

黑马程序员™  
www.itheima.com

传智播客旗下  
高端IT教育品牌

# vue 基础入门



# 目录 Contents

## ◆ vue 简介

### ◆ vue 的基本使用

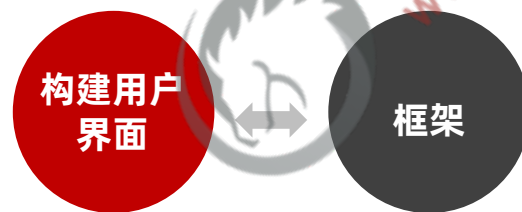
### ◆ vue 的调试工具

### ◆ vue 的指令与过滤器

### ◆ 品牌列表案例

## 1. 什么是 vue

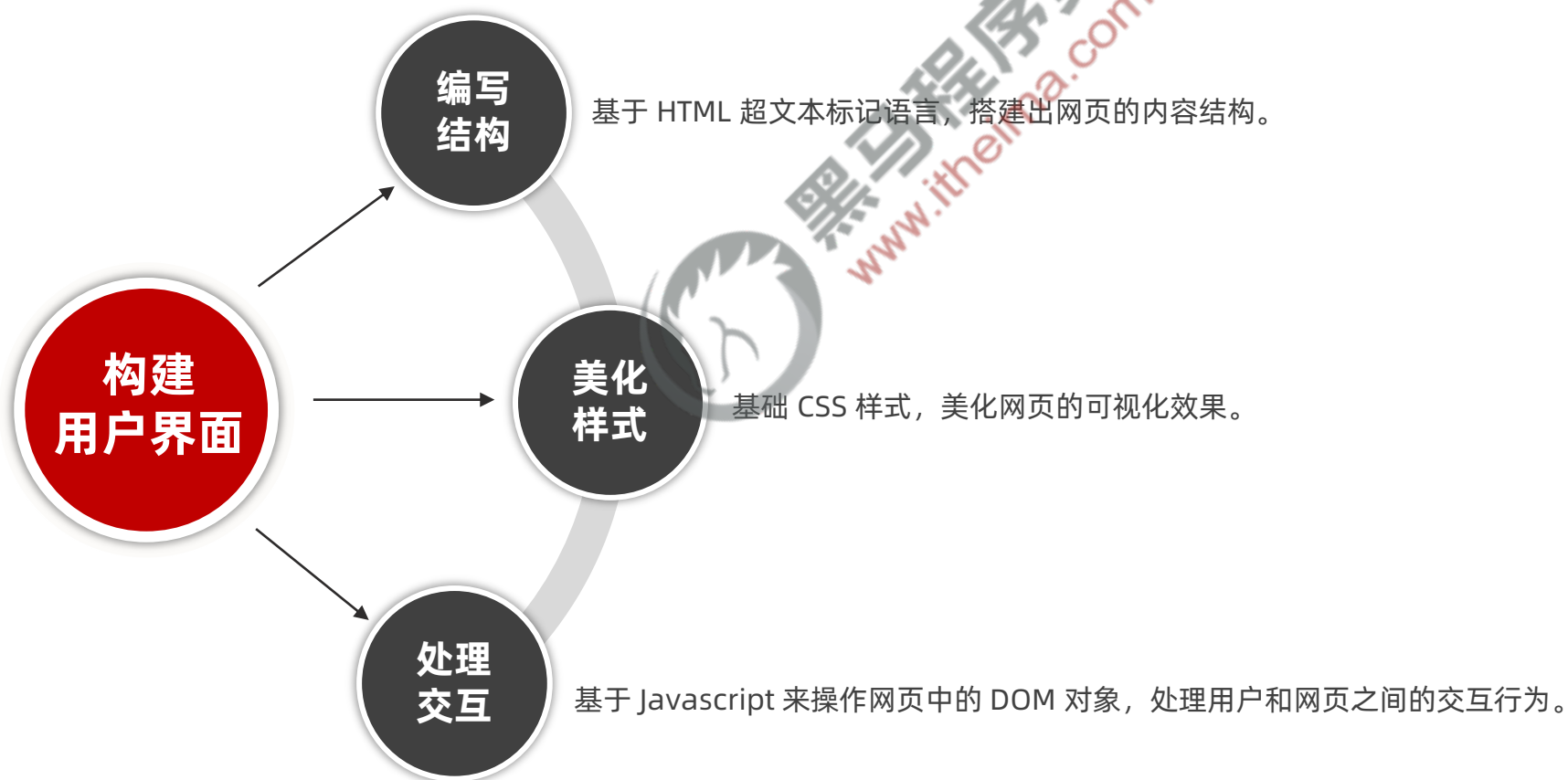
官方给出的概念：Vue (读音 /vju:/，类似于 view) 是一套用于构建用户界面的前端框架。



