第3章

Vue.js指令

本章学习目标

通过本章的学习，能够理解 Vue.js 指令定义与分类，掌握 Vue.js 的内置指令使用方法，掌握自定义指令的注册方式，学会使用自定义指令实现相关工程项目的需要。

Web 前端开发工程师应知应会以下内容：

- 理解 Vue.js 指令的定义与分类；
- 掌握条件渲染指令的使用与注意事项；
- 掌握列表渲染指令 v-for 的多种定义方法及与 key 属性配合使用的方法；
- 掌握数据绑定的多种方式；
- 掌握事件处理指令及事件修饰符的使用方法；
- 掌握其他内置指令的作用与使用方法；
- 掌握 Vue 自定义指令定义与注册的方法。

指令(Directive)本质是模板中出现的特殊符号，让处理模板的 JavaScript 库能够知道对相关的 DOM 元素进行一些相应的处理。Vue.js 中的指令是以带前缀(v-)的 HTML 属性(Attribute)的形式出现。指令是 Vue 给 HTML 标记新增的、拓展的属性，这些属性不属于标准的 HTML 属性，只有 Vue.js 认为是有效的，能够处理它。

指令的作用是当表达式的值改变时，将其产生的连带影响，响应式(Reactive)地作用于 DOM 上，也就是双向数据绑定。指令既可以用于普通标记，也可以用于< template ></template >标记。

Vue.js 指令分为内置指令和自定义指令两种类型。Vue.js 的指令是以 v-开头的。Vue 提供的指令有 v-if、v-else、v-else-if、v-show、v-for、v-bind、v-on、v-model、v-text、v-html、v-pre、v-cloak、v-once 等。

【基础语法】

```
< element prefix - directiveId[ :argument] = "expression"></element >
<!-- 以下是示例 -- >
< p v - html = "message"></p >
< ul v - on:click = "clickHandler"></ul >
< span v - text = "msg"></span >          <!-- 等价于< span >{{msg}}</span > -- >
< p v - if = "greeting"> Hello World </p >
< p v - show = "greeting"> Hello Chu </p >
< a v - bind:href = "url">…</a >
```

【语法说明】

element 表示标记名称；prefix-directiveId 表示通用的指令格式，如 v-if，v-是前缀，if 是指令 ID；expression 表示表达式。一些指令能够接受一个 argument(参数)，在指令名称之后以冒号表示。

指令的值是表达式，指令的值和文本插值表达式的写法是一样的。

3.1 条件渲染

3.1.1 v-if/v-else/v-else-if 指令

v-if 指令用于条件性地渲染一块内容。当指令的表达式的值为 true 时，内容被渲染。

v-else 指令必须搭配 v-if 使用，需要紧跟在 v-if 或 v-else-if 后面，否则不起作用。可以用 v-else 指令给 v-if 或 v-else-if 添加一个 else 块。

v-else-if 指令充当 v-if 的 else-if 块，可以链式地使用多次，以实现 switch 语句的功能。

【基本语法】

```
<!-- 以下是 v-if、v-else 指令的示例 -->
<标记名称 v-if="flag">v-if 指令：当 flag 为真时，我被渲染！</标记名称>
<标记名称 v-else>v-else 指令：当 flag 为假时，轮到我被渲染！</标记名称>
<!-- 以下是 v-if、v-else-if、v-else 指令的示例，替代 switch 结构 -->
<标记名称 v-if="expressionA">等级为优秀</标记名称>
<标记名称 v-else-if="expressionB">等级为合格</标记名称>
<标记名称 v-else>等级为不合格</标记名称>
```

【语法说明】

v-if 指令必须赋值为逻辑值，为 true 时渲染，为 false 时不会渲染。与 v-if 指令配套使用的 v-else 指令不需要指定属性值，只需要给指定的标记添加此属性即可。条件渲染时，满足条件的标记才能被渲染出来，所以包含 v-if 和 v-else 指令的标记只能有一个被渲染。v-else-if 指令与 v-if 指令必须赋值，否则不会有效果。expressionA、expressionB 表达式的值为逻辑值。

【例 3-1】 条件渲染综合应用。代码如下，页面效果如图 3-1 和图 3-2 所示。

```
1.  <!-- vue-3-1.html -->
2.  <!DOCTYPE html>
3.  <html>
4.    <head>
5.      <meta charset="UTF-8" />
6.      <title>Vue 条件渲染指令的应用</title>
7.      <script src="../js/vue.global.js" type="text/javascript"></script>
8.    </head>
9.    <body>
10.     <h3>条件渲染综合应用</h3>
11.     <div id="app">
12.       <div>
13.         <fieldset>
14.           <legend>使用 v-if、v-else 指令综合应用</legend>
15.           <button type="button" v-on:click="change">{{flag?"隐藏":"显示"}}</button>
16.           <h3 v-if="flag">v-if 指令：当 flag 为{{flag}}时，我被渲染！</h3>
17.           <h3 v-else>v-else 指令：当 flag 为{{flag}}时，这回该轮到我被渲染！</h3>
18.           <template v-if="flag">
19.             <h4>template 模板内容</h4>
20.             <p>渐进式前端框架 Vue.js 简单易学！</p>
21.           </template>
22.         </fieldset>
23.       </div>
24.       <div>
25.         <fieldset>
26.           <legend>v-if、v-else-if、v-else 指令综合应用</legend>
27.           <label for="">政府采购评审专家考试成绩：</label>
28.           <input type="text" v-model="score" />
29.           <h3 v-if="score>=90">结果：优秀！</h3>
```

```
30.            <h3 v-else-if="score>=80">结果:合格!</h3>
31.            <h3 v-else>结果:不合格,请补考!</h3>
32.          </fieldset>
33.        </div>
34.      </div>
35.      <script type="text/javascript">
36.        const App = {
37.          data() {
38.            return {
39.              flag: true,
40.              score: 97,
41.            };
42.          },
43.          methods: {
44.            change() {
45.              this.flag = this.flag ? false : true;
46.            },
47.          },
48.        };
49.        const instance = Vue.createApp(App).mount("#app");
50.      </script>
51.    </body>
52. </html>
```

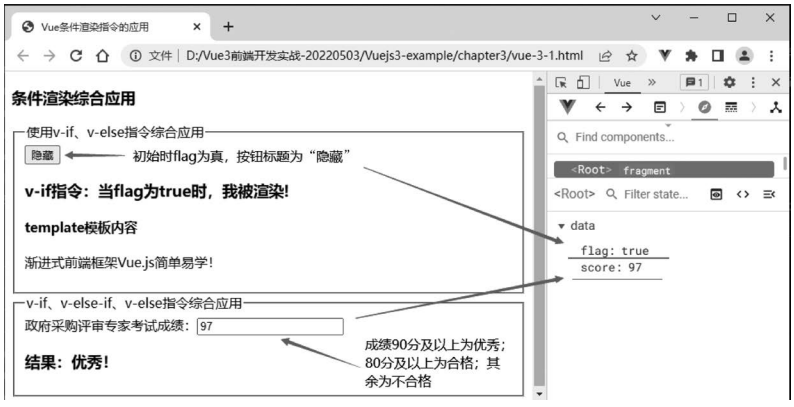

图 3-1　条件渲染综合应用初始渲染界面

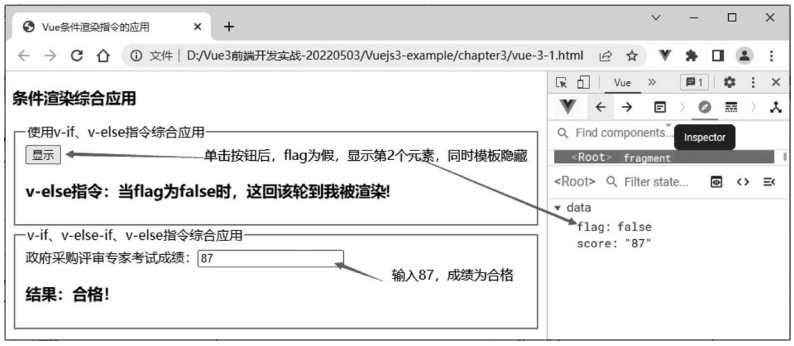

图 3-2　单击按钮和改变成绩后渲染的界面效果

上述代码中，第 15 行使用 v-on 指令绑定 click 事件，调用 change()方法，根据 flag 的值决定显示/隐藏第 16 行或第 17 行的<p>标记的内容及第 18～21 行模板的内容；第 25～32 行主要用于多分支 if-else 结构，根据第 28 行文本输入框中输入的数值，将成绩分数转换为等级（优秀、合格、不合格）的形式。

3.1.2 Vue 3.x 中 key 值的应用

key 值在 Vue 中是非常重要的，可以在 DOM Diff 中提高 DOM 的可复用性。那么，key 值在 Vue 2.x 和 Vue 3.x 中的使用有什么不同呢？

(1) 在 v-if/v-else/v-else-if 指令中，key 值绑定不再是必需的，因为 Vue 3.x 会自动生成对应的唯一 key 值。

(2) 若在 Vue 3.x 中(如 v-if)手动绑定 key 值，那么其他对应指令中(如 v-else)也必须手动绑定 key 值。

(3) < template v-for >中的 key 值应该绑定在< template >中，而不再绑定在它的子元素中。

1. v-if/v-else-if/v-else 指令中的 key 值

在 Vue 2.x 中，推荐在 v-if/v-else-if/v-else 指令中绑定 key 值，格式如下。

```
<!-- Vue 2.x -->
<div v-if="condition" key="yes">Yes</div>
<div v-else key="no">No</div>
```

但是，在 Vue 3.x 中，不再需要手动绑定 key 值，因为 Vue 3.x 会自动生成唯一的 key 值，格式如下。

```
<!-- Vue 3.x -->
<div v-if="condition">Yes</div>
<div v-else>No</div>
```

如果在 Vue 3.x 中手动绑定了 key 值，那么每个分支都必须绑定唯一的 key 值，格式如下。

```
<!-- Vue 3.x 没有手动绑定 key 值时 -->
<div v-if="condition">Yes</div>
<div v-else>No</div>
<!-- Vue 3.x 手动绑定 key 值时，每个分支均需要绑定 key 值 -->
<div v-if="condition" key="a">Yes</div>
<div v-else key="b">No</div>
```

2. < template >标记使用 v-for 中的 key 值

在 Vue 2.x 中，< template >标记不能绑定 key 值，而是绑定在它的子元素上，格式如下。

```
<!-- Vue 2.x -->
<template v-for="item in list">
  <div :key="item.id">…</div>
  <span :key="item.id">…</span>
</template>
```

在 Vue 3.x 中，key 值应该绑定在< template >标记上，而不是它的子元素上，格式如下。

```
<!-- Vue 3.x -->
<template v-for="item in list" :key="item.id">
  <div>...</div>
  <span>...</span>
</template>
```

3. < template v-for >和 v-if/v-else/v-else-if 一起使用的 key 值

在 Vue 3.x 中，如果< template >使用了 v-for 指令，并且它的子元素中使用了 v-if/v-else-if/v-else 指令，那么 key 值也需要绑定在< template >上，格式如下。

```
<!-- Vue 2.x -->
<template v-for="item in list">
```

```
    <div v-if="item.isVisible" :key="item.id">...</div>
    <span v-else :key="item.id">...</span>
  </template>
  <!-- Vue 3.x -->
  <template v-for="item in list" :key="item.id">
    <div v-if="item.isVisible">...</div>
    <span v-else>...</span>
  </template>
```

在 Vue 2.x 与 Vue 3.x 中使用 key 时，一定要注意使用场合，不能混淆，否则会报错。尤其需要注意 v-for 与 v-if/v-else-if/v-else 指令同时使用时，需要在不同的元素上使用。

【例 3-2】 使用 key 属性管理可复用的元素。代码如下，页面效果如图 3-3 和图 3-4 所示。

```
1.  <!-- vue-3-2.html -->
2.  <!DOCTYPE html>
3.  <html>
4.    <head>
5.      <meta charset="UTF-8" />
6.      <title>Vue 3.x 中 key 值的使用</title>
7.      <script src="../js/vue.global.js" type="text/javascript"></script>
8.    </head>
9.    <body>
10.     <div id="app">
11.       <h4>1.v-if/v-else</h4>
12.       <template v-if="flag">
13.         <label>用户名称:</label>
14.         <input placeholder="输入您的姓名" />
15.       </template>
16.       <template v-else>
17.         <label>地址:</label>
18.         <input placeholder="输入您的地址" />
19.       </template>
20.       <h4>2.template 上 v-for, 子元素上 v-if/v-else</h4>
21.       <label>数组元素:</label>
22.       <template v-for="item in numArr" :key="item.id">
23.         <span v-if="numArr.length!=0">{{item}}</span>
24.         <p v-else>数组为空!</p>
25.       </template>
26.       <p>
27.         <button @click="change">切换显示</button>
28.         <button @click="clear">清空数组</button>
29.       </p>
30.     </div>
31.     <script type="text/javascript">
32.       const App = {
33.         data() {
34.           return {
35.             flag: true,
36.             numArr: [100, 300, 125, 656, 345, 205],
37.           };
38.         },
39.         methods: {
40.           change() {
41.             this.flag = !this.flag;
42.           },
43.           clear() {
44.             this.numArr.splice(0);
45.           },
46.         },
47.       };
```

```
48.        const instance = Vue.createApp(App).mount("#app");
49.      </script>
50.      <style>
51.        span {margin: 2px 5px;}
52.      </style>
53.    </body>
54. </html>
```

