第2章 模板语法和内置指令

本章节我们将开始学习 Vue.js 3 模板语法和内置指令。在学习前，让我们先看看本章节的源代码管理方式。遵循第1章的规范，目录结构如下：

```
VueCode
├── chapter01
├── chapter02
|   └── js
|       └── Vue.js
|   └── 01_Mustache插值语法.html
|   └── ......
```

2.1 插值语法

在 React 中编写组件时，我们使用 JSX 语法，这是一种类似于JavaScript的语法。然后在使用Babel将JSX编译成React.createElement函数调用后，就能渲染出组件。当然，Vue.js 3也支持JSX语法开发模式（见第11节），但大多数情况下，Vue.js 3使用HTML模板（**template**）语法，通过声明式将组件实例的数据和DOM绑定在一起。

模板语法的核心是插值语法（**Mustache**）和指令。对于学习Vue.js 3来说，掌握模板语法非常重要。

下面我们来介绍一下插值语法。

在Vue.js 3中，想要将数据显示到模板中，常见的方式是使用插值语法，也称双大括号语法，如下代码所示：

```
<div>{{message}}</div>
```

插值语法还支持其他的写法。新建一个 `04_计数器案例-Vue3实现.html` 文件，演示插值语法的其他写法，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <!-- 1.mustache语法的基本使用 -->
    <h4>{{message}} - {{isShow}}</h4>
    <!-- 2.mustache语法包含一个表达式 -->
    <h4>{{counter * 10}}</h4>
    <h4>{{ message.split(" ").join("-") }}</h4>
    <!-- 3.mustache语法中调用方法 -->
    <h4>{{getReverseMessage()}}</h4>
    <!-- 4.mustache语法调用computed计算属性(先了解) -->
```

```
    <!-- 5.mustache语法支持三元运算符 -->
    <h4>{{ isShow ? "三元运算符": "" }}</h4>
    <button @click="toggle">切换控制显示</button>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          message: "Hello World",
          counter: 100,
          isShow: true
        }
      },
      methods: {
        getReverseMessage() {
          return this.message.split(" ").reverse().join(" ");
        },
        toggle() {
          this.isShow = !this.isShow;
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，在插值语法中不仅支持绑定data中的属性，还支持JavaScript表达式、调用方法，以及三元运算符。在浏览器中运行代码，效果如图2-1所示。

图2-1 Mustache语法

另外，以下是Mustache语法错误的写法，代码如下所示：

```
<!-- 错误写法一：不支持赋值语句 -->
<h4>{{var name = "Hello"}}</h2>
<!-- 错误写法二：不支持控制流程if语句，但是支持三元运算符 -->
<h4>{{ if (true) { return message } }}</h2>
```

2.2 基本指令

在`template`上除了Mustache语法外，还会经常看到标签中有`v-`开头的属性（**attribute**），这些`v-`开头的attribute被称为指令。

通常，指令带有`v-`前缀，它们是Vue.js 3提供的特殊属性（**attribute**）。它们会在渲染的`DOM`上应用特殊的响应式行为。例如，下面的`v-noce`指令用于指定元素或组件只渲染一次。

2.2.1 v-once

`v-once`指令用于指定元素或组件只渲染一次。当数据发生变化时，元素或组件以及其所有的子组件将视为静态内容，跳过更新。通常在需要进行性能优化时会使用`v-once`指令。

新建`02_基本指令-v-once.html`文件，在 template 中使用`v-once`指令，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <!-- 会重新渲染 -->
    <h2>h2 : {{counter}}</h2>
    <!-- 以下的只会渲染一次 -->
    <h3 v-once>h3: {{counter}}</h3>
    <div v-once>
      <h4>h4: {{counter}}</h4>
      <h5>h5: {{message}}</h5>
    </div>
    <!-- 单击按钮触发重新渲染 -->
    <button @click="increment">+1</button>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          counter: 100,
          message: "Hello Vue.js 3"
        }
      },
      methods: {
```

```
        increment() {
          this.counter++;
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，上面使用了`v-once`来绑定了`<h3>`和`<div>`元素。当单击"+1"时，只有`<h2>`会重新渲染，而`<h3>`、`<div>`、`<h4>`和`<h5>`元素都不会重新渲染。因为带有`v-once`绑定的元素及其子元素不会重新渲染。当然，如果`v-once`绑定的是组件，也同样适用。