核心关键词

## 1.1 解读核心关键词：构建用户界面

前端开发者最主要的工作，就是为网站的使用者（又称为：网站的用户）构建出美观、舒适、好用的网页。



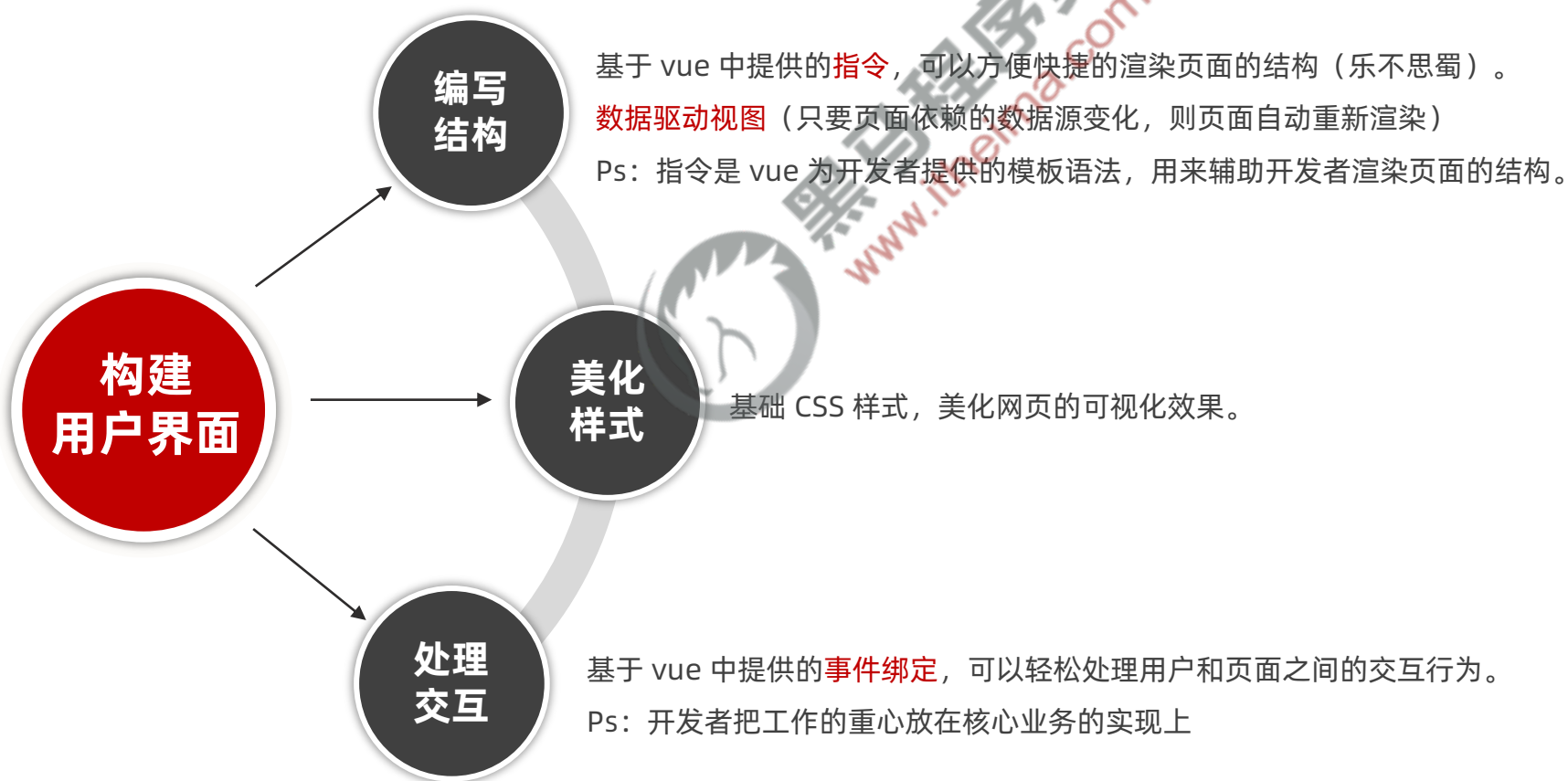
## 1.2 构建用户界面的传统方式

在传统的 Web 前端开发中，是基于 **jQuery + 模板引擎** 的方式来构建用户界面的。



## 1.3 使用 vue 构建用户界面

使用 vue 构建用户界面，解决了 **jQuery + 模板引擎** 的诸多痛点，极大的提高了前端开发的效率和体验。



## 1.4 解读核心关键词：框架

官方给 vue 的定位是前端框架，因为它提供了构建用户界面的一整套解决方案（俗称 vue 全家桶）：

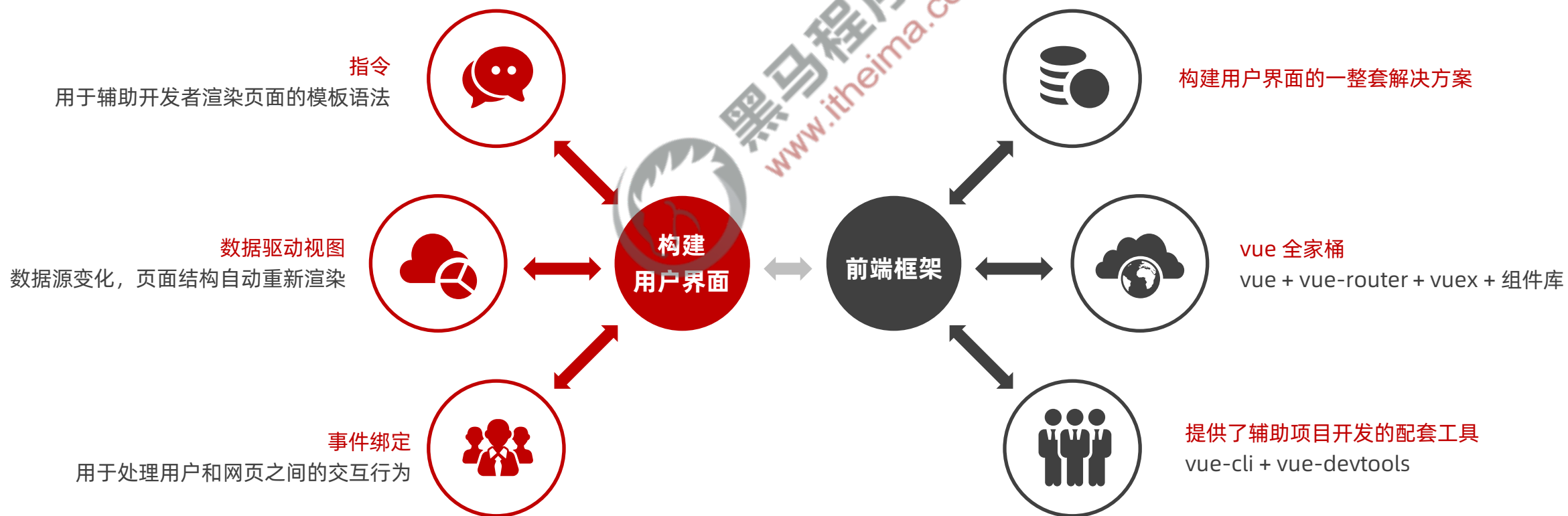
- vue（核心库）
- vue-router（路由方案）
- vuex（状态管理方案）
- vue 组件库（快速搭建页面 UI 效果的方案）

以及辅助 vue 项目开发的一系列工具：

- vue-cli（npm 全局包：一键生成工程化的 vue 项目 - 基于 webpack、大而全）
- vite（npm 全局包：一键生成工程化的 vue 项目 - 小而巧）
- vue-devtools（浏览器插件：辅助调试的工具）
- vetur（vscode 插件：提供语法高亮和智能提示）