图 3-3 未单击按钮之前的初始页面

图 3-4 单击“切换显示”和“清空数组”按钮之后的页面效果

上述代码中，第 12～19 行定义两个模板，使用 v-if 和 v-else 指令切换使用用户名称、地址界面，不需要绑定 key 值；第 22 行在模板上应用 v-for 指令，同时绑定 key 值，其内的子元素分别应用 v-if 和 v-else 指令；第 27 行和第 28 行定义两个按钮，分别绑定 change()和 clear()方法，一个用于改变 flag 的值，另一个用于清空数组元素。

3.1.3 v-show 指令

v-show 指令用于根据条件展示元素，从而实现元素的隐藏与显示。有 v-show 指令的元素始终会被渲染并保留在 DOM 中。

【基本语法】

```
<标记名称 v-show="true|false">标记的内容</标记名称>
<h1 v-show="ok">Hello!</h1>
```

【语法说明】

v-show 的值是逻辑值，为 true 时渲染到页面上；为 false 时不渲染到页面上。

带有 v-show 的元素始终会被渲染并保留在 DOM 中，v-show 只是简单地切换元素的 display CSS 属性。

注意 v-show 不支持<template></template>标记，也不支持 v-else 指令。

v-show 与 v-if 在使用上既有相同点，也有不同点。相同点是两者都是在判断 DOM 节点是否要显示。不同点如下。

(1) 实现方式不同。v-if 是根据属性值的真假判断直接从 DOM 树上删除或重建元素；v-show 只是在修改元素的 display 属性值，无论 v-show 的值为何，元素始终在 DOM 树上。

（2）编译过程不同。v-if 切换有一个局部编译/卸载的过程，切换过程中合适地销毁和重建内部的事件侦听和子组件；v-show 只是简单地基于 CSS 切换。

（3）编译条件不同。v-if 是惰性的，如果初始条件为 false，则什么也不做，只有在条件第 1 次变为 true 时才开始局部编译；v-show 是在任何条件下（首次条件是否为 true）都被编译，然后被缓存，而且 DOM 元素始终被保留。

（4）性能消耗不同。v-if 有更高的切换消耗，不适合频繁的切换；v-show 有更高的初始渲染消耗，适合频繁的切换。

【例 3-3】 v-show 指令与 v-if 指令使用比较。代码如下，页面效果如图 3-5 和图 3-6 所示。

```
<!-- vue-3-3.html -->
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>v-if与v-show指令应用区别</title>
    <script type="text/javascript" src="../js/vue.global.js"></script>
  </head>
  <body>
    <div id="app">
      <h3 v-if="flag">条件为{{flag}}时，渲染'这是v-if'</h3>
      <h3 v-show="flag">条件为{{flag}}时，渲染 h3-这是v-show</h3>
      <div v-show="flag" class="div1">div</div>
      <button type="button" @click="change">切换元素-{{flag}}</button>
    </div>
    <script type="text/javascript">
      const App = {
        data() {
          return {flag: true, };
        },
        methods: {
          change() {
            this.flag = !this.flag;      //取反赋值
          },
        },
      };
      Vue.createApp(App).mount("#app");
    </script>
    <style>
      .div1 {width: 260px; height: 100px; background: #faf8fa; }
    </style>
  </body>
</html>
```

图 3-5 未单击按钮前页面效果

图 3-6 按钮单击后页面效果

上述代码中，第 10～15 行定义一个 id 为 app 的< div >标记，其内包含一个按钮、两个< h3 >和一个< div >。第 14 行定义一个按钮，侦听 click 事件，绑定 change()方法，用于改变 flag 的值。分别使用 v-if 和 v-show 指令用于切换 3 个标记是否被渲染并显示，v-if 和 v-show 的属性值均为 flag。当 flag 值为 true 时，3 个标记都一起被渲染并显示在页面上，如图 3-5 所示。当 flag 值为 false 时，第 1 个< h3 >不会被渲染，而其他< h3 >、< div >标记一起被渲染，但由于 CSS 属性 display：none，所以不显示在页面上，如图 3-6 所示。

3.2 列表渲染(v-for 指令)

可以用 v-for 指令基于一个数组渲染一个列表。v-for 指令可以使用 item in|of items 形式的特殊语法，其中 items 是源数据数组(可以是普通数组或对象数组)，而 item 则是被迭代的数组元素的别名。v-for 指令的值中可以使用关键字 in 或 of。符号|表示或的意思。

1. 使用单一参数的 v-for 指令循环遍历对象数组

【基本语法】

```
<ul>
    <li v-for="item in|of items">{{item}}</li>
</ul>
<script type="text/javascript">
    data() {
      return{
        items: [
         {bookName: "Web 前端开发技术", isbn: '9787302488637', author: "储久良"},
         {bookName: "Vue.js 实战",isbn: '9787302484929',  author: "梁灏"},
         {bookName: "Spring Boot + Vue 全栈开发实战",  isbn: '9787302517979',author: "王松"}],
         student: {name: '储久良', class: '19 软件工程 3 班',age: 20},
         array:[100,200,300,205,110,96]
      }
    }
</script>
```

【语法说明】

定义数组类型的数据 items，数组中每个元素有 3 个属性，分别是 bookName、isbn、author，用 v-for 指令对< li >标记循环遍历渲染，如图 3-7 所示。

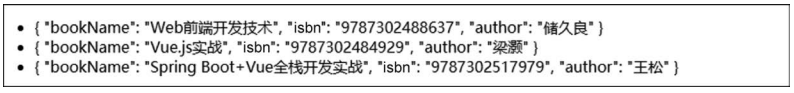

图 3-7 使用单一参数的 v-for 指令循环遍历对象数组

从图 3-7 可以看出，对象中的属性并没有分项列出，而是一次性列出。可以使用带 3 个参数的 v-for 指令的形式遍历对象数组。

2. 使用两个参数的 v-for 指令循环遍历对象数组

【基本语法】

```
<ul>
    <li v-for="(item, index) in items">
       {{index+1}}-{{item.bookName}}-{{item.issn}}-{{item.author}}
    </li>
</ul>
```

【语法说明】

v-for 属性值中支持第 2 个参数 index，作为当前项的索引。index 不可以放在第 1 个参数位置上，否则渲染后不会出结果。多个参数必须使用括号包裹，并使用逗号分隔参数。依次访问数组中对象的属性 item.bookName、item.issn、item.author，结果如图 3-8 所示。

- 1-Web前端开发技术-9787302488637-储久良
- 2-Vue.js实战-9787302484929-梁灏
- 3-Spring Boot+Vue全栈开发实战-9787302517979-王松

图 3-8　使用两个参数的 v-for 指令循环遍历对象数组

3. 使用 3 个参数的 v-for 指令循环遍历对象的属性

v-for 指令也可以对对象的属性进行遍历，使用(value,key,index) in student 的形式遍历数组元素。

【基本语法】

```
<ul>
   <li v-for="(value,key,index) in student">{{index}}-{{key}}-{{value}}</li>
</ul>
```

【语法说明】

student 是一个对象，它有 3 个属性，分别为 name、class、age。采用 v-for 指令分别遍历对象的每个属性。3 个参数的位置顺序可以任意排列，默认第 1 个参数为 value，第 2 个参数为 key，第 3 个参数为 index，结果如图 3-9 所示。

- 0-name-储久良
- 1-class-19软件工程3班
- 2-age-20

图 3-9　使用 3 个参数的 v-for 指令循环遍历对象的属性

4. v-for 指令循环遍历普通数组

【基本语法】

```
<ul>
    <li v-for="(value,index) in array">{{index}}-{{value}}</li>
</ul>
```

在 data 选项中增加定义普通数组 array，格式如下。

```
array: [100, 200, 300, 205, 110, 96],
```

【语法说明】

array 为普通数组，可以使用单一参数和带第 2 个参数(index)的 v-for 指令遍历普通数组元素，结果如图 3-10 所示。

图 3-10 v-for 指令循环遍历普通数组

5. v-for 指令循环遍历某一范围内的整数

【基本语法】

```
<div id="app">
   <span v-for="count in 20">{{count}} </span>
</div>
```

【语法说明】

使用 v-for 指令遍历某一范围内的整数，count 表示 20 以内的每个整数，然后在一行中输出所有整数，结果如图 3-11 所示。

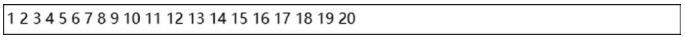

图 3-11 v-for 指令循环遍历某一范围内的整数

6. v-for 指令使用 key 关键字循环遍历数组

【基本语法】

```
<div id="app">
   <p v-for="user in students" :key='user.id'>{{user.id}} -- {{user.name}}</p>
</div>
```

在 data 选项中增加下列属性定义。

```
data(){
  return{
    students:[{id:1,name:'张开民'}],
      id:'',
      name:''
  }
},
```

【语法说明】

段落<p>标记绑定 key 属性，使段落<p>标记唯一被渲染。采用 v-for 指令遍历数组。

【例 3-4】 v-for 指令的综合应用。代码如下，页面效果如图 3-12 所示。

```
1.  <!-- vue-3-4.html -->
2.  <!DOCTYPE html>
3.  <html>
4.    <head>
5.      <meta charset="UTF-8" />
6.      <title>列表渲染 v-for 指令的应用</title>
7.      <script type="text/javascript" src="../js/vue.global.js"></script>
8.    </head>
9.    <body>
10.     <div id="app">
11.       <fieldset>
12.         <legend>数组对象遍历</legend>
13.         <ul>
14.           <li v-for="(item,index) in items">
15.             {{index+1}}-{{item.bookName}}-{{item.press}}-{{item.author}}
16.           </li>
17.         </ul>
```

```
18.       </fieldset>
19.       <fieldset>
20.         <legend>某一范围内数字遍历</legend>
21.         <span v-for="number in 20">{{number}} </span>
22.       </fieldset>
23.       <fieldset>
24.         <legend>对象的属性遍历</legend>
25.         <p>
26.           <span v-for="(value,key,index) in teacher">
27.             {{index}}-{{key}}-{{value}}
28.           </span>
29.         </p>
30.       </fieldset>
31.       <fieldset>
32.         <legend>普通数组遍历</legend>
33.         <p>
34.           <span v-for="(value,index) in array">{{index}}-{{value}}</span>
35.         </p>
36.       </fieldset>
37.       <fieldset>
38.         <legend>绑定 key 对象数组遍历</legend>
39.         <label>序号:</label>
40.         <input type="text" v-model="id" placeholder="输入序号" />
41.         <label>姓名:</label>
42.         <input type="text" v-model="name" placeholder="输入姓名" />
43.         <button type="button" v-on:click="addStudent">添加学生名单</button>
44.         <p v-for="user in students" v-bind:key="user.id">
45.           <input type="checkbox" /> {{user.id}}--{{user.name}}
46.         </p>
47.       </fieldset>
48.     </div>
49.     <script type="text/javascript">
50.       const App = {
51.         data() {
52.           return {
53.             items: [
54.               {
55.                 bookName: "计算机网络",
56.                 press: "清华社",
57.                 author: "张功萱",
58.               },
59.               {
60.                 bookName: "Vue.js 前端框架技术与实战",
61.                 press: "清华社",
62.                 author: "储久良",
63.               },
64.               {
65.                 bookName: "HTML+CSS+JavaScript 页面设计",
66.                 press: "电子社",
67.                 author: "王大松",
68.               },
69.             ],
70.             teacher: {
71.               name: "储久良",
72.               no: "12004004",
73.               age: 56,
74.             },
75.             array: [356, 324, 124, 334, 36, -234],
76.             students: [
77.               { id: 1, name: "李春进" },
```

```
78.                 { id: 2, name: "张大明" },
79.               ],
80.               id: "",
81.               name: "",
82.             };
83.           },
84.           methods: {
85.             addStudent: function() {
86.               this.students.unshift({
87.                 id: this.id,
88.                 name: this.name,
89.               });
90.             },
91.           },
92.         };
93.         Vue.createApp(App).mount("#app");
94.       </script>
95.       <style>
96.         span { margin: 0 5px;}
97.       </style>
98.     </body>
99. </html>
```

