
| // 全局注册组件,采用**模板字符串**来写template          Vue.component('mycom2', {              template: `                  <div>                      <h2>我是第二种组件的创建，用的是模板字符串</h2>                  </div>`,          }); |

<template></template>模板标签，由vue提供，跟<transition>和<transition-group>类似

7、第三种，配合<template>标签使用

定义一个template的标签，定义组件的UI模板结构。这样VScode有智能提示，跟结构比较直观。

|  |
| --- |
| <div class="box">          <mycom3></mycom3>      </div>      <template id="com3">          <div>              <h2>hello 我外置的template</h2>          </div>      </template>  再进行全局注册          Vue.component('mycom3', {              template: '#com3',          }); |

（1）<template>的位置不能随意

如果写在 vm 实例的控制范围，那么它会被解析出来。

|  |
| --- |
| <div class="box">          <mycom3></mycom3>          <template id="com3">              <div>                  <h2>hello 我外置的template</h2>              </div>          </template>      </div>  1 |

如果放在body结束标签后，那么它无法获取到，报错。

8、Vue.component使用的注意点

template的内容不能是纯文本，组件中的DOM结构，有且只能有唯一的根元素（Root Element）来进行包裹！

|  |
| --- |
| Vue.component('mycom2', {              template: '纯文本',          });  1 |
| Vue.component('mycom2', {              template: '<h2></h2><h3></h3>',          });  1  h2，h3是两根元素了。 |
| const template = {              template: '<div><h1>hello</h1></div>',          }          Vue.component('mycom2', template);  这种写法也是可以的。 |

9、定义私有组件

|  |
| --- |
| <body>      <div class="box"></div>      <div class="box1">  **<private-com></private-com>**      </div>      <template id="templateCom">          <div>              <h2>我私有的template</h2>          </div>      </template>        <script type="text/javascript" src="./lib/vue-2.4.0.js"></script>      <script type="text/javascript">            // 全局注册组件,采用模板字符串来写template          var vm = new Vue({              el: '.box',              data: {              },              methods: {              },          });          // 控制box1的vm          var box1 = new Vue({              el: '.box1',  **components: {**  **// 这里写了个小横线，所有要带单引号**  **'private-com': {**  **template: '#templateCom',**  **},**  **},**          });      </script>  如果在box中引用会报如下的错误  1 |

十六、组件中展示数据和响应事件

1、组件数据和方法

（1）data，return数据对象

在 组件中，可以有自己的私有数据，但是，组件的 data 必须是一个 function，并且函数内部 return 一个数据对象。

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta http-equiv="X-UA-Compatible" content="IE=edge">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body>      <div class="box">          <com></com>      </div>      <template id="template">          <div>  **<h2 v-on:click="say">{{ msg }}</h2>**          </div>      </template>      <script type="text/javascript" src="./lib/vue-2.4.0.js"></script>      <script type="text/javascript">          Vue.component('com', {              template: '#template',  // 内部会自己先调用下这个函数，初始化data              data: function () {                  return {                      msg: 'hello world',                  }              },              methods: {                  say: function () {                      console.log('hello');                  },              },          });          // 全局注册组件,采用模板字符串来写template          var vm = new Vue({              el: '.box',              data: {              },              methods: {              },          });      </script>  </body>  </html> |

1. 组件的复用

<div id="components-demo">

<button-counter></button-counter>

<button-counter></button-counter>

<button-counter></button-counter>

</div>

注意当点击按钮时，每个组件都会各自独立维护它的 count。因为你每用一次组件，就会有一个它的**新实例**被创建。

|  |
| --- |
| 用extend的方式来理解这个新实例的创建过程  **<my-com></my-com>// vue编译到这里，就会去使用com构造函数创建一个组件实例，称为实例a**  **<my-com></my-com>// vue编译到这里，又会去使用com构造函数创建一个组件实例，称为实例a1**  **注意：这里的com构造函数的参数已经是被我们指定好了，就是data,methods这些。**  **// 先得到组件的构造函数**  **const com = Vue.extend({**  **template: '<div><h2>我是div中的h2,我是一个组件</h2></div>',**  **});**  **// 全局注册组件**  **Vue.component('my-com', com);**  还没有完善有问题，它的实例都指向了一个com对象，大概是这么个思路。具体明天研究 |

从更抽象的角度来说，每个组件，就相当于是一个自定义的元素。

1. 补充的使用 ：

将模板字符串，定义到script标签种：

注意：

组件中展示数据和响应事件

1. 在组件中，data需要被定义为一个方法，例如：

1. 在子组件中，如果将模板字符串，定义到了script标签中，那么，要访问子组件身上的data属性中的值，需要使用this来访问；

【重点】为什么组件中的data属性必须定义为一个方法并返回一个对象

1. 通过计数器案例演示

使用components属性定义局部子组件

1. 组件实例定义方式：

引用组件：

使用flag标识符结合v-if和v-else切换组件

1. 页面结构：

Vue实例定义：

使用:is属性来切换不同的子组件,并添加切换动画

1. 组件实例定义方式：

使用component标签，来引用组件，并通过:is属性来指定要加载的组件：

添加切换样式：

问题：

1、如果看到.then，那么代表前面的那个方法是用promise方法封装的。Promise后边再学习，它的返回值是一个promise的，这个暂时没有学到。

2、那个测试接口怎么搞通的，好像涉及到了跨域。

答：服务器设置了响应头就行。允许所有的源访问。它视频的接口也是这么处理的。

3、jsonp直接传回来匿名函数吗？？？还是说是服务器规定的。

4、它们似乎是通过http请求去处理的。

答：它用的就是ajax，它是一个封装。

1. async和await的学习 ES7的内容

6、FormData是什么样的数据

7、服务器只能传输字符串，数字吗

8、Vue动画如果用transform:translate

除了 vue-resource 之外，还可以使用 axios 的第三方包实现实现数据的请求

问题：

在对象中ES6的语法：lang(){.........}

1、为什么访问msg没有通过data，访问data却为undefined（暂时没有完全解决，看后续学习有没有遇到）

答：百度vue method访问data。

实例创建之后，可以通过 vm.$data 访问原始数据对象。Vue 实例也**代理了 data 对象上所有的属性**，因此访问 vm.a 等价于访问 vm.$data.a。

2、v-for key中的渲染问题，跟diff算法的理解，Vue它渲染的机制，目前理解不了。只能有一个简单的认识。

1. 数组的forEach、some、filter、findIndex，这些都属于数组的新方法，都会对数组中的每一项，进行遍历，执行相关的操作，但是里边有不同，待理解

还有要总结下，包括字符串的方法有哪一些，还是挺多的。

js some()方法

语法

arr.some(callback(element[, index[, array]])[, thisArg])

参数

callback

用来测试每个元素的函数，接受三个参数：

element

数组中正在处理的元素。

index 可选

数组中正在处理的元素的索引值。

array可选

some()被调用的数组。

thisArg可选

执行 callback 时使用的 this 值。

返回值

数组中有至少一个元素通过回调函数的测试就会返回true；所有元素都没有通过回调函数的测试返回值才会为false。

如果用一个空数组进行测试，在任何情况下它返回的都是false。some() 被调用时不会改变数组。

4、CDN是什么东西？？？