## 1.5 总结：什么是 vue

vue 是一套用于构建用户界面的前端框架。





## 2. vue 的特性

vue 框架的特性，主要体现在如下两方面：

- ① 数据驱动视图
- ② 双向数据绑定



## 2.1 数据驱动视图

在使用了 vue 的页面中，vue 会监听数据的变化，从而自动重新渲染页面的结构。示意图如下：



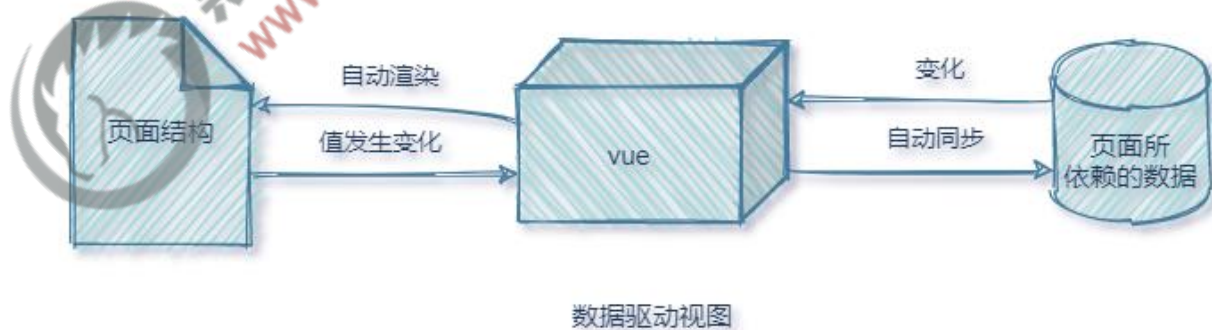
- ① 好处：当页面数据发生变化时，页面会自动重新渲染！
- ① 注意：数据驱动视图是单向的数据绑定。

## 2.2 双向数据绑定

在填写表单时，双向数据绑定可以辅助开发者在不操作 DOM 的前提下，自动把用户填写的内容同步到数据源中。示意图如下：

```
// javascript:  
var data = {  
  name: 'Bob',  
  email: 'bob@example.com'  
};
```

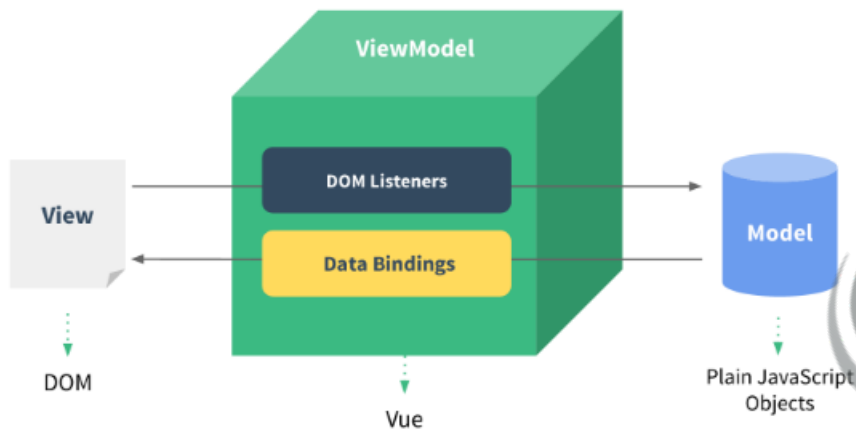
```
<!-- html -->  
<input name="name" type="text">  
<input name="email" type="text">
```



**i** 好处：开发者不再需要手动操作 DOM 元素，来获取表单元素最新的值！

## 2.3 MVVM

**MVVM** 是 vue 实现**数据驱动视图**和**双向数据绑定**的核心原理。它把每个HTML 页面都拆分成了如下三个部分：



**MVVM 示意图**

在 MVVM 概念中：

**View** 表示当前页面所渲染的 DOM 结构。

**Model** 表示当前页面渲染时所依赖的数据源。

**ViewModel** 表示 vue 的实例，它是 MVVM 的核心。

## 2.4 MVVM 的工作原理

ViewModel 作为 MVVM 的核心，是它把当前页面的数据源（Model）和页面的结构（View）连接在了一起。



当数据源发生变化时，会被 ViewModel 监听到，VM 会根据最新的数据源自动更新页面的结构

当表单元素的值发生变化时，也会被 VM 监听到，VM 会把变化过后最新的值自动同步到 Model 数据源中

## 3. vue 的版本

当前，vue 共有 3 个大版本，其中：

2.x 版本的 vue 是目前企业级项目开发中的主流版本

3.x 版本的 vue 于 2020-09-19 发布，生态还不完善，尚未在企业级项目开发中普及和推广

1.x 版本的 vue 几乎被淘汰，不再建议学习与使用

总结：

3.x 版本的 vue 是未来企业级项目开发的趋势；

2.x 版本的 vue 在未来（1 ~ 2年内）会被逐渐淘汰；

## 3.1 vue3.x 和 vue2.x 版本的对比

vue2.x 中**绝大多数的 API 与特性**，在 vue3.x 中**同样支持**。同时，vue3.x 中还**新增了 3.x 所特有的功能**、并**废弃了某些 2.x 中的旧功能**：

新增的功能例如：

**组合式 API**、多根节点组件、更好的 TypeScript 支持等

废弃的旧功能如下：

**过滤器**、不再支持 \$on, \$off 和 \$once 实例方法等

详细的变更信息，请参考官方文档给出的迁移指南：

<https://v3.vuejs.org/guide/migration/introduction.html>

# 目录 Contents

- ◆ vue 简介
- ◆ vue 的基本使用
- ◆ vue 的调试工具
- ◆ vue 的指令与过滤器
- ◆ 品牌列表案例



## 1. 基本使用步骤

- ① 导入 vue.js 的 script 脚本文件
- ② 在页面中声明一个将要被 vue 所控制的 DOM 区域
- ③ 创建 vm 实例对象（vue 实例对象）

```
<body>
  <!-- 2. 在页面中声明一个将要被 vue 所控制的 DOM 区域 -->
  <div id="app">{{username}}</div>

  <!-- 1. 导入 vue.js 的 script 脚本文件 -->
  <script src="./lib/vue-2.6.12.js"></script>
  <script>
    // 3. 创建 vm 实例对象（vue 实例对象）
    const vm = new Vue({
      // 3.1 指定当前 vm 实例要控制页面的哪个区域
      el: '#app',
      // 3.2 指定 Model 数据源
      data: {
        username: 'zs'
      }
    })
  </script>
</body>
```

## 2. 基本代码与 MVVM 的对应关系

```
1 <body>
2 <!-- 2. 在页面中声明一个将要被 vue 所控制的 DOM 区域 -->
3 <div id="app">{{username}}</div>
4
5 <!-- 1. 导入 vue.js 的 script 脚本文件 -->
6 <script src="../lib/vue-2.6.12.js"></script>
7 <script>
8   // 3. 创建 vm 实例对象 (vue 实例对象)
9   const vm = new Vue({
10     // 3.1 指定 当前 vm 实例要控制页面的哪个区域
11     el: '#app',
12     // 3.2 指定 Model 数据源
13     data: {
14       username: 'zs'
15     }
16   })
17 </script>
18 </body>
```