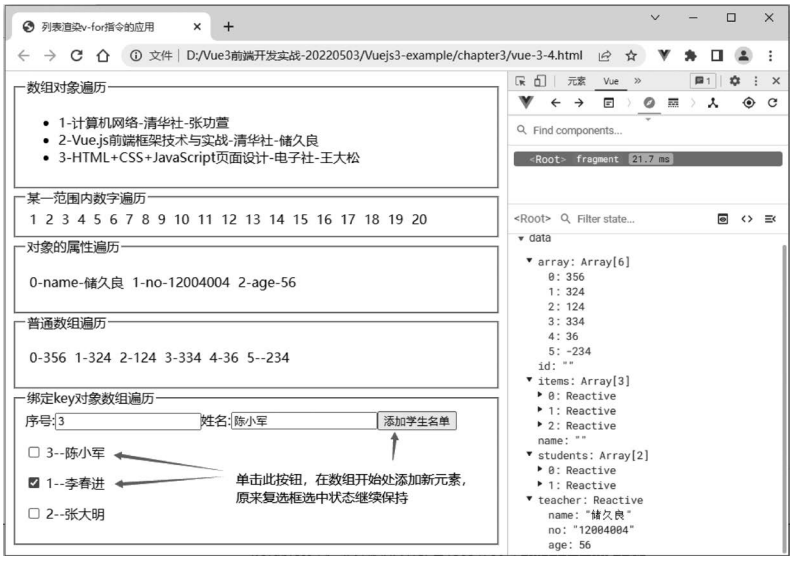

图 3-12 v-for 指令的综合应用

7. v-for 与 v-if 指令的执行优先级

当 v-for 与 v-if 指令一起使用时，即当它们处于同一节中，v-for 具有比 v-if 更高的优先级，这意味着 v-if 将分别重复运行于每个 v-for 循环中。所以，不推荐 v-for 和 v-if 指令同时使用。

针对下列两种常见的情况，建议采用的做法如下。

为了过滤一个列表中的项目(如 v-for="user in users" v-if="user.isActive")，在这种情况下，请将 users 替换为一个计算属性(如 activeUsers)，让其返回过滤后的列表。

为了避免渲染本应该被隐藏的列表(如 v-for="user in users" v-if="isDisplayUsers")，在这种情况下，请将 v-if 移动至容器元素上(如<ul>、<ol>)。

在 Vue 3.x 中，将 v-for 和 v-if 指令同时作用在同一个元素上会报错。在 Vue 2.x 中可以，但不建议这么做。在父元素<ul>上使用 v-for 指令，在子元素上使用 v-if 指令，结果如

图 3-13 所示。

```
<!-- Vue 2.x 中可以,Vue 3.x 中会报错 -->
<h3>通常做法:v-for 与 v-if 同时作用于同一元素上,显示活跃用户</h3>
<ul>
    <li v-for="user in users" :key="user.id" v-if="user.isActive===true">
        {{user.id}}:{{ user.name }} -- 状态:{{user.isActive}}
    </li>
</ul>
<!-- 以下写法是可行的 -->
<-- 建议做法:v-for 与 v-if 分别作用于不同元素上,显示活跃用户 -->
<ul v-for="user in users" :key="user.id">
   <li v-if="user.isActive===true">
        {{user.id}}:{{ user.name }} -- 状态:{{user.isActive}}
   </li>
</ul>
```

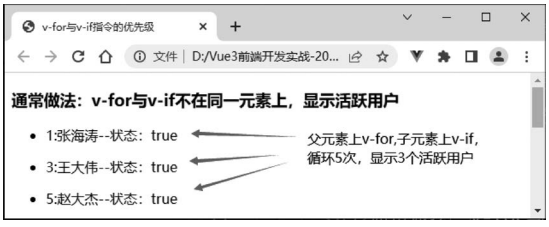

图 3-13 v-for 与 v-if 分别作用在不同元素上

【例 3-5】 v-for 与 v-if 指令配合使用的解决方案。代码如下,页面效果如图 3-14 所示。

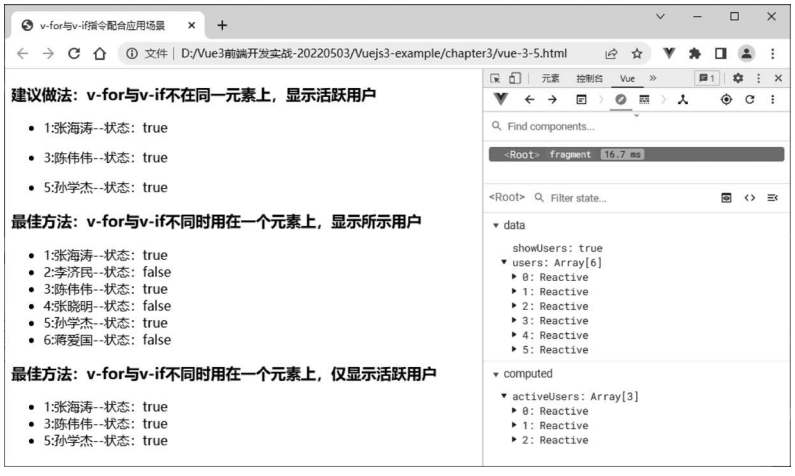

图 3-14 v-for 与 v-if 配合使用页面效果

```
1.  <!-- vue-3-5.html -->
2.  <!DOCTYPE html>
3.  <html>
4.    <head>
5.      <meta charset="UTF-8" />
6.      <title>v-for 与 v-if 指令配合应用场景</title>
7.      <script src="../js/vue.global.js" type="text/javascript"></script>
8.    </head>
9.    <body>
10.     <div id="app">
11.       <h3>建议做法:v-for 与 v-if 不在同一元素上,显示活跃用户</h3>
12.       <ul v-for="user in users" :key="user.id">
13.         <li v-if="user.isActive===true">
```

```
            {{user.id}}:{{ user.name }} -- 状态:{{user.isActive}}
          </li>
        </ul>
        <h3>最佳方法:v-for 与 v-if 不同时用在一个元素上,显示所示用户</h3>
        <ul v-if="showUsers">
          <li v-for="user in users" :key="user.id">
            {{user.id}}:{{ user.name }} -- 状态:{{user.isActive}}
          </li>
        </ul>
        <h3>最佳方法:v-for 与 v-if 不同时用在一个元素上,仅显示活跃用户</h3>
        <ul v-if="showUsers">
          <li v-for="user in activeUsers" :key="user.id">
            {{user.id}}:{{ user.name }} -- 状态:{{user.isActive}}
          </li>
        </ul>
      </div>
      <script type="text/javascript">
        const App = {
          data() {
            return {
              users: [
                { id: 1, name: "张海涛", isActive: true },
                { id: 2, name: "李济民", isActive: false },
                { id: 3, name: "陈伟伟", isActive: true },
                { id: 4, name: "张晓明", isActive: false },
                { id: 5, name: "孙学杰", isActive: true },
                { id: 6, name: "蒋爱国", isActive: false },
              ],
              showUsers: true,
            };
          },
          computed: {
            activeUsers() {
              return this.users.filter(function (user) {
                return user.isActive;
              });
            },
          },
        };
        Vue.createApp(App).mount("#app");
      </script>
    </body>
  </html>
```

3.3 类与样式绑定(v-bind 指令)

v-bind 指令用于动态地、响应地更新 HTML 属性,从而实现与 class、style 属性的绑定。只需要通过表达式计算出字符串结果即可。不过,字符串拼接麻烦且易错。因此,在将 v-bind 指令用于 class 和 style 属性时,Vue.js 做了专门的增强。表达式结果的类型除了字符串之外,还可以是对象或数组。

通常可以将 v-bind:attribute 形式缩写为:attribute 形式,将 v-on:event 形式缩写为 @event 形式,这种形式称为“语法糖”。所谓“语法糖”,是指在不影响功能的情况下,添加某种方法实现同样的效果,从而方便程序开发,因此使用语法糖可以简化代码书写。

【基本语法】

(1) HTML 中定义如下。

```
<div v-bind:class="classObject"></div>  <!-- v-bind 基本语法 -->
<div :class="classObject"></div>        <!-- v-bind 语法糖及绑定对象 -->
<div v-bind:class="{active: isActive }"></div>
```

（2）v-bind:class 指令也可以与普通的 class 属性共存。

```
<div class="static" v-bind:class="{active: isActive, 'text-danger': hasError }"></div>
<div v-bind:class="[classA, classB]"></div>          <!-- 数组语法 -->
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }">对象</div>
<div v-bind:style="[styleObjectA, styleObjectB]">style 数组</div>
//组件的 data 选项定义
data(){
  return{
    classObject: {'class-a': true, 'class-b': false},
    isActive: true,
    hasError: false,
    classA: 'class-a',
    classB: 'class-b',
    styleObjectA:{ color:'blue'},
    styleObjectB:{ background:'#DFDFDF'},
  }
}
```

【语法说明】

v-bind 可以绑定 class 属性，也可以绑定 style 属性。需要将 v-bind 绑定到特定的元素上，从而动态地渲染元素的样式。其中，与 class 属性绑定时，其值可以是变通变量、对象、数组等；与 style 属性绑定时，其值可以为对象、数组。

在实际工程中需要分 3 步实现。

（1）在 HTML 元素上完成属性绑定，并定义其值的类型。

（2）在 Vue 组件的 data 选项中定义相关属性的值。

（3）在 CSS 部分需要定义相关类的样式。

【例 3-6】 class 和 style 绑定综合应用。代码如下，页面效果如图 3-15 所示。

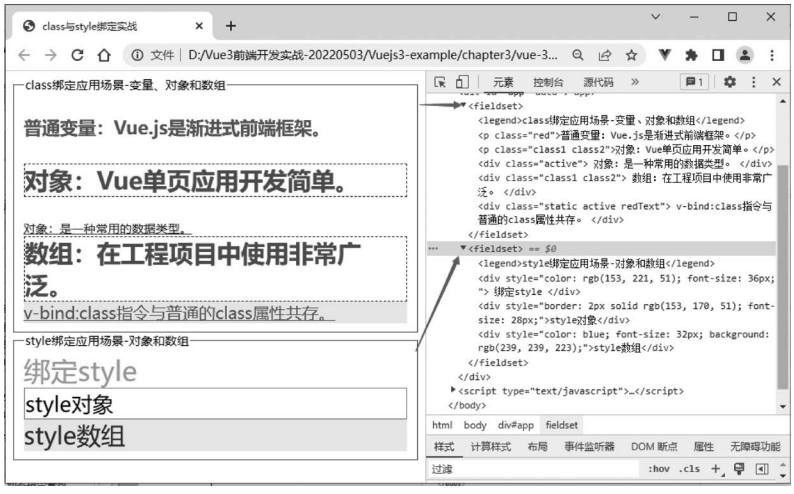

图 3-15　class 和 style 绑定综合应用

```
1.  <!-- vue-3-6.html -->
2.  <!DOCTYPE html>
3.  <html>
4.    <head>
5.      <meta charset="UTF-8" />
```

```
<title>class 与 style 绑定实战</title>
<script src="../js/vue.global.js"></script>
<style type="text/css">
  .red { color: red; font-size: 24px; font-weight: bold; }
  .class1 {color: green;  font-size: 32px; font-weight: bolder; }
  .class2 { border: 1px dashed #0033cc; }
  .active { color: blue; text-decoration: underline;}
  .static {color: #667788;  font-size: 22px;  }
  .redText { color: red; background: #ededed;}
</style>
</head>
<body>
  <div id="app">
    <fieldset>
      <legend>class 绑定应用场景 - 变量、对象和数组</legend>
      <p v-bind:class="myClass">普通变量:Vue.js 是渐进式前端框架.</p>
      <p :class="classObject">对象:Vue 单页应用开发简单.</p>
      <div v-bind:class="{active: isActive }">对象:是一种常用的数据类型.</div>
      <div v-bind:class="[classA, classB]">数组:在工程项目中使用非常广泛. </div>
      <div  class="static" v-bind:class="{active: isActive, redText: hasError }" >
        v-bind:class 指令与普通的 class 属性共存.</div>
    </fieldset>
    <fieldset>
      <legend>style 绑定应用场景 - 对象和数组</legend>
      <div v-bind:style="{color:activeColor,fontSize:fontSize + 'px'}">绑定 style</div>
      <div :style="styleObject">style 对象</div>
      <div v-bind:style="[styleObjectA, styleObjectB]">style 数组</div>
    </fieldset>
  </div>
  <script type="text/javascript">
    const App = {
      data() {
        return {
          myClass: "red",
          classObject: {class1: true,class2: true,},
          classA: "class1",
          classB: "class2",
          isActive: true,
          hasError: true,
          activeColor: "#99DD33",
          fontSize: 36,
          styleObject: { border: "2px" + " solid" + "#99AA33",fontSize: 28 + "px",},
          styleObjectA: { color: "blue", fontSize: 32 + "px",},
          styleObjectB: { background: "#EFEFDF", },
        };
      },
    };
    const instance = Vue.createApp(App).mount("#app");
  </script>
</body>
</html>
```

3.4 事件处理(v-on 指令)

v-on 指令(简写为 @)主要用来侦听 DOM 事件,并在触发时执行对应的 JavaScript。用法:v-on:click="methodName" 或 @click="handler"。