View 视图区域

el 指向的选择器,  
就是 View 视图区域

data 指向的对象,  
就是 Model 数据源

new Vue() 构造函数,  
得到的 vm 实例对象,  
就是 ViewModel

# 录 Contents

- ◆ vue 简介
- ◆ vue 的基本使用
- ◆ vue 的调试工具
- ◆ vue 的指令与过滤器
- ◆ 品牌列表案例

## 1. 安装 vue-devtools 调试工具

vue 官方提供的 vue-devtools 调试工具，能够方便开发者对 vue 项目进行调试与开发。

Chrome 浏览器在线安装 vue-devtools

vue 2.x 调试工具：


<https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd>

vue 3.x 调试工具：

<https://chrome.google.com/webstore/detail/vuejs-devtools/ljjemllljcmogpfapbkkighbhppjdbg>

注意：vue2 和 vue3 的浏览器调试工具不能交叉使用！

## 2. 配置 Chrome 浏览器中的 vue-devtools

点击 Chrome 浏览器右上角的  按钮，选择更多工具 -> 扩展程序 -> Vue.js devtools 详细信息，并勾选如下  
的两个选项：



权限

- 读取和更改您在访问的网站上的所有数据

有权访问的网站

允许此扩展程序读取和更改您在所访问的网站上留存的所有数据：

☐ 点击时

☐ 在特定网站上

☒ 在所有网站上

在无痕模式下启用

警告：Google Chrome 无法阻止扩展程序记录您的浏览活动。要在无痕模式下停用该扩展程序，请取消选中此选项。

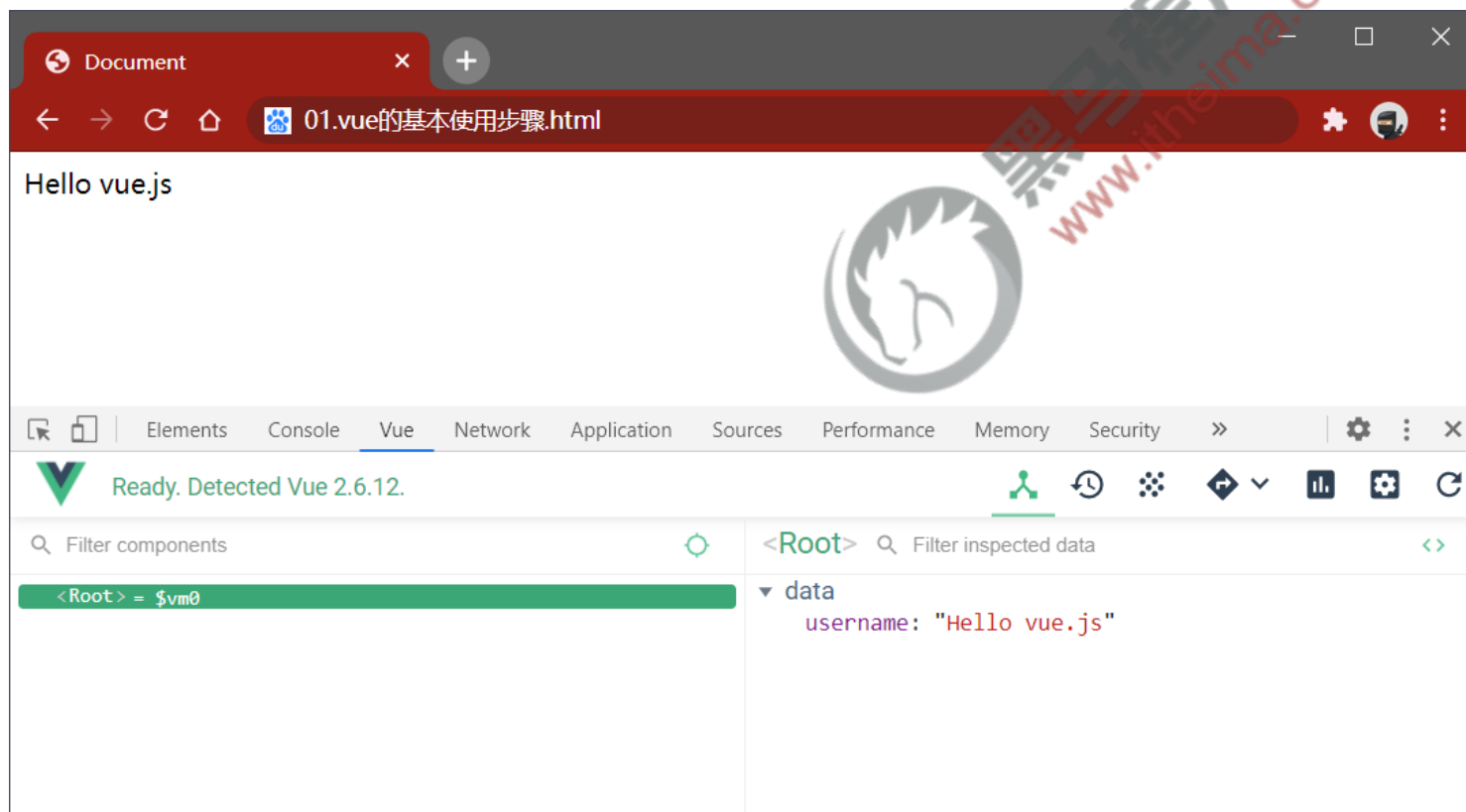
允许访问文件网址 ☒

在 Chrome 网上应用店中查看详情

⚠ 注意：修改完配置项，须重启浏览器才能生效！

## 3. 使用 vue-devtools 调试 vue 页面

在浏览器中访问一个使用了 vue 的页面，打开浏览器的开发者工具，切换到 Vue 面板，即可使用 vue-devtools 调试当前的页面。



# 目录

## Contents

- ◆ vue 简介
- ◆ vue 的基本使用
- ◆ vue 的调试工具
- ◆ vue 的指令与过滤器
- ◆ 品牌列表案例

## 1. 指令的概念

指令 (Directives) 是 vue 为开发者提供的模板语法，用于辅助开发者渲染页面的基本结构。

vue 中的指令按照不同的用途可以分为如下 6 大类：

- ① 内容渲染指令
- ② 属性绑定指令
- ③ 事件绑定指令
- ④ 双向绑定指令
- ⑤ 条件渲染指令
- ⑥ 列表渲染指令

注意：指令是 vue 开发中最基础、最常用、最简单的知识点。



## 1.1 内容渲染指令

**内容渲染指令**用来辅助开发者渲染 DOM 元素的文本内容。常用的内容渲染指令有如下 3 个：

- v-text
- {{ }}
- v-html



黑马程序员  
www.itheima.com

## v-text

用法示例：

```
1 <!-- 把 username 对应的值，渲染到第一个 p 标签中 -->
2 <p v-text="username"></p>
3
4 <!-- 把 gender 对应的值，渲染到第二个 p 标签中 -->
5 <!-- 注意：第二个 p 标签中，默认的文本“性别”会被 gender 的值覆盖掉 -->
6 <p v-text="gender">性别</p>
```

注意：v-text 指令会覆盖元素内默认的值。

## {{ }} 语法

vue 提供的 `{{ }}` 语法，专门用来解决 `v-text` 会覆盖默认文本内容的问题。这种 `{{ }}` 语法的专业名称是**插值表达式**（英文名为：**Mustache**）。

```
1 <!-- 使用 {{ }} 插值表达式，将对应的值渲染到元素的内容节点中， -->
2 <!-- 同时保留元素自身的默认值 -->
3 <p>姓名: {{username}}</p>
4 <p>性别: {{gender}}</p>
```

注意：相对于 `v-text` 指令来说，**插值表达式在开发中更常用一些**！因为它不会覆盖元素中默认的文本内容。



## v-html

**v-text** 指令和**插值表达式**只能渲染**纯文本内容**。如果要把**包含 HTML 标签的字符串**渲染为页面的 HTML 元素，则需要用到 **v-html** 这个指令：

```
1 <!-- 假设 data 中定义了名为 discription 的数据，数据的值为包含 HTML 标签的字符串： -->
2 <!-- '<h5 style="color: red;">我在黑马程序员学习 vue.js 课程.</h5>' -->
3
4 <p v-html="discription"></p>
```

最终渲染的结果为：

姓名：zs

性别：男

我在黑马程序员学习 vue.js 课程。

## 1.2 属性绑定指令

如果需要为元素的属性动态绑定属性值，则需要用到 `v-bind` 属性绑定指令。用法示例如下：

```
1 <!-- 假设有如下的 data 数据:
2 data: {
3   inputValue: '请输入内容',
4   imgSrc: 'https://cn.vuejs.org/images/logo.png'
5 }
6 -->
7
8 <!-- 使用 v-bind 指令, 为 input 的 placeholder 动态绑定属性值 -->
9 <input type="text" v-bind:placeholder="inputValue" />
10 <br />
11 <!-- 使用 v-bind 指令, 为 img 的 src 动态绑定属性值 -->
12 
```



## 属性绑定指令的简写形式

由于 **v-bind** 指令在开发中使用频率非常高，因此，vue 官方为其提供了**简写形式**（简写为英文的 **:**）。

```
1 <!-- 假设有如下的 data 数据:
2 data: {
3   inputValue: '请输入内容',
4   imgSrc: 'https://cn.vuejs.org/images/logo.png'
5 }
6 -->
7
8 <!-- 使用 v-bind 指令, 为 input 的 placeholder 动态绑定属性值 -->
9 <input type="text" :placeholder="inputValue" />
10 <br />
11 <!-- 使用 v-bind 指令, 为 img 的 src 动态绑定属性值 -->
12 
```

## 使用 Javascript 表达式

在 vue 提供的模板渲染语法中，除了支持绑定简单的数据值之外，还支持 Javascript 表达式的运算，例如：

```
1 {{ number + 1 }}  
2  
3 {{ ok ? 'YES' : 'NO' }}  
4  
5 {{ message.split('').reverse().join('') }}  
6  
7 <div v-bind:id="'list-' + id"></div>
```



## 1.3 事件绑定指令

vue 提供了 **v-on 事件绑定** 指令，用来辅助程序员为 DOM 元素绑定事件监听。语法格式如下：

```
1 <h3>count 的值为: {{count}}</h3>
2 <!-- 语法格式为 v-on:事件名称="事件处理函数的名称" -->
3 <button v-on:click="addCount">+1</button>
```

注意：原生 DOM 对象有 **onclick**、**oninput**、**onkeyup** 等原生事件，替换为 vue 的事件绑定形式后，分别为：**v-on:click**、**v-on:input**、**v-on:keyup**



## 1.3 事件绑定指令

通过 v-on 绑定的事件处理函数，需要在 **methods 节点** 中进行声明：

```
1 const vm = new Vue({
2   el: '#app',
3   data: { count: 0 },
4   methods: {           // v-on 绑定的事件处理函数，需要声明在 methods 节点中
5     addCount() { // 事件处理函数的名字
6       // this 表示当前 new 出来的 vm 实例对象，
7       // 通过 this 可以访问到 data 中的数据
8       this.count += 1
9     }
10  },
11 })
```



## 事件绑定的简写形式

由于 **v-on** 指令在开发中使用频率非常高，因此，vue 官方为其提供了**简写形式**（简写为英文的 **@**）。

```
1 <div id="app">
2   <h3>count 的值为: {{count}}</h3>
3
4   <!-- 完整写法 -->
5   <button v-on:click="addCount">+1</button>
6
7   <!-- 简写形式，把 v-on: 简写为 @ 符号 -->
8   <!-- 如果事件处理函数中的代码足够简单，只有一行代码，则可以简写到行内 -->
9   <button @click="count += 1">+1</button>
10 </div>
```



## 事件对象 event

在原生的 DOM 事件绑定中，可以在事件处理函数的形参处，接收事件对象 event。同理，在 **v-on** 指令（简写为 **@**）所绑定的事件处理函数中，**同样可以接收到事件对象 event**，示例代码如下：

```
1 <h3>count 的值为: {{count}}</h3>
2 <button v-on:click="addCount">+1</button>
3 // -----分割线-----
4 methods: {
5     addCount(e) { // 接收事件参数对象 event，简写为 e
6         const nowBgColor = e.target.style.backgroundColor
7         e.target.style.backgroundColor = nowBgColor === 'red' ? '' : 'red'
8         this.count += 1
9     }
10 }
```

## 绑定事件并传参

在使用 v-on 指令绑定事件时，可以使用 **()** 进行传参，示例代码如下：

```
1 <h3>count 的值为: {{count}}</h3>
2 <button @click="addNewCount(2)">+2</button>
3 // -----分割线-----
4 methods: {
5     // 在形参处用 step 接收传递过来的参数值
6     addNewCount(step) {
7         this.count += step
8     }
9 }
```