事件处理器的值如下。

（1）内联事件处理器：事件被触发时执行的内联 JavaScript 语句（与 onclick 类似）。

（2）方法事件处理器：一个指向组件上定义的方法的属性名或路径。

1. 内联事件处理器

内联事件处理器通常用于简单场景，举例如下。

```
// 单文件组件中，采用组合式 API(选项式 API 中使用 methods)
const count = ref(0)
<button @click="count++">Add 1</button>
<p>Count is: {{ count }}</p>
```

2. 方法事件处理器

随着事件处理器的逻辑变得越来越复杂，内联代码方式变得不够灵活。因此，v-on 指令也可以接受一个方法名或对某个方法的调用，举例如下。

```
// 在单文件组件中，采用组合式 API(选项式 API 中使用 methods)
const name = ref('Vue.js')
function greet(event) {
  alert(`Hello ${name.value}!`)
  // event 是 DOM 原生事件
  if (event) {
    alert(event.target.tagName)
  }
}
<!-- greet 是上面定义过的方法名 -->
<button @click="greet">Greet</button>
```

方法事件处理器会自动接受原生 DOM 事件并触发执行。在上面的例子中，能够通过被触发事件的 event.target.tagName 访问到该 DOM 元素。

3. 方法与内联事件判断

模板编译器会通过检查 v-on 指令的值是否是合法的 JavaScript 标识符或属性访问路径断定是何种形式的事件处理器。举例来说，foo、foo.bar 和 foo['bar']会被视为方法事件处理器，而 foo()和 count++会被视为内联事件处理器。

4. 在内联处理器中调用方法

除了直接绑定方法名，还可以在内联事件处理器中调用方法，并允许向方法传入自定义参数以代替原生事件。举例如下。

```
// 在单文件组件中采用组合式 API(选项式 API 中使用 methods)
function say(message) {
  alert(message)
}
<button @click="say('hello')">Say hello</button>
<button @click="say('bye')">Say bye</button>
```

5. 在内联事件处理器中访问事件参数

有时我们需要在内联事件处理器中访问原生 DOM 事件。可以向该处理器方法传入一个特殊的 $event 变量，或者使用内联箭头函数，举例如下。

```
<!-- 使用特殊的 $event 变量 -->
<button @click="warn('Form cannot be submitted yet.', $event)">Submit</button>
<!-- 使用内联箭头函数 -->
<button @click="(event) => warn('Form cannot be submitted yet.', event)">
  Submit
</button>
```

```
function warn(message, event) {
  // 这里可以访问原生事件
  if (event) {
    event.preventDefault()
  }
  alert(message)
}
```

v-on 指令绑定事件处理函数可以带参数，也可以不带参数。使用时需要注意以下 3 点。

（1）普通方法。不使用圆括号，event 被自动当作实参传入。

（2）内联处理器中的方法，即方法中调用其他的方法，如 JavaScript 原生方法（如阻止冒泡）、自定义的方法。使用圆括号，必须显式地传入 event 对象。Vue 提供了一个特殊变量 $event，用于访问原生 DOM 事件。

（3）v-on 指令可以使用语法糖简写，格式为@click，等同于 v-on:click。

【例 3-7】 v-on 指令综合应用。代码如下，页面效果如图 3-16 所示。

图 3-16 v-on 指令综合应用

```
1.  <!-- vue-3-7.html -->
2.  <!DOCTYPE html>
3.  <html>
4.    <head>
5.      <meta charset="UTF-8" />
6.      <title>事件侦听实战</title>
7.      <script type="text/javascript" src="../js/vue.global.js"></script>
8.    </head>
9.    <body>
10.     <div id="app">
11.       <h3>v-on 属性值为一般表达式：计数器为{{count}}</h3>
12.       <button type="button" @click="count += 1">开始计数 - {{count}}</button>
13.       <h3>v-on 属性值为方法：当前系统日期为{{now}}</h3>
14.       <button type="button" v-on:click="newDate">显示时期</button>
15.       <h3>v-on 属性值为带参数的方法：产生 6 个随机区间的整数</h3>
16.       <p>{{numArr}}</p>
17.       <button type="button" @click="createRndNum6(100,300)">
18.         产生 6 个随机区间整数
19.       </button>
20.       <h3>v-on 属性值为带 $event 的方法：访问原生的 event 对象为{{information}}</h3>
21.       <button type="button" @click="showInfo1('用 $event 传入!', $event)">
22.         带 $event 参数
23.       </button>
24.       <h3>v-on 属性值不带参数的方法：默认传入 event，类型为{{typeObject}}</h3>
25.       <button type="button" @click="showInfo2">不带参数</button>
26.     </div>
27.     <script type="text/javascript">
28.       const App = {
```

```
29.          data() {
30.            return {
31.              count: 0,
32.              now: "",
33.              numArr: [],
34.              typeObject: "",
35.              information: "",
36.            };
37.          },
38.          methods: {
39.            newDate() {
40.              this.now = new Date().toLocaleDateString();
41.            },
42.            createRndNum6(start, end) {
43.              for (var i = 0; i < 6; i++) {
44.                this.numArr[i] = Math.floor(
45.                  Math.random() * (end - start) + start
46.                );
47.              }
48.              console.log(this.numArr);
49.            },
50.            showInfo1(message, event) {
51.              // 访问原生事件对象,用来阻止默认事件
52.              if (event) event.preventDefault();
53.              this.information = message;
54.              alert(event.target.tagName);     // $event 拿到当前单击事件的事件对象
55.            },
56.            showInfo2(event) {
57.              // 访问原生事件对象,用来阻止默认事件
58.              if (event) event.preventDefault();
59.              this.typeObject = typeof event;
60.            },
61.          },
62.        };
63.        const instance = Vue.createApp(App).mount("#app");
64.      </script>
65.    </body>
66. </html>
```

上述代码中,第 17 行 v-on 指令绑定带参数的事件处理方法 createRndNum6(100,300),随机产生 6 个[100,300]的整数;第 21 行 v-on 指令绑定带参数的事件处理方法 showInfo1('用 $event 传入!', $event),传入特殊变量 $event,用于访问原生 DOM 事件;第 25 行 v-on 指令绑定不带参数的事件处理方法 showInfo2(),在 methods(function)中定义时需要将 event 作为参数。

6. 事件修饰符

在处理事件时调用 event.preventDefault()或 event.stopPropagation()方法是很常见的。尽管可以直接在方法内调用,但如果方法能更专注于数据逻辑而不用去处理 DOM 事件的细节会更好。

为解决这一问题,Vue 为 v-on 指令提供了事件修饰符(Modifier)。修饰符是用.表示的指令后缀。Vue 中的事件修饰符主要如下。

(1).stop:等同于 JavaScript 中的 event.stopPropagation()方法,防止事件冒泡。

(2).prevent:等同于 JavaScript 中的 event.preventDefault()方法,防止执行预设的行为(如果事件可取消,则取消该事件,而不停止事件的进一步传播)。

(3).capture:与事件冒泡的方向相反,事件捕获由外到内。

(4).self:只会触发自己范围内的事件,不包含子元素。

(5).once:只会触发一次。

(6).passive:不检查是否默认事件被阻止,用于触发滚动时性能会更好。

下面分别介绍每个修饰符的作用。

```
<!-- 单击事件将停止传递 -->
<a @click.stop="doThis"></a>

<!-- 提交事件将不再重新加载页面 -->
<form @submit.prevent="onSubmit"></form>

<!-- 修饰符可以使用链式书写 -->
<a @click.stop.prevent="doThat"></a>

<!-- 也可以只有修饰符 -->
<form @submit.prevent></form>

<!-- 仅当 event.target 是元素本身时才会触发事件处理器 -->
<!-- 例如,事件处理器不来自子元素 -->
<div @click.self="doThat">...</div>
```

注意 使用修饰符时需要注意调用顺序,因为相关代码是以相同的顺序生成的。因此,使用@click.prevent.self会阻止元素及其子元素的所有单击事件的默认行为,而@click.self.prevent则只会阻止对元素本身的单击事件的默认行为。

.capture、.once和.passive修饰符与原生的addEventListener事件相对应。

```
<!-- 添加事件监听器时,使用 capture 捕获模式 -->
<!-- 例如,指向内部元素的事件,在被内部元素处理前,先被外部处理 -->
<div @click.capture="doThis">...</div>

<!-- 单击事件最多被触发一次 -->
<a @click.once="doThis"></a>

<!-- 滚动事件的默认行为 (scrolling) 将立即发生而非等待 onScroll 完成 -->
<!-- 以防其中包含 event.preventDefault() -->
<div @scroll.passive="onScroll">...</div>
```

.passive修饰符一般用于触摸事件的侦听器,可以用来改善移动端设备的滚屏性能。

注意 请勿同时使用.passive和.prevent修饰符,因为.passive已经向浏览器表明了不想阻止事件的默认行为。如果这么做了,则.prevent会被忽略,并且浏览器会抛出警告。

【例 3-8】 事件修饰符综合应用。代码如下,页面效果如图3-17所示。

图 3-17 事件修饰符综合应用

```
<!-- vue-3-8.html -->
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>事件修饰符综合应用</title>
    <script type="text/javascript" src="../js/vue.global.js"></script>
  </head>
  <body>
    <div id="vue38">
      <h3>事件修饰符综合应用</h3>
      <div id="app">
        <div v-on:click.prevent.self="divHandler(1)" class="div1">div1</div>
        <div v-on:click.self.prevent="divHandler(2)" class="div2">
          <input
            type="button"  value="button1--单击"  @click.stop="btnHandler(1)"/>
          <a href="http://www.edu.cn" @click.prevent.self="atEdu">教育网(默认行为阻止)</a><br />
            <input type="button" value="button2--单击" @click.capture="btnHandler(2)" />
          <a href="http://www.edu.cn" target="_blank" @click="atEdu"  >教育网(默认行为允许)</a>
          <fieldset>
            <legend>.self 修饰符</legend>
              <input type="button" value="button3--单击" @click.self.prevent="btnHandler(3)"/>
          </fieldset>
          <form v-on:submit.prevent></form>
          <button type="button" @click.once="btnHandler(4)">
            button4--单击多次,只执行一次</button>
          <div v-on:scroll.passive="onScroll" class="div3">
            <h3>Vue.js 是什么</h3>
            <p>Vue 是一款用于构建用户界面的 JavaScript 框架.它基于标准 HTML、CSS 和 JavaScript 构建,并提供了一套声明式的、组件化的编程模型,帮助用户高效地开发用户界面.无论是简单还是复杂的界面,Vue 都可以胜任.</p>
          </div>
          <p>滚动 <span id="demo">{{scrollCount}}</span> 次.</p>
        </div>
      </div>
    </div>
    <script type="text/javascript">
      const App = {
        data() {
          return {
            scrollCount: 0,
            count: 0,
          };
        },
        methods: {
          divHandler(num) {
            console.log("div" + num + "-被单击了!");
          },
          btnHandler(num) {
            this.count = this.count + 1;
            console.log("button-" + num + "被单击了!--" + this.count + "下");
          },
          atEdu() {
            console.log("中国教育和科研计算机网");
          },
          onScroll() {
```

```
            for (var i = 0; i < 10; i++) {
              console.log(i);
            }
            this.scrollCount += 1;
          },
        },
      };
      const instance = Vue.createApp(App).mount("#app");
    </script>
    <style>
      .div1 { width: 300px; height: 60px; background: #eeeeee; }
      .div2 { width: 300px; height: 60px; background: #adeeee; }
      .div3 { width: 300px; height: 150px; border: 4px solid red; overflow: scroll; }
    </style>
  </body>
</html>
```

7. 按键修饰符

在侦听键盘事件时,经常需要检查详细的按键。Vue 允许为 v-on 或@在侦听键盘事件时添加按键修饰符。

```
<!-- 仅在 key 为 Enter 时调用 instance.submit() -->
<input @keyup.enter="submit" />
```

可以直接使用 KeyboardEvent.key 暴露的按键名称作为修饰符,但需要转为 kebab-case 形式。

```
<input @keyup.page-down="onPageDown" />
```

在上面的例子中,仅会在 $event.key 为'PageDown'时调用事件处理。

1) 按键别名

Vue.js 提供了按键别名,也称为快捷名称,如表 3-1 所示。

表 3-1 按键快捷名称

序 号	快捷名称	描 述	序 号	快捷名称	描 述
1	.enter	Enter 键	7	.down	↓键
2	.tab	Tab 键	8	.left	←键
3	.delete	删除和退格键	9	.right	→键
4	.esc	Esc 键	10	.ctrl	Ctrl 键
5	.space	空格键	11	.alt	Alt 键
6	.up	↑键	12	.shift	Shift 键

根据按键的快捷名称,可以对按钮事件增加修饰符,代码如下。

```
<!-- 按下 Tab 键跳到此处时触发事件 -->
<input type="text" v-on:keyup.tab=" inputName ">
<input type="text" v-on:keyup.enter=" inputName ">
<input type="text" v-on:keyup.shift=" inputAge" >
```