## \$event

**\$event** 是 vue 提供的特殊变量，用来表示原生的事件参数对象 event。**\$event** 可以解决事件参数对象 event 被覆盖的问题。示例用法如下：

```
1 <h3>count 的值为: {{count}}</h3>
2 <button @click="addNewCount(2, $event)">+2</button>
3 // -----分割线-----
4 methods: {
5   // 在形参处用 e 接收传递过来的原生事件参数对象 $event
6   addNewCount(step, e) {
7     const nowBgColor = e.target.style.backgroundColor
8     e.target.style.backgroundColor = nowBgColor === 'cyan' ? '' : 'cyan'
9     this.count += step
10   }
11 }
```



## 事件修饰符

在事件处理函数中调用 `preventDefault()` 或 `stopPropagation()` 是非常常见的需求。因此，vue 提供了**事件修饰符**的概念，来辅助程序员更方便的**对事件的触发进行控制**。常用的 5 个事件修饰符如下：

事件修饰符	说明
<code>.prevent</code>	阻止默认行为（例如：阻止 a 连接的跳转、阻止表单的提交等）
<code>.stop</code>	阻止事件冒泡
<code>.capture</code>	以捕获模式触发当前的事件处理函数
<code>.once</code>	绑定的事件只触发1次
<code>.self</code>	只有在 event.target 是当前元素自身时触发事件处理函数

语法格式如下：

```
1 <!-- 触发 click 点击事件时，阻止 a 链接的默认跳转行为 -->
2 <a href="https://www.baidu.com" @click.prevent="onLinkClick">百度首页</a>
```



## 按键修饰符

在监听键盘事件时，我们经常需要判断详细的按键。此时，可以为键盘相关的事件添加按键修饰符，例如：

```
1 <!-- 只有在 `key` 是 `Enter` 时调用 `vm.submit()` -->
2 <input @keyup.enter="submit">
3
4 <!-- 只有在 `key` 是 `Esc` 时调用 `vm.clearInput()` -->
5 <input @keyup.esc="clearInput">
```



## 1.4 双向绑定指令

vue 提供了 **v-model** 双向数据绑定指令，用来辅助开发者在不操作 DOM 的前提下，快速获取表单的数据。

```
1 <p>用户名是: {{username}}</p>
2 <input type="text" v-model="username" />
3
4 <p>选中的省份是: {{province}}</p>
5 <select v-model="province">
6   <option value="">请选择</option>
7   <option value="1">北京</option>
8   <option value="2">河北</option>
9   <option value="3">黑龙江</option>
10 </select>
```

注意：v-model 指令只能配合表单元素一起使用！



## v-model 指令的修饰符

为了方便对用户输入的内容进行处理，vue 为 v-model 指令提供了 3 个修饰符，分别是：

修饰符	作用	示例
.number	自动将用户的输入值转为数值类型	<input v-model.number="age" />
.trim	自动过滤用户输入的首尾空白字符	<input v-model.trim="msg" />
.lazy	在“change”时而非“input”时更新	<input v-model.lazy="msg" />

## 1.5 条件渲染指令

条件渲染指令用来辅助开发者按需控制 DOM 的显示与隐藏。条件渲染指令有如下两个，分别是：

- v-if
- v-show



黑马程序员  
www.itheima.com

## v-if 和 v-show 的区别

实现原理不同：

- v-if 指令会动态地创建或移除 DOM 元素，从而控制元素在页面上的显示与隐藏；
- v-show 指令会动态为元素添加或移除 style="display: none;" 样式，从而控制元素的显示与隐藏；

性能消耗不同：

v-if 有更高的切换开销，而 v-show 有更高的初始渲染开销。

- 如果需要非常频繁地切换，则使用 v-show 较好
- 如果在运行时条件很少改变，则使用 v-if 较好

## v-else

v-if 可以单独使用，或配合 v-else 指令一起使用：

```
1 <div v-if="Math.random() > 0.5">
2   随机数大于 0.5
3 </div>
4 <div v-else>
5   随机数小于或等于 0.5
6 </div>
```

## v-else-if

v-else-if 指令，顾名思义，充当 v-if 的“else-if 块”，可以连续使用：

```
1 <div v-if="type === 'A'">优秀</div>
2 <div v-else-if="type === 'B'">良好</div>
3 <div v-else-if="type === 'C'">一般</div>
4 <div v-else>差</div>
```

## 1.6 列表渲染指令

vue 提供了 **v-for** 指令，用来辅助开发者基于一个数组来循环渲染相似的 UI 结构。

v-for 指令需要使用 **item in items** 的特殊语法，其中：

- items 是待循环的数组
- item 是当前的循环项

```
1 data: {  
2   list: [ // 列表数据  
3     { id: 1, name: 'zs' },  
4     { id: 2, name: 'ls' }  
5   ]  
6 }  
7 // -----分割线-----  
8 <ul>  
9   <li v-for="item in list">姓名是: {{item.name}}</li>  
10 </ul>
```

## v-for 中的索引

v-for 指令还支持一个可选的第二个参数，即当前项的索引。语法格式为 `(item, index) in items`，示例代码如下：

```
1 data: {  
2   list: [ // 列表数据  
3     { id: 1, name: 'zs' },  
4     { id: 2, name: 'ls' }  
5   ]  
6 }  
7 // -----分割线-----  
8 <ul>  
9   <li v-for="(item, index) in list">索引是: {{index}}, 姓名是: {{item.name}}</li>  
10 </ul>
```

注意：v-for 指令中的 `item` 项和 `index` 索引都是形参，可以根据需要进行重命名。例如 `(user, i) in userlist`



## 使用 key 维护列表的状态

当列表的数据变化时，默认情况下，vue 会尽可能的复用已存在的 DOM 元素，从而提升渲染的性能。但这种默认的性能优化策略，会导致有状态的列表无法被正确更新。

为了给 vue 一个提示，以便它能跟踪每个节点的身份，从而在保证有状态的列表被正确更新的前提下，提升渲染的性能。此时，需要为每项提供一个唯一的 key 属性：

```
1 <!-- 用户列表区域 -->
2 <ul>
3   <!-- 加 key 属性的好处: -->
4   <!-- 1. 正确维护列表的状态 -->
5   <!-- 2. 复用现有的 DOM 元素, 提升渲染的性能 -->
6   <li v-for="user in userlist" :key="user.id">
7     <input type="checkbox" />
8     姓名: {{user.name}}
9   </li>
10 </ul>
```





## key 的注意事项

- ① key 的值只能是字符串或数字类型
- ② key 的值必须具有唯一性（即：key 的值不能重复）
- ③ 建议把数据项 id 属性的值作为 key 的值（因为 id 属性的值具有唯一性）
- ④ 使用 index 的值当作 key 的值没有任何意义（因为 index 的值不具有唯一性）
- ⑤ 建议使用 v-for 指令时一定要指定 key 的值（既提升性能、又防止列表状态紊乱）