2) 系统按键修饰符

可以使用系统按键修饰符触发鼠标或键盘事件侦听器,只有当按键被按下时才会触发。系统按键修饰符有.ctrl、.alt、.shift、.meta。

注意 在 Mac 键盘上,meta 是 Command 键(⌘);在 Windows 键盘上,meta 是 Windows 键(⊞);在 Sun 微机系统键盘上,meta 是钻石键(◆)。在某些键盘上,特别是 MIT 和 Lisp 机器的键盘及其后代版本的键盘,如 Knight 键盘、space-cadet 键盘,meta 都被标记为

META。在 Symbolics 键盘上，meta 也被标记为 META 或 Meta。

举例如下。

```
<!-- Alt + Enter -->
<input @keyup.alt.enter = "clear" />
<!-- Ctrl + 单击 -->
<div @click.ctrl = "doSomething"> Do something </div>
```

注意 系统按键修饰符和常规按键不同。与 keyup 事件一起使用时，该按键必须在事件发生时处于按下状态。换句话说，keyup.ctrl 只会在仍然按住 Ctrl 键但松开了另一个键时被触发。若单独松开 Ctrl 键，将不会触发。

3）.exact 修饰符

.exact 修饰符允许控制触发一个事件所需的确定组合的系统按键修饰符。

```
<!-- 当按下 Ctrl 键时，即使同时按下 Alt 键或 Shift 键也会触发 -->
<button @click.ctrl = "onClick"> A </button>
<!-- 仅当按下 Ctrl 键且未按任何其他键时才会触发 -->
<button @click.ctrl.exact = "onCtrlClick"> A </button>
<!-- 仅当没有按下任何系统按键时触发 -->
<button @click.exact = "onClick"> A </button>
```

4）鼠标按键修饰符

鼠标按键修饰符有.left、.righ、.middle，这些修饰符将处理程序限定为由特定鼠标按键触发的事件。

【例 3-9】 按键修饰符的应用。代码如下，页面效果如图 3-18 所示。

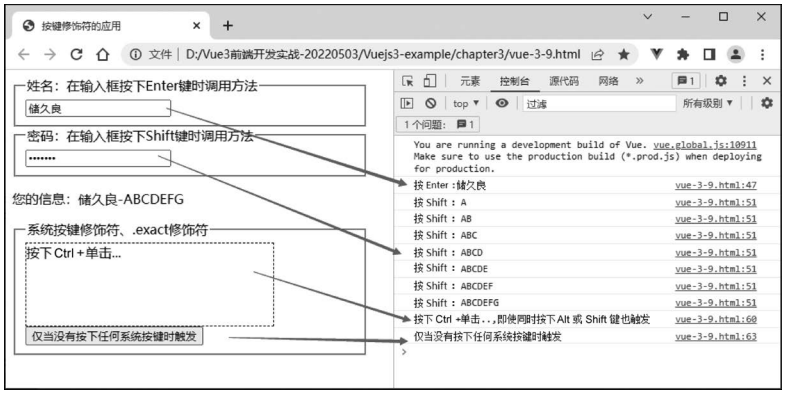

图 3-18 按键修饰符的应用

```
1.  <!-- vue-3-9.html -->
2.  <!DOCTYPE html>
3.  <html lang = "en">
4.    <head>
5.      <meta charset = "UTF-8" />
6.      <title>按键修饰符的应用</title>
7.      <script src = "../js/vue.global.js"></script>
8.    </head>
9.    <body>
10.     <div id = "app">
11.       <fieldset>
12.         <legend>姓名：在输入框按下 Enter 键时调用方法</legend>
13.         <input   type = "text"   v-on:keyup.enter = "inputName"
14.           placeholder = "输入姓名"   v-model = "myName"   />
```

```
    </fieldset>
    <fieldset>
      <legend>密码:在输入框按下 Shift 键时调用方法</legend>
      <input  type="password"  v-on:keyup.Shift="inputPassword"
        placeholder="输入密码"   v-model="myPassword"  />
    </fieldset>
    <p>您的信息:{{myInformation}}</p>
    <fieldset>
      <legend>系统按键修饰符、.exact 修饰符</legend>
      <div class="div1" @click.ctrl="doSomething">按 Crtl + 单击 div</div>
      <button @click.exact="onClick">仅当没有按下任何系统按键时触发</button>
    </fieldset>
  </div>
  <script>
    const App = {
      data() {
        return {
          myInformation: "",
          myName: "",
          myPassword: "",
        };
      },
      methods: {
        inputName() {
          console.log("按 Enter:" + this.myName);
          this.myInformation = this.myName + "-" + this.myPassword;
        },
        inputPassword() {
          console.log("按 Shift:" + this.myPassword);
          this.myInformation = this.myName + "-" + this.myPassword;
        },
        metaClick() {
          console.log("meta + 按键触发");
        },
        doSomething() {
          document.getElementsByClassName("div1")[0].innerHTML =
            "按下 Ctrl + 单击...<br/>";
          console.log("按下 Ctrl + 单击..,即使同时按下 Alt 或 Shift 键也触发");
        },
        onClick() {
          console.log("仅当没有按下任何系统按键时触发");
        },
      },
    };
    const instance = Vue.createApp(App).mount("#app");
  </script>
  <style>
    .div1 { border: 1px dashed black; width: 300px; height: 100px; }
    fieldset { width: 400px;  }
  </style>
 </body>
</html>
```

3.5 表单输入绑定(v-model 指令)

在 Vue.js 中可以通过 v-model 指令在表单的 input、textarea 及 select 元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法更新元素。尽管有些神奇,但 v-model 指令本质上不过是语法糖。它负责侦听用户的输入事件以更新数据,并对一些极端场景进行一些

特殊处理。

v-model 指令会忽略所有表单元素的 value、checked、selected 属性的初始值，而总是将 Vue 实例或组件的数据作为数据来源。程序设计人员可以通过 JavaScript 在组件的 data 选项中声明初始值。

v-model 在内部为不同的输入元素使用不同的属性并抛出不同的事件。

（1）文本输入框和多行文本域元素使用 value 属性和 input 事件。

（2）复选框和单选按钮使用 checked 属性和 change 事件。

（3）下拉列表框将 value 作为 prop，并将 change 作为事件。

注意 对于需要使用输入法（如中文、日文、韩文等）的语言，v-model 指令不会在输入法组合文字的过程中得到更新。如果需要处理这个过程，可使用 input 事件。

【基本语法】

```
<!-- 1.文本输入框,绑定到 value -->
<input type="text" placeholder="输入姓名" v-model="myName">
<!-- 2.多行文本域,绑定到 value -->
<textarea  v-model="message" placeholder="请输入您的建议或意见"></textarea>
<!-- 3.单个复选框,绑定到布尔值 -->
<input type="checkbox" id="checkbox" v-model="checked">
<!-- 4.多个复选框,绑定到同一个数组(checkedNames 值为 []) -->
<input type="checkbox" id="jack" value="Jack" v-model="checkedNames">
<label for="jack">Jack</label>
<input type="checkbox" id="john" value="John" v-model="checkedNames">
<label for="john">John</label>
<input type="checkbox" id="mike" value="Mike" v-model="checkedNames">
<label for="mike">Mike</label>
<span>Checked names: {{ checkedNames }}</span>
<!-- 5.多个单选按钮,绑定到同一个变量 -->
<input type="radio" id="man" value="男"  v-model="sex" />男
<input type="radio" id="woman" value="女"  v-model="sex" />女
<!-- 6.列表框单选时,绑定到一个变量.多选时,绑定到一个数组 -->
<select name="" v-model="mySelected" multiple>
      <option disabled value="">请选择</option>
      <option value="Java 程序设计">Java 程序设计</option>
      <option value="算法设计与分析">算法设计与分析</option>
      <option value="计算机网络">计算机网络</option>
      <option value="数据挖掘与分析">数据挖掘与分析</option>
</select>
```

【语法说明】

在多行文本区域插值（<textarea>{{text}}</textarea>）并不会生效，应用 v-model 代替。

单个复选框，绑定到布尔值；多个复选框，绑定到同一个数组，同时给每个复选框设置不同的 id 值和 value 值。选择的结果将返回到数组中。

要实现一组单选按钮互斥的功能，就需要 v-model 配合 value 使用。给一组单选按钮设置不同的 id 值和不同的 value 值，v-model 绑定同一个数据变量。

在下拉列表框中，如果 v-model 表达式的初始值未能匹配任何选项，select 元素将被渲染为“未选中”状态。在 iOS 中，容易造成用户无法选择第 1 个选项。因为在此情况下，iOS 不会触发 change 事件。因此，推荐像上面这样提供一个值为空的禁用选项。若下拉列表支持多选，请将 v-model 绑定到数组上，与复选框多选类似。

【例 3-10】 表单绑定的综合应用。代码如下，页面效果如图 3-19 所示。

```
<!-- vue-3-10.html -->
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>v-model 绑定表单</title>
    <script src="../js/vue.global.js"></script>
    <style type="text/css">
      fieldset {width: 450px; height: 480px;border: 1px dashed #2c3e50;}
    </style>
  </head>
  <body>
    <div id="app">
      <fieldset>
        <legend align="center">信息调查表</legend>
        姓名:<input type="text" placeholder="输入姓名" v-model="myName" />
        <!-- 姓名:<input type="text" placeholder="输入姓名" v-on:input="myName=
$event.target.value" v-bind:value="myName"> -->
        <p>您的姓名:{{myName}}</p>
        <p>建议或意见:{{message}}</p>
        <textarea v-model="message"  placeholder="请输入您的建议或意见"
          rows="3"  cols="60"></textarea>
        <p>您的兴趣与爱好:{{checkMyLove}}</p>
        <input  type="checkbox" id="music"
          value="绘画" v-model="checkMyLove"  />绘画
        <input  type="checkbox" id="netword"
          value="网络小说"  v-model="checkMyLove" />网络小说
        <input  type="checkbox"  id="game"
          value="音乐"  v-model="checkMyLove"  />音乐
        <p>您的性别:{{ sex }}</p>
        <input type="radio" id="man" value="男" v-model="sex" />男
        <input type="radio" id="woman" value="女" v-model="sex" />女
        <p>选择专业选修课:{{mySelected}}</p>
        <select name="" v-model="mySelected" multiple>
          <option disabled value="">请选择</option>
          <option value="Java 程序设计">AngularJS 前端框架</option>
          <option value="算法设计与分析">Fluttert 移动开发</option>
          <option value="计算机网络">Vue.js 前端框架技术</option>
          <option value="数据挖掘与分析">JSP 程序设计</option>
        </select>
      </fieldset>
    </div>
    <script>
      const App = {
        data() {
          return {
            myName: "",
            message: "",
            checkMyLove: [],
            sex: "",
            mySelected: [],
          };
        },
      };
      Vue.createApp(App).mount("#app");
    </script>
  </body>
</html>
```

上述代码中,第22～28行定义一组复选框,用于选择兴趣与爱好,分别为3个复选框定义不同的id和value,使用v-model指令绑定同一checkMyLove数组,用于存放用户选择的选

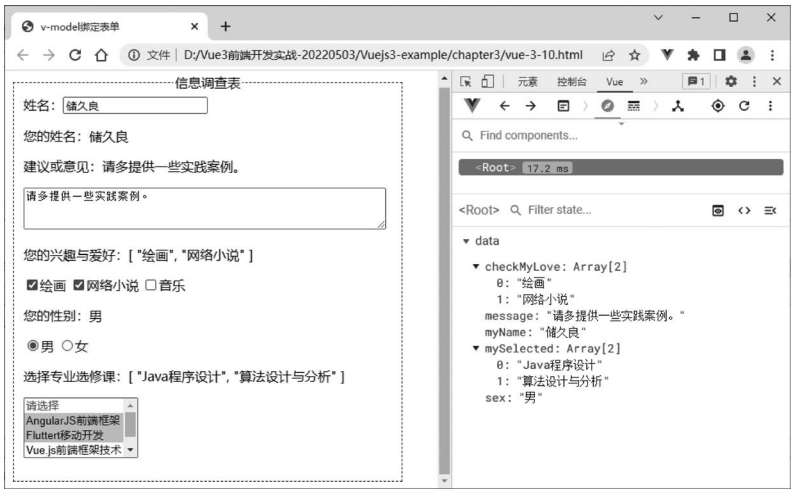

图 3-19　表单绑定的综合应用

项；第 30 行和第 31 行定义一组单选按钮，为每个单选按钮定义不同的 id 和 value，并通过 v-model 指令绑定 sex 属性，sex 的值为选中的单选按钮的 value；第 33～39 行定义下拉列表框，用于显示选择的专业选修课，支持多选，通过 v-model 指令绑定 mySelected 数组（若为单选，可以设为简单变量），其值为选中选项的 value。

【例 3-11】　下拉列表框选项自动渲染。代码如下，页面效果如图 3-20 所示。

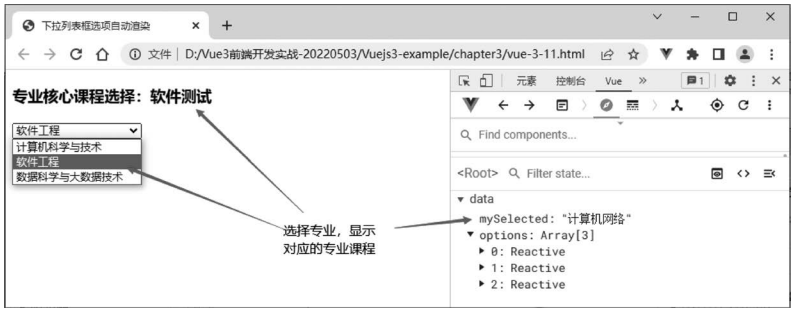

图 3-20　下拉列表框选项自动渲染

```
1.  <!-- vue-3-11.html -->
2.  <!DOCTYPE html>
3.  <html>
4.    <head>
5.      <meta charset="UTF-8" />
6.      <title>下拉列表框选项自动渲染</title>
7.      <script src="../js/vue.global.js"></script>
8.    </head>
9.    <body>
10.     <div id="app">
11.       <h3>专业核心课程选择:{{mySelected}}</h3>
12.       <select v-model="mySelected">
13.         <option v-for="(option,index) in options" v-bind:value="option.value">
14.           {{option.text}}
15.         </option>
16.       </select>
17.     </div>
18.     <script>
19.       const App = {
```

```
        data() {
          return {
            mySelected: "计算机网络",
            options: [
              { value: "计算机网络", text: "计算机科学与技术", },
              { value: "软件测试", text: "软件工程",},
              { value: "Python 程序设计",text: "数据科学与大数据技术",},
            ],
          };
        },
      };
      const instance = Vue.createApp(App).mount("#app");
    </script>
  </body>
</html>
```

1. 表单元素值绑定

对于单选按钮、复选框及下拉列表框的选项，v-model 指令绑定的值通常是静态字符串或布尔值。但是，有时需要把值绑定到 Vue 组件或实例的一个动态属性上，就可以用 v-bind 指令实现，并且这个属性的值可以不是字符串。

【基本语法】

```
<input type="radio" v-model="myRadio" value="radio" >
<input type="radio" v-model="myRadio" v-bind:value="radio1" >
<input type="checkbox" v-model="myChecked1">
<input type="checkbox" v-model="myChecked2"
:true-value="valueA" :false-value="valueB">
<select v-model="mySelected">
   <option value="html">HTML</option>
   <option :value="{valueC: 'js'}">JS</option>
</select>
//以下需要在 Vue 组件或实例中的 data 选项中定义
myChecked1:'',
myChecked2:'',
mySelected: '',
myRadio: '',
radio1: '123',
valueA: "A",
valueB: "B",
```

【语法说明】

当选中第 1 个单选按钮时，myRadio 的值为字符串"radio"；当选中第 2 个单选按钮时，myRadio 的值为字符串"123"。当选中第 1 个复选框时，myChecked1 为 true，否则为 false；当选中第 2 个复选框时，myChecked2 为"A"(动态绑定数据属性)。当选中下拉列表框中第 1 个选项时，mySelected 为字符串 "html"；当选中下拉列表框中第 2 个选项时，mySelected 为对象{valueC：'js'}。具体代码参见例 3-12。

【例 3-12】 表单元素值绑定。代码如下，页面效果如图 3-21 所示。

```
<!-- vue-3-12.html -->
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>表单元素值绑定</title>
    <script src="../js/vue.global.js"></script>
  </head>
  <body>
```

```
    <div id="app">
      <h3>
        单选按钮值绑定:
        <input type="radio" v-model="myRadio" value="逛街" />上街去逛逛
        <input type="radio" v-model="myRadio" v-bind:value=" radio1 " /> 在家宅着
      </h3>
      <p>单选按钮 myRadio:{{myRadio}}</p>
      <h3>
        复选框值绑定:
        <input type="checkbox" v-model="myChecked1" value="checked" />音乐
        <input
          type="checkbox" v-model="myChecked2"
          :true-value="valueA" :false-value="valueB" />绘画
      </h3>
      <p>复选框 1:{{myChecked1}},复选框 2:{{myChecked2}}</p>
      <h3>下拉列表框:{{mySelected}}</h3>
      <select v-model="mySelected">
        <option value="请选择" disabled>请选择</option>
        <option value="计算机网络">计算机网络</option>
        <option :value="{classHours: '56 学时'}">Vue.js 前端框架技术</option>
      </select>
    </div>
    <script type="text/javascript">
      const App = {
        data() {
          return {
            myChecked1: "",
            myChecked2: "",
            mySelected: "",
            myRadio: "",
            radio1: "宅家",
            valueA: "我喜欢",
            valueB: "不想去",
          };
        },
      };
      const instance = Vue.createApp(App).mount("#app");
    </script>
  </body>
</html>
```

图 3-21　表单元素值绑定

2. v-model 修饰符

v-model 指令的修饰符一般用于控制数据同步的时机。Vue 内置的 v-model 指令修饰符如表 3-2 所示。

表 3-2 v-model 指令修饰符

修饰符	描述
.trim	自动过滤输入内容最开始和最后的空格，中间会保留一个空格，多的会被过滤掉
.number	自动将输入的数据转换为 number 类型
.lazy	一般情况下，在 input 上使用 v-model 指令会一直同步输入的内容与显示的内容，不过添加 .lazy 后，就会变成在失去焦点或按 Enter 键时才更新数据

【基本语法】

```
<input v-model.lazy="myNo"  placeholder="输入学号">
<input v-model.number="myAge" type="number" placeholder="输入年龄">
<input v-model.trim="myIdea" placeholder="输入建议">
```

【语法说明】

v-model.lazy 修饰符表示在输入域失去焦点或按 Enter 键时数据才更新；v-model.number 修饰符表示将输入域中内容转换为数值型数据；v-model.trim 修饰符表示将输入域中内容的前后空格过滤掉，并将字符串中间的多个空格转换为一个空格。

【例 3-13】 v-model 修饰符的应用。代码如下，页面效果如图 3-22 和图 3-23 所示。

图 3-22 v-model 修饰符的应用（初始页面）

图 3-23 v-model 修饰符的应用（输入信息页面）

```
1.  <!-- vue-3-13.html -->
2.  <!DOCTYPE html>
3.  <html>
4.      <head>
5.          <meta charset="UTF-8">
6.          <title>v-model 修饰符的应用</title>
7.          <script type="text/javascript" src="../js/vue.global.js"></script>
8.      </head>
9.      <body>
10.       <div id="app">
11.           <fieldset>
12.               <legend>v-model 修饰符的应用-教学意见反馈表</legend>
```

```
13.             姓名<input v-model.lazy="name" placeholder="输入姓名"><br>
14.             学号<input v-model.number="myNo" type="number" placeholder="输入学号"><br>
15.             教学意见:<textarea v-model.trim="myIdea" rows="3" cols="50" placeholder
 ="输入意见"></textarea>
16.             <h4>信息汇总</h4>
17.             <p>姓名:{{name}},学号:{{myNo}}({{typeof myNo}}),意见:{{myIdea}}</p>
18.             </fieldset>
19.             </div>
20.         <script>
21.             const App= {
22.               data() {
23.                 return {
24.                 name: '',
25.                 myNo: '',
26.                 myIdea: '',
27.                 }
28.               },
29.             };
30.             const instance=Vue.createApp(App).mount('#app');
31.         </script>
32.     </body>
33. </html>
```

上述代码中，第 13 行文本框使用了 v-model.lazy 修饰符，输入姓名时不会在第 17 行的<p>标记内立即更新，而是等待姓名完全输入结束后才更新；第 14 行文本框使用了 v-model.number 修饰符，会将文本框输入的字符串型数据转换为数值型数据，并在第 17 行<p>标记中显示学号和类型；第 15 行使用了 v-model.trim 修饰符，将用户输入的建议的开头和结束处多余的空格全部删除，并将字符串中间输入的多个空格变成一个空格。

3.6 v-html 与 v-text 指令

当网速很慢或 JavaScript 出错时，会暴露{{message}}。因此，推荐使用 v-text 指令，可以很好地解决这个问题。为了输出真正的 HTML，可以使用 v-html 指令。

【基本语法】

```
<p>{{myText}}</p>
<p v-text="myText"></p>              <!-- 以上两种形式是等价的 -->
<span v-html="varHtml"></span>       <!-- 解析为 HTML,并显示相关标记的样式效果 -->
<span v-text="varHtml"></span>       <!-- 解析为纯文本,HTML 标记也显示出来 -->
```

【语法说明】

v-text 指令更新元素的 textContent 属性。可以读取文本，但不能读取<html>标记，解析为文本。如果要更新部分的 textContent 值发生改变，插值处的内容也会随之更新。

v-html 指令更新元素的 innerHTML 属性。可以读取<html>标记，解析出<html>标记。它与 v-text 指令的区别在于 v-text 输出的是纯文本，浏览器不会对其再进行 HTML 解析，但 v-html 指令会将其解析为<html>标记后输出。

注意 在正式环境中动态渲染 HTML 是很危险的，因为容易受到 XSS 攻击。

【例 3-14】 v-text 与 v-html 指令比较应用。代码如下，页面效果如图 3-24 所示。

```
1.  <!-- vue-3-14.html -->
2.  <!DOCTYPE html>
```

```
<html>
  <head>
    <meta charset="UTF-8" />
    <title>v-text 和 v-html 指令的应用</title>
    <script type="text/javascript" src="../js/vue.global.js"></script>
  </head>
  <body>
    <div id="app">
      <fieldset>
        <legend>v-text 指令</legend>
        <p>插值:{{myContent}}</p>
        <!-- 两种形式是等价的 -->
        <p>v-text:<span v-text="myContent">我被替换啦!</span></p>
      </fieldset>
      <fieldset>
        <legend>v-html 指令与 v-text 指令区别</legend>
        <p><b>htmlCode 值为:</b>{{htmlCode}}</p>
        <p>使用 v-html 结果:<span v-html="htmlCode"></span></p>
        <p>使用 v-text 结果:<span v-text="htmlCode"></span></p>
      </fieldset>
    </div>
    <script type="text/javascript">
      const instance = Vue.createApp({
        data() {
          return {
            myContent: "Vue.js 是渐进式前端框架!",
            htmlCode:
              '<p style="color:green"><strong>Vue.js 是渐进式前端框架!</strong></p>',
          };
        },
      }).mount("#app");
    </script>
  </body>
</html>
```

图 3-24 v-text 与 v-html 指令比较应用

3.7 v-once、v-cloak、v-pre 指令

【基本语法】

```
<p v-once>v-once:{{myInput}}</p>
<p><span v-cloak>{{information}} </span></p>
<span v-pre>{{message}}</span>
```

【语法说明】

v-once 指令：只执行一次 DOM 渲染，渲染完成后视为静态内容，跳出以后的渲染过程。

v-pre 指令：在模板中跳过 Vue 的编译，直接输出原始值。这时并不会输出 message 值，而是直接在网页中显示{{message}}。

v-cloak 指令：保持在元素上直到关联实例结束编译。和 CSS 规则([v-cloak]{display: none;})一起使用时，这个指令可以隐藏未编译的{{information}}，直到实例准备完毕。渲染时会出现变量闪烁。

【例 3-15】 v-pre、v-once、v-cloak 指令综合应用。代码如下，页面效果如图 3-25 所示。

(a) 初始渲染的页面　　(b) 输入信息后渲染的页面

图 3-25　v-once、v-cloak、v-pre 指令综合应用

```
<!-- vue-3-15.html -->
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>v-pre、v-once 及 v-cloak 指令的应用</title>
    <style type="text/css">
      [v-cloak] {display: none;}
    </style>
    <script type="text/javascript" src="../js/vue.global.js"></script>
  </head>
  <body>
    <div id="app" v-cloak>
      <fieldset>
        <legend>v-once 指令</legend>
        <p v-once>设置 v-once,只渲染一次:{{myInput}}</p>
        <p>未设置 v-once,同步更新:{{myInput}}</p>
        <div>输入信息:<input type="text" v-model="myInput" size="60" /></div>
      </fieldset>
      <fieldset>
        <legend>v-pre 指令与 v-text 指令的区别</legend>
        <p>message 值为:{{message}}</p>
        <p>v-text: <span v-text="message"></span></p>
        <p>v-pre: <span v-pre>{{message}}</span></p>
        <p>v-cloak:<span v-cloak>{{information}} </span></p>
      </fieldset>
    </div>
    <script type="text/javascript">
      Vue.createApp({
        data() {
          return {
            message: "Web 前端框架技术",
            myInput: "请输入信息",
```

```
34.              information: "Vue.js 前端框架技术与实战",
35.            };
36.          },
37.        }).mount("#app");
38.      </script>
39.    </body>
40. </html>
```

上述代码中，第 25 行使用 v-cloak 指令消除页面内闪烁，配合 CSS 样式，实现当{{information}}未渲染完成时隐藏，当渲染结束时显示。实际上效果未体现。