## 2. 过滤器

过滤器（Filters）常用于文本的格式化。例如：

hello -> Hello

过滤器应该被添加在 JavaScript 表达式的尾部，由“管道符”进行调用，示例代码如下：

```
1 <!-- 在双花括号中通过“管道符”调用 capitalize 过滤器，对 message 的值进行格式化 -->
2 <p>{{ message | capitalize }}</p>
3
4 <!-- 在 v-bind 中通过“管道符”调用 formatId 过滤器，对 rawId 的值进行格式化 -->
5 <div v-bind:id="rawId | formatId"></div>
```

过滤器可以用在两个地方：插值表达式和 v-bind 属性绑定。

## 2.1 定义过滤器

在创建 vue 实例期间，可以在 **filters** 节点中定义过滤器，示例代码如下：

```
1 const vm = new Vue({
2   el: '#app',
3   data: {
4     message: 'hello vue.js',
5     info: 'title info'
6   },
7   filters: {           // 在 filters 节点下定义“过滤器”
8     capitalize(str) { // 把首字母转为大写的过滤器
9       return str.charAt(0).toUpperCase() + str.slice(1)
10    }
11  }
12 })
```



## 2. 过滤器

过滤器（Filters）是 vue 为开发者提供的功能，常用于文本的格式化。过滤器可以用在两个地方：插值表达式和 v-bind 属性绑定。

过滤器应该被添加在 JavaScript 表达式的尾部，由“管道符”进行调用，示例代码如下：

```
1 <!-- 在双花括号中通过“管道符”调用 capitalize 过滤器，对 message 的值进行格式化 -->
2 <p>{{ message | capitalize }}</p>
3
4 <!-- 在 v-bind 中通过“管道符”调用 formatId 过滤器，对 rawId 的值进行格式化 -->
5 <div v-bind:id="rawId | formatId"></div>
```



## 2.2 私有过滤器和全局过滤器

在 filters 节点下定义的过滤器，称为“私有过滤器”，因为它只能在当前 vm 实例所控制的 el 区域内使用。

如果希望在多个 vue 实例之间共享过滤器，则可以按照如下的格式定义全局过滤器：

```
1 // 全局过滤器 - 独立于每个 vm 实例之外
2 // Vue.filter() 方法接收两个参数:
3 //   第 1 个参数，是全局过滤器的“名字”
4 //   第 2 个参数，是全局过滤器的“处理函数”
5 Vue.filter('capitalize', (str) => {
6   return str.charAt(0).toUpperCase() + str.slice(1) + '~~'
7 })
```

## 2.3 连续调用多个过滤器

过滤器可以**串联地**进行调用，例如：

```
1 <!-- 把 message 的值，交给 filterA 进行处理 -->
2 <!-- 把 filterA 处理的结果，再交给 filterB 进行处理 -->
3 <!-- 最终把 filterB 处理的结果，作为最终的值渲染到页面上 -->
4 {{ message | filterA | filterB }}
```



## 2.3 连续调用多个过滤器

示例代码如下：

```
1 <!-- 串联调用多个过滤器 -->
2 <p>{{text | capitalize | maxLength}}</p>
3
4 // 全局过滤器 - 首字母大写
5 Vue.filter('capitalize', (str) => {
6   return str.charAt(0).toUpperCase() + str.slice(1) + '~~~'
7 })
8
9 // 全局过滤器 - 控制文本的最大长度
10 Vue.filter('maxLength', (str) => {
11   if (str.length <= 10) return str
12   return str.slice(0, 10) + '...'
13 })
```



## 2.4 过滤器传参

过滤器的本质是 JavaScript 函数，因此可以接收参数，格式如下：

```
1 <!-- arg1 和 arg2 是传递给 filterA 的参数 -->
2 <p>{{ message | filterA(arg1, arg2) }}</p>
3
4 // 过滤器处理函数的形参列表中：
5 //   第一个参数：永远都是“管道符”前面待处理的值
6 //   从第二个参数开始，才是调用过滤器时传递过来的 arg1 和 arg2 参数
7 Vue.filter('filterA', (msg, arg1, arg2) => {
8   // 过滤器的代码逻辑...
9 })
```



## 2.4 过滤器传参

示例代码如下：

```
1 <!-- 调用 maxLength 过滤器时传参 -->
2 <p>{{text | capitalize | maxLength(5)}}</p>
3
4 // 全局过滤器 - 首字母大写
5 Vue.filter('capitalize', (str) => {
6   return str.charAt(0).toUpperCase() + str.slice(1) + '~~'
7 })
8
9 // 全局过滤器 - 控制文本的最大长度
10 Vue.filter('maxLength', (str, len = 10) => {
11   if (str.length <= len) return str
12   return str.slice(0, len) + '...'
13 })
```



## 2.5 过滤器的兼容性

过滤器仅在 vue 2.x 和 1.x 中受支持，在 **vue 3.x** 的版本中**剔除了过滤器**相关的功能。

在企业级项目开发中：

- 如果使用的是 2.x 版本的 vue，则依然可以使用过滤器相关的功能
- 如果项目已经升级到了 3.x 版本的 vue，官方建议使用**计算属性**或**方法**代替被剔除的过滤器功能

具体的迁移指南，请参考 vue 3.x 的官方文档给出的说明：

<https://v3.vuejs.org/guide/migration/filters.html#migration-strategy>

# 录 Contents

- ◆ vue 简介
- ◆ vue 的基本使用
- ◆ vue 的调试工具
- ◆ vue 的指令与过滤器
- ◆ 品牌列表案例

## 1. 案例效果

← → ↺ 🏠 🐾 day1综合案例.html

添加品牌

品牌名称

请输入品牌名称

添加品牌

#	品牌名称	状态	创建时间	操作
1	宝马	<input checked="" type="checkbox"/> 已启用	2020-11-03 11:03:56	删除
2	奥迪	<input type="checkbox"/> 已禁用	2020-11-03 11:03:56	删除
3	奔驰	<input checked="" type="checkbox"/> 已启用	2020-11-03 11:03:56	删除



## 2. 用到的知识点

bootstrap 4.x 相关的知识点:

卡片 (Card)、表单相关 (Forms)、按钮 (Buttons)、表格 (Tables)

vue 指令与过滤器相关的知识点:

插值表达式、属性绑定、事件绑定、双向数据绑定、修饰符、条件渲染、列表渲染、全局过滤器



## 3. 整体实现步骤

- ① 创建基本的 vue 实例
- ② 基于 vue 渲染表格数据
- ③ 实现添加品牌的功能
- ④ 实现删除品牌的功能
- ⑤ 实现修改品牌状态的功能





### 3.1 创建基本的 vue 实例

步骤1：导入 vue 的 js 文件：

```
1 <!-- 1. 导入 vue 的 js 文件 -->  
2 <script src="../lib/vue-2.6.12.js"></script>
```



### 3.1 创建基本的 vue 实例

步骤2：在 <body> 标签中声明 el 区域：

```
1 <!-- 2. 创建 el 区域 -->
2 <div id="app"></div>
```





### 3.1 创建基本的 vue 实例

步骤3：创建 vue 实例对象：

```
1 // 3. 创建 vue 实例对象
2 const vm = new Vue({
3   el: '#app',
4   data: {
5     // 品牌列表数据
6     brandlist: [
7       { id: 1, brandname: '宝马', state: true, addtime: new Date() },
8       { id: 2, brandname: '奥迪', state: false, addtime: new Date() },
9       { id: 3, brandname: '奔驰', state: true, addtime: new Date() },
10    ],
11  },
12 })
```



### 3.2 基于 vue 渲染表格数据

步骤1：使用 v-for 指令循环渲染表格的数据：

```
1 <tbody>
2   <tr v-for="(item, index) in brandlist" :key="item.id">
3     <td>{{index + 1}}</td>
4     <td>{{item.brandname}}</td>
5     <td>{{item.state}}</td>
6     <td>{{item.addtime}}</td>
7     <td><a href="#">删除</a></td>
8   </tr>
9 </tbody>
```



### 3.2 基于 vue 渲染表格数据

步骤2：将品牌的状态渲染为 Switch 开关效果：

```
1 <td>
2   <div class="custom-control custom-switch">
3     <!-- 基于 checkbox 渲染出 switch 开关效果 -->
4     <input type="checkbox" class="custom-control-input" :id="item.id" v-model="item.state" />
5     <!-- switch 开关后面的描述文本 -->
6     <label class="custom-control-label" :for="item.id" v-if="item.state">已启用</label>
7     <label class="custom-control-label" :for="item.id" v-else>已禁用</label>
8   </div>
9 </td>
```

Switch 开关效果的官方文档地址：

<https://v4.bootcss.com/docs/components/forms/#switches>



### 3.2 基于 vue 渲染表格数据

步骤3：使用全局过滤器对时间进行格式化：

```
1 <!-- 调用 dateFormat 过滤器 -->
2 <td>{{item.addtime | dateFormat}}</td>
```



## 3.2 基于 vue 渲染表格数据

步骤3：使用全局过滤器对时间进行格式化：

```
1 Vue.filter('dateFormat', (dtStr) => { // 定义格式化时间的 dateFormat 全局过滤器
2   const dt = new Date(dtStr)
3
4   const y = dt.getFullYear()
5   const m = padZero(dt.getMonth() + 1)
6   const d = padZero(dt.getDate())
7   const hh = padZero(dt.getHours())
8   const mm = padZero(dt.getMinutes())
9   const ss = padZero(dt.getSeconds())
10
11   return `${y}-${m}-${d} ${hh}:${mm}:${ss}`
12 })
```



### 3.2 基于 vue 渲染表格数据

步骤3：使用全局过滤器对时间进行格式化：

```
1 // 定义补零的函数
2 function padZero(n) {
3   return n > 9 ? n : '0' + n
4 }
```



### 3.3 添加品牌

步骤1：阻止表单的默认提交行为：

```
1 <form class="form-inline" @submit.prevent>
2   <!-- 省略表单内的 DOM 结构 -->
3 </form>
```



### 3.3 添加品牌

步骤2：为 input 输入框进行 v-model 双向数据绑定：

```
1 <input type="text" class="form-control" placeholder="请输入品牌名称" v-model.trim="brandname" />
```

注意：需要在 data 数据中声明 **brandname** 属性字段。





### 3.3 添加品牌

步骤3：为“添加品牌”的 button 按钮绑定 click 事件处理函数：

```
1 <button type="submit" class="btn btn-primary mb-2" @click="addNewBrand">添加品牌</button>
```



### 3.3 添加品牌

步骤4：在 `data` 中声明 `nextId` 属性（用来记录下一个可用的 id 值），并在 `methods` 中声明 `addNewBrand` 事件处理函数：

```
1 data: {  
2   nextId: 4, // 下一个可用的 Id 值  
3 }  
4  
5 methods: {  
6   addNewBrand() { // 添加新品牌的事件处理函数  
7     if (!this.brandname) return alert('品牌名称不能为空!')  
8     this.brandlist.push({ id: this.nextId, brandname: this.brandname, state: true, addtime: new Date() })  
9     this.brandname = ''  
10    this.nextId++  
11  },  
12 }
```



### 3.3 添加品牌

步骤5：监听 input 输入框的 **keyup** 事件，通过 **.esc** 按键修饰符快速清空文本框中的内容：

```
1 <!-- 注意：如果事件处理函数的逻辑只是简单的赋值操作，则可以简写到行内 -->
2 <input type="text" class="form-control" placeholder="请输入品牌名称" v-
  model.trim="brandname" @keyup.esc="brandname = ''">
```



### 3.4 删除品牌

**步骤1：**为删除的 a 链接绑定 click 点击事件处理函数，并阻止其默认行为：

```
1 <td>
2   <!-- 通过 .prevent 事件修饰符，阻止 a 链接的默认跳转行为 -->
3   <!-- 把 item.id 作为参数，传递给 removeBrand 函数 -->
4   <a href="#" @click.prevent="removeBrand(item.id)">删除</a>
5 </td>
```



### 3.4 删除品牌

步骤2：在 methods 节点中声明 `removeBrand` 事件处理函数如下：

```
1  methods: {  
2    // 删除品牌  
3    removeBrand(id) {  
4      // 借助于数组的 filter 方法进行过滤  
5      this.brandlist = this.brandlist.filter(x => x.id !== id)  
6    },  
7  },
```



# 总结

## ① 能够知道 vue 的**基本使用步骤**

- 导入 vue.js 文件
- new Vue() 构造函数，得到 vm 实例对象
- 声明 el 和 data 数据节点
- MVVM 的对应关系

## ② 掌握 vue 中常见**指令**的基本用法

- 插值表达式、v-bind、v-on、v-if 和 v-else
- v-for 和 :key、v-model

## ③ 掌握 vue 中**过滤器**的基本用法

- 全局过滤器 Vue.filter('过滤器名称', function)
- 私有过滤器 filters 节点



黑马程序员

[www.itheima.com](http://www.itheima.com)

传智播客旗下高端IT教育品牌