3.8 Vue.js 自定义指令

除了 Vue 内置的一系列指令(如 v-model 或 v-show)之外，Vue 还允许注册自定义的指令(Custom Directives)。Vue 中重用代码的方式有组件和组合式函数。组件是主要的构建模块，而组合式函数则侧重于有状态的逻辑。另外，自定义指令主要是为了重用涉及普通元素的底层 DOM 访问的逻辑。

一个自定义指令由一个包含类似组件生命周期钩子的对象来定义。钩子函数会接受指令所绑定元素作为其参数。下面是一个自定义指令的例子，当一个 input 元素被 Vue 插入 DOM 中后，它会被自动聚焦，代码如下。

```
// 组件中 JS 部分
const focus = {
  mounted: (el) => el.focus()          // 使用箭头函数定义
}
export default {
  directives: {                        // 组件内局部注册
    // 在模板中启用 v-focus
    focus
  }
}
<!--  HTML 模板中 -->
<input v-focus />
```

将一个自定义指令全局注册到应用层级也是一种常见的做法，代码如下。

```
const app = Vue.createApp({})
// 使 v-focus 在所有组件中都可用
app.directive('focus', {
  /* ... */
})
```

3.8.1 自定义指令注册

【基本语法】

```
<标记名称 v-my-directive>使用方式与内置指令相同</标记名称>  <!-- 自定义指令 -->
```

(1) 注册全局自定义指令 v-my-directive。第 1 种定义方法如下。

```
const app = Vue.createApp({});
app.directive('my-directive', {
  // 生命周期钩子函数
})
```

第 2 种定义方法如下。

```
app.directive('my-directive', function(el, binding, vnode){}),
// 其中 el 为 DOM,vnode 为 Vue 的虚拟 DOM,binding 为一个较复杂的对象
// 使用箭头函数的形式定义
app.directive("color", (el, binding) => {
  // 这会在 mounted 和 updated 时都调用
  el.style.color = binding.value;
  console.log(el.style.color);
});
```

(2) 注册局部自定义指令 my-directive。在组件内使用 directives 选项定义,代码如下。

```
directives: {
 my-directive: { }       // 包含生命周期钩子的指令对象
}
```

(3) 指令钩子函数。

一个指令的定义对象可以提供几种钩子函数,包括 created、beforeMount、mounted、beforeUpdate、updated、beforeUnmount、unmounted,不过它们都是可选的,定义如下。

```
const myDirective = {
  // 在绑定元素的属性前或事件监听器应用前调用
  created(el, binding, vnode, prevVnode) {
    // 下面介绍各个参数的细节
  },
  // 在元素被插入 DOM 前调用
  beforeMount(el, binding, vnode, prevVnode) {},
  // 在绑定元素的父组件及其自己的所有子节点都挂载完成后调用
  mounted(el, binding, vnode, prevVnode) {},
  // 绑定元素的父组件更新前调用
  beforeUpdate(el, binding, vnode, prevVnode) {},
  // 在绑定元素的父组件及其自己的所有子节点都更新后调用
  updated(el, binding, vnode, prevVnode) {},
  // 绑定元素的父组件卸载前调用
  beforeUnmount(el, binding, vnode, prevVnode) {},
  // 绑定元素的父组件卸载后调用
  unmounted(el, binding, vnode, prevVnode) {}
}
```

【语法说明】

指令注册方式分为全局注册和局部注册。使用组件的 directives 选项定义的指令称为局部自定义指令,使用 Vue.createApp({}).directive()或 app.directive()方法定义的指令称为全局自定义指令。另外,定义的指令不支持驼峰式(如 myDirective)写法。

在使用时,钩子函数也会传入以下参数。

(1) el:指令所绑定的元素,可以用来直接操作 DOM。

(2) binding:一个对象,包含以下属性。

- value:传递给指令的值,如 v-my-directive="1 * 3"中,绑定值为 3。
- oldValue:之前的值,仅在 beforeUpdate 和 updated 中可用。无论值是否更改,它都可用。
- arg:传递给指令的参数(如果有的话),如在 v-my-directive:foo 中,参数是"foo"。
- modifiers:一个包含修饰符的对象(如果有的话)。例如,在 v-my-directive.foo.bar 中,修饰符对象是{foo:true,bar:true}。
- instance:使用该指令的组件实例。
- dir:指令的定义对象。

(3) vnode:代表绑定元素的底层 VNode。

（4）prevNode：之前的渲染中代表指令所绑定元素的 VNode。仅在 beforeUpdate 和 updated 中可用。

举例来说，像下面这样使用指令：

```
<div v-example:foo.bar="baz">     <!-- 模板中使用方法 -->
```

binding 参数会是一个这样的对象：

```
{
  arg: 'foo',
  modifiers: { bar: true },
  value: /* baz 的值 */,
  oldValue: /* 上一次更新时 baz 的值 */
}
```

自定义指令的参数也可以是动态的。在 v-my-directive:[argument]="value" 中，argument 参数可以根据组件实例数据进行更新。这使得自定义指令可以在应用中被灵活使用。

例如，以下指令的参数会基于组件的 arg 数据属性响应式地更新。

```
<div v-example:[arg]="value"></div>
```

注意 除了 el 外，其他参数都是只读的，不要更改它们。若需要在不同的钩子间共享信息，推荐通过元素的 dataset 属性实现。

对于自定义指令，一个很常见的情况是仅仅需要在 mounted 和 updated 钩子上实现相同的行为，除此之外并不需要其他钩子。这种情况下可以直接用一个函数定义指令。

```
<!--  模板中使用方法  -->
<div v-color="color"></div>
// JS 部分
app.directive('color', (el, binding) => {
// 默认会在 mounted 和 updated 时都调用
  el.style.color binding.value = "red"              //颜色可以是英文名、十六进制或 RGB 等
 })
```

【例 3-16】 自定义指令。设计要求：①采用全局注册方式定义元素自动获得焦点的 v-focus 指令；②采用局部注册方式定义随机改变元素的背景颜色的 v-color 指令。代码如下，页面效果如图 3-26 所示。

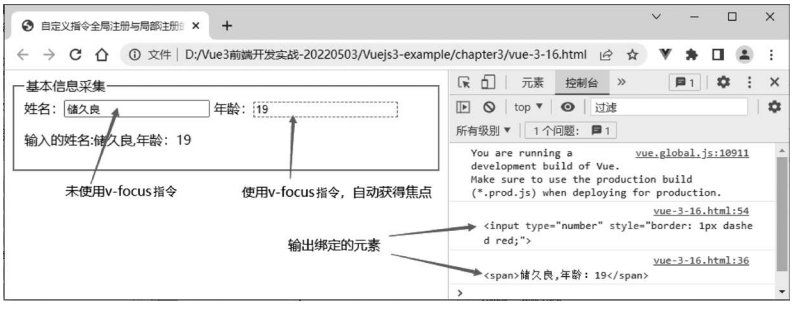

图 3-26 自定义指令

```
1.  <!-- vue-3-16.html -->
2.  <!DOCTYPE html>
3.  <html>
```

```
4.    <head>
5.      <meta charset="UTF-8" />
6.      <title>自定义指令全局注册与局部注册的应用</title>
7.      <script type="text/javascript" src="../js/vue.global.js"></script>
8.    </head>
9.    <body>
10.     <div id="app">
11.       <fieldset>
12.         <legend>基本信息采集</legend>
13.         姓名:<input type="text" v-model="myName" />
14.         年龄:<input type="number" v-focus v-model="myAge"/>
15.         <p>输入的姓名:<span v-color>{{myName}},年龄:{{myAge}}</span></p>
16.       </fieldset>
17.     </div>
18.     <script type="text/javascript">
19.       const App = {
20.         data() {
21.           return {
22.             myName: "储久良",
23.             newColor: "red",
24.             myAge: 19,
25.           };
26.         },
27.         directives: {
28.           //注册一个局部指令,随机改变元素的背景颜色
29.           color: {
30.             mounted: function (el, binding) {
31.               console.log(el);
32.               var redColor = Math.floor(Math.random() * 256).toString(16);
33.               var greenColor = Math.floor(Math.random() * 256).toString(16) ;
34.               var blueColor = Math.floor(Math.random() * 256.toString(16));
35.               el.style.backgroundColor = "#" + redColor + greenColor + blueColor;
36.             },
37.           },
38.         },
39.       };
40.       const app = Vue.createApp(App);
41.       //注册一个全局自定义指令 v-focus
42.       app.directive("focus", {
43.         // 当被绑定的元素插入 DOM 中时
44.         mounted: function (el) {
45.           console.log(el);          // 就是<input>标记
46.           el.focus();               // 聚焦元素
47.           el.style.border = "1px dashed red";
48.         },
49.       });
50.       const instance = app.mount("#app");
51.     </script>
52.   </body>
53. </html>
```

上述代码中，第 14 行使用自定义全局指令 v-focus 实现绑定元素自动获得焦点；第 15 行使用自定义局部指令 v-color 实现绑定的元素自动随机添加背景颜色；第 19～39 行定义 Vue 组件 App，并在其中局部注册 v-color 指令，实现绑定元素随机改变背景颜色；第 42～49 行全局注册自定义指令 v-focus，当元素挂载到 DOM 时，实现自动获得焦点。

3.8.2 对象字面量

在计算机科学中，字面量(Literal)是用于表达源代码中一个固定值的表示法(Notation)。字面量表示如何表达这个值，一般除去表达式，给变量赋值时，等号右边都可以认为是字面量。

字面量分为字符串字面量(String Literal)、数组字面量(Array Literal)和对象字面量(Object Literal),另外还有函数字面量(Function Literal)。

对象字面量是封闭在花括号对({})中的一个对象的零个或多个“属性名:值”列表。

若自定义指令需要多个值,可以传入一个 JavaScript 对象字面量。指令可以使用任意合法的 JavaScript 表达式。

```
<div v-my-directive ="{color: 'red', text: 'Vue 简单易学!'}"></div>
app.directive('my-directive', (el,binding) =>{
  console.log(binding.value.color)          // "red"
  console.log(binding.value.text)           // " Vue 简单易学!"
})
```

当在组件上使用自定义指令时,它会始终应用于组件的根节点,和透传 Attributes 类似。

```
<!--  在模板中使用  -->
<MyComponent v-demo="test" />
<!-- MyComponent 的模板 -->
<div>     <!-- v-demo 指令会被应用在此处 -->
  <span>My component content</span>
</div>
```

注意 组件可能含有多个根节点。当应用到一个多根组件时,指令将会被忽略且抛出一个警告。和 attribute 不同,指令不能通过 v-bind="$attrs"传递给一个不同的元素。总的来说,不推荐在组件上使用自定义指令。

【例 3-17】 自定义指令钩子函数与动态参数实战。设计要求:①采用全局注册方式定义元素特定样式的 v-black-bold 指令,样式效果为“加粗,黑色,1px 黑色虚线边框”;②输出主要钩子函数的相关参数和值;③定义全局自定义指令 v-move-top,功能是通过固定布局方式将元素固定在离顶部一定像素的位置上;④定义全局自定义指令 v-move,功能是通过固定布局方式将元素固定在离顶部或左边一定像素的位置上。代码如下,页面效果如图 3-27 和图 3-28 所示。

```
1.  <!-- vue-3-17.html -->
2.  <!DOCTYPE html>
3.  <html>
4.    <head>
5.      <meta charset="UTF-8" />
6.      <title>钩子函数参数与动态参数的应用</title>
7.      <script src="../js/vue.global.js" type="text/javascript"></script>
8.    </head>
9.    <body>
10.     <div id="app">
11.       <fieldset class="down">
12.         <legend>根据输入的偏移量定位元素</legend>
13.         <span v-move-top="valueA">将我从页面顶部下移{{valueA}}px</span>
14.         <p v-move:[direction]="valueA">将我从页面{{direction=='left'?'左边右移':'顶部
下移'}}{{valueA}}px</p>
15.       </fieldset>
16.       <fieldset>
17.         <legend>操作区域</legend>
18.         <input  type="number"  v-model="valueA"
19.           min="10"  max="150"  />
20.         <button @click="change">改变方向</button>
21.       </fieldset>
22.       <fieldset>
23.         <legend>输出钩子函数的相关参数值</legend>
24.         <p v-black-bold:myarg.x.y="message"></p>
25.       </fieldset>
26.     </div>
```

```
27.     <script type="text/javascript">
28.       const App = {
29.         data() {
30.           return {
31.             message: "钩子函数参数的应用",
32.             valueA: 10,
33.             direction: "left",
34.           };
35.         },
36.         methods: {
37.           change() {
38.             this.direction = this.direction == "left" ? "top" : "left";
39.             console.log(this.direction);
40.           },
41.         },
42.       };
43.       const app = Vue.createApp(App);
44.       app.directive("black-bold", (el, binding, vnode) => {
45.        el.style.fontWeight = "bold";
46.        el.style.color = "black";
47.        el.style.border = "1px dashed #000000";
48.        el.innerHTML = "value: " +  binding.value + "<br>" +  "argument: " +
49.         binding.arg +  "<br>" + "modifiers: " + JSON.stringify(binding.modifiers) +
50.         "<br>" +  "dir:" +  binding.dir +  "<br>" +
51.         "vnode keys: " +  Object.keys(vnode).join(", ");
52.       });
53.       app.directive("move-top", (el, binding) => {
54.        el.style.position = "fixed";
55.        el.style.border = "1px dashed red";
56.        el.style.top = binding.value + "px";
57.       });
58.       app.directive("move", (el, binding) => {
59.        el.style.position = "fixed";
60.        el.style.backgroundColor = "#ADADAD";
61.        var leftortop = binding.arg == "left" ? "left" : "top";
62.        el.style[leftortop] = binding.value + "px";
63.       });
64.       app.mount("#app");
65.     </script>
66.     <style>
67.       fieldset {width: 500px;}
68.       .down { width: 500px;height: 200px;}
69.       .down p { margin-top: 20px; }
70.       span { margin-left: 180px;  margin-top: 20px; }
71.     </style>
72.   </body>
73. </html>
```

上述代码中，第 18 行定义文本输入框，使用 v-model 指令绑定 valueA，单击微调按钮触发数据更新；第 24 行使用自定义全局指令 v-black-bold 实现绑定元素特定样式（黑色，加粗），元素渲染结果如图 3-27 所示；第 44～52 行全局注册自定义指令 v-black-bold，其中第 47 行实现绑定元素的边框样式为“1px dashed #000000'”，第 48～51 行输出钩子函数相关参数和值；第 53～57 行使用全局注册方式自定义 v-move-top 指令，该指令实现绑定元素采用固定布局方式，从顶部向下移动 valueA 像素；第 14 行使用自定义指令 v-move，并设置动态参数[direction]，绑定 valueA，动态参数可以赋值为 left 或 top，默认为 left，该元素向右移 valueA 像素；第 58～63 行全局注册 v-move 指令，根据动态参数决定实现该元素以固定布局方式、有边框效果、左移/右移 valueA 像素。当对象的属性以变量的形式出现时，通常使用数组的方式

图 3-27 自定义指令综合应用(初始页面)

(如代码第 62 行)设置或返回对象的属性值。当单击“改变方向”按钮时,改变 direction(默认为 left,单击后为 top)。此时灰色段落移动方向改为上下移动,如图 3-28 所示。其中 3 个自定义指令会在 mounted 和 updated 钩子函数中调用。

图 3-28 自定义指令综合应用(操作页面)

本章小结

本章主要介绍了 Vue.js 指令的定义与指令分类,重点介绍了指令的注册方式。

Vue.js 指令分为内置指令与自定义指令。指令注册方式分为全局注册与局部注册。

Vue.js 内置指令主要有条件渲染指令(v-if、v-else-if、v-else、v-show)、列表渲染指令(v-for)、类与样式绑定指令(v-bind)、事件处理指令(v-on)、表单输入绑定指令(v-model)、输出元素内容指令(v-text)、输出元素内 HTML 指令(v-html)、隐藏元素指令(v-cloak)、原样输出元素内容指令(v-pre)、只执行一次指令(v-once)。事件修饰符有.stop、.prevent、.capture、.self、.passive、.once。v-model 指令有修饰符,分别为.trim、.lazy、.number。键盘

事件也可以设置按键修饰符。

Vue.js 自定义指令主要是对 Vue.js 内置指令功能的补充。自定义指令使用 app.directive() 方法进行全局注册。定义时根据需要选择不同的钩子函数，有 created、beforeMount、mounted、beforeUpdate、updated、beforeUnmount、unmounted。要求熟练地使用相关参数，主要参数有 el、binding、vnode、prevNode，其中 binding 是一个复杂的对象，包含 value、oldValue、arg、modifiers、instance、dir 等属性。

Vue.js 自定义指令还支持动态参数，格式为 v-my-directive:[argument]="value"，argument 参数可以根据组件实例数据进行更新。

扫一扫

自测题

练习 3

1. 选择题

（1）下列选项中，采用语法糖绑定 href 属性的是（　　）。

A. v-bind:href='ref2'　　B. v-on:click='addOne'

C. @href='ref3'　　D. :href='href1'

（2）下列选项中，在输入框中输入内容后按 Enter 键的按键修饰符是（　　）。

A. .tab　　B. .enter　　C. .ctrl　　D. .shift

（3）下列选项中，能够防止执行预设的行为的事件修饰符是（　　）。

A. .prevent　　B. .capture　　C. .stop　　D. .passive

（4）下列 v-model 修饰符中，能够在输入结束失去焦点或按 Enter 键时才更新数据的是（　　）。

A. .number　　B. .trim　　C. .lazy　　D. 所有选项都是

（5）下列选项中，能够解析为 HTML 代码，并且显示相关标记的样式效果的是（　　）。

A. v-bind　　B. v-html　　C. v-text　　D. v-pre

（6）下列选项中，能够将标记的内容原样输出的指令是（　　）。

A. v-pre　　B. v-html　　C. v-once　　D. v-on:click.once

（7）下列选项中，能够进行事件处理的指令是（　　）。

A. v-for　　B. v-on:click(@ click)

C. v-if　　D. v-model

（8）下列选项中，能够进行表单输入绑定的指令是（　　）。

A. v-html　　B. v-bind　　C. v-model　　D. v-cloak

2. 填空题

（1）Vue.js 中条件渲染指令有 v-if、_____、v-else 和_____。

（2）Vue 组件中定义数据选项 data 为_____形式，其中定义的“属性/值”对，必须_____出来，才能正确提供响应式数据。

（3）显示< html >标记的样式效果和内容的指令是_____；仅作为纯属文本输出内容的指令是_____；能够渲染列表的指令是_____；只渲染一次的指令是_____。

（4）自定义指令分为_____注册、_____注册两种方式。全局注册使用_____来定义。

3. 简答题

（1）简述 v-if 与 v-show 指令在使用上的区别。

（2）事件处理的修饰符有哪些？分别代表什么意思？

（3）自定义指令钩子函数有哪些？

项目实战 3

扫一扫

视频讲解

1. 内置指令实战——课程模块维护

【实战要求】

（1）学会引入 vue.global.js 文件，完成简易表单设计。

（2）学会定义 Vue 组件对象，会配置 data 和 methods 等选项。

（3）学会使用 v-on、v-model 及 v-for 等内置指令。

（4）学会使用数组相关方法，完成课程增加与删除操作。

【设计要求】

按如图 3-29 所示的页面效果，完成项目实战。具体设计要求如下。

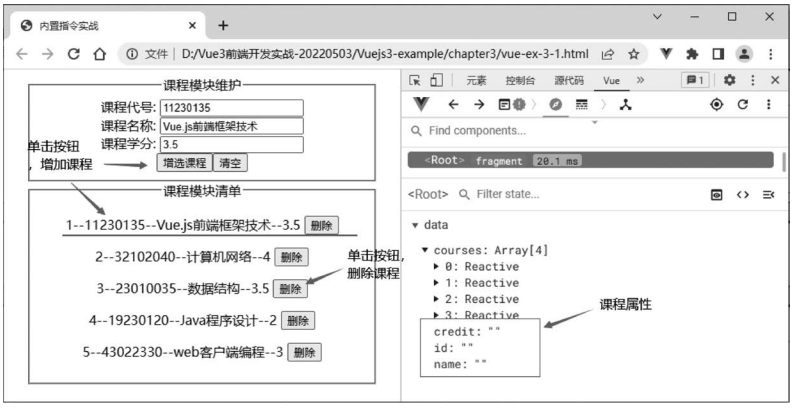

图 3-29 课程管理页面

（1）设计表单。表单中有 3 个文本输入框，分别输入课程代号、课程名称和课程学分，分别使用 v-model 指令绑定 id、name 和 credit 属性。另外，还有一个普通按钮和一个清空按钮，value 值分别为“增选课程”“清空”，绑定单击事件，分别调用 addCourse()和 clear()方法。

（2）课程模块清单显示。课程属性包含课程代号(id)、课程名称(name)和课程学分(credit)。在< p >元素上采用 v-for 指令遍历所有课程信息。课程信息保存在 courses 对象数组中，并初始设置 4 门课程信息，如图 3-29 所示。每门课程后面添加一个“删除”普通按钮，单击此按钮，调用 deleteCourse(i)方法，将从课程对象数组中删除此门课程。

（3）定义相关方法。在 methods 选项中定义 addCourse()、deleteCourse(i)和 clear()方法。其中，addCourse()方法的功能是将添加课程信息(不能为空)从数组的开头处插入课程对象数组中；deleteCourse(i)方法的功能是删除 i 位置上指定的元素；clear()方法的功能是清除 3 个文本输入框中的内容。

【实战步骤】

（1）建立 HTML 文件。将项目文件命名为 vue-ex-3-1.html，在< head >标记中引入 vue.global.js 文件，按设计要求(1)完成“课程模块维护”表单内容的设计。

（2）定义 Vue 组件。分别完成 data、methods 等选项的配置。

- 定义 data(){return{}}。分别定义 id、name、credit 等变量，按设计要求(2)完成 courses 对象数组的初始化工作。

- 配置 methods 选项。定义 addCourse()、deleteCourse(i)和 clear()等 3 个方法。方法定义的形式如下。

```
functionName(){…}
functionName:function(){…}
```

（3）项目调试并运行。在浏览器中调试页面，输入课程代号、课程名称和课程学分后，单击“增选课程”按钮，查看课程信息是否添加在所有课程信息的最前面，如果是，则程序设计正确。再单击“清空”按钮，查看 3 个文本输入框中的内容是否清空。如果有错，可根据错误信息进行修改。单击课程信息后面的“删除”按钮，查看是否可以正确删除该门课程。

扫一扫

视频讲解

2. 自定义指令实战——自定义 v-img 指令

【实战要求】

（1）学会引入 vue.global.js 文件，完成 HTML 文档基本结构的定义。

（2）学会定义 Vue 组件对象，并配置 data 和 directives 等选项。

（3）学会全局注册自定义指令 v-img。使用箭头函数定义指令相关参数(el、binding)。

（4）学会使用 el.style 对象的相关 DOM 属性，完成图像加载等样式设置。

【设计要求】

按如图 3-30 所示的页面效果，完成项目实战。具体设计要求如下。

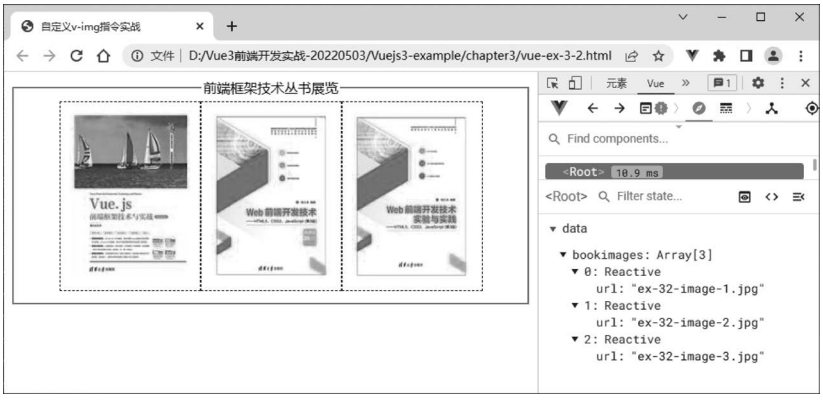

图 3-30　自定义 v-img 指令

（1）在< body >中插入一个< div >标记，作为视图展示区，其 id 为 app，< div >标记内包含一个域标记< fieldset >，域标记中包裹域标题标记和一个< div >标记，子< div >用于展示自定义 v-img 指令执行的图像加载等样式效果，采用 v-for 指令循环遍历 bookimages 对象数组。在< script >标记内定义组件 App 对象，并在其中定义 data 选项属性。定义 bookimages 数组，每个对象类似于{url："ex-32-image-1.jpg"}，3 个文件名分别为 ex-32-image-1.jpg、ex-32-image-2.jpg、ex-32-image-3.jpg。

（2）定义全局自定义指令 v-img。使用箭头函数定义 el.style 对象的相关属性，分别为 width、height、display、padding、border、backgroundImage、backgroundRepeat、backgroundPosition。

【实战步骤】

（1）建立 HTML 文件。将项目文件命名为 vue-ex-3-2.html，在< head >标记中引入 vue.global.js 文件，按设计要求(1)完成页面基本内容的设计。

（2）定义 Vue 组件。完成 data 选项的配置(定义 bookimages 对象数组)。

（3）全局注册自定义指令 v-img。部分参考代码如下。

```
app.directive("img", (el, binding) =>{
 el.style.width = 145 + "px";
 el.style.height = 202 + "px";
 …
 // 设置背景图像代码片段
 el.style.backgroundImage = "url(" + binding.value + ")";
 …  // 背景图像不重复,水平、垂直均居中
});
```

(4) 项目调试并运行。在浏览器中调试页面,效果如图 3-30 所示。