在浏览器中运行代码，效果如图2-2所示。

图2-2 v-once

2.2.1 v-text

`v-text`指令用于更新元素的`textContent`。新建`03_基本指令-v-text.html`文件，在`template`中使用`v-text`指令，示例代码如下：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <h2 v-text="message"></h2>
    <!-- 上面写法等价于下面写法 -->
    <h3>{{message}}</h3>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
```

```
      data() {
        return {
          message: "Hello Vue.js 3"
        }
      }
    }

    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，在`<h2>`元素上使用了`v-text`指令将`message`变量进行绑定，在`<h3>`元素上使用了插值语法将`message`变量进行绑定。它们的效果是一样的，实际上`v-text`指令就相当于插值语法。

在浏览器中运行代码，效果如图2-3所示。

图2-3 v-text

2.2.2 v-html

当展示的内容是 HTML 字符串时，Vue.js 3 不会对其进行特殊的解析。如果希望 HTML 字符串的内容可以被 Vue.js 3 解析出来，可以使用`v-html`指令来展示。新建`04_基本指令-v-html.html`文件，在`<template>`中使用`v-html`指令，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <!-- 1.绑定字符串 -->
    <h2>{{message}}</h2>
    <!-- 2.绑定 HTML 字符串 -->
    <div v-html="message"></div>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          message: '<span style="color:red; background: blue;">你好呀
Vue3</span>'
```

```
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，先在`data`中定义了`message`变量，并赋值一段HTML字符串文本，然后在`template`中使用`v-html`把`message`绑定到`<div>`元素中，将`message`中的HTML字符串文本当成HTML网页来显示。

在浏览器中运行代码，效果如图2-3所示。

图2-4 v-html

2.2.3 v-pre

`v-pre`指令用于跳过元素及其子元素的编译过程，从而加快编译速度。新建`05_基本指令-v-pre.html`文件，在template中使用`v-pre`指令将`message`变量绑定到`<h2>`元素，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <h2 v-pre>{{message}}</h2>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          message: "Hello World"
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，在`data`中定义了`message`变量，然后在`template`中使用`v-pre`把`message`绑定到`<h2>`元素中。浏览器将Mustache语法当成字符串来显示，并不会显示`message`中的值，因为绑定`v-pre`指令后`<h2>`元素和它的子元素将跳过编译过程。

在浏览器中运行代码，效果如图2-5所示。

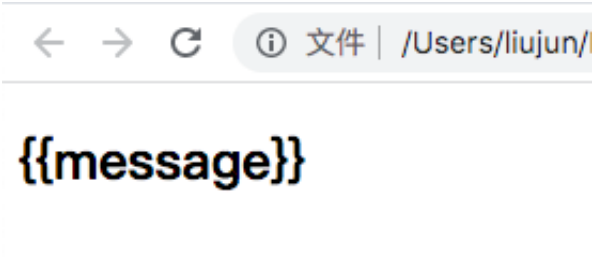

图2-5 v-pre

2.2.4 v-cloak

`v-cloak`指令可以隐藏未编译的 Mustache 语法的标签，直到组件实例完成编译。它需要和 CSS 规则一起使用，例如`[v-cloak] { display: none }`。

在Vue.js 3中，v-cloak指令使用频率不高，因为在生产阶段的模板已提前编译完成，所以不需要使用v-cloak指令。但在开发阶段实时进行模板编译时，v-cloak指令却非常有用，如果在开发阶段使用插值语法时，发现浏览器先显示Mustache语法，然后又立马显示正常（该问题仅会出现在性能较差的计算机上），就可以按照下面的方式使用v-cloak指令来处理。

新建`06_基本指令-v-cloak.html`文件，在template中使用`v-cloak`指令，代码如下所示：

```
<head>
  ......
  <style>
    /* 属性选择器，选中包含v-cloak属性的元素，如 h2 */
    [v-cloak] {
      display: none;
    }
  </style>
</head>
<body>
  <div id="app"></div>

  <template id="my-app">
    <!-- v-cloak指令会一直保留在元素上，等到该组件完成编译就会移除 -->
    <h2 v-cloak>{{message}}</h2>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
```

```
      data() {
        return {
          message: "Hello Vue3"
        }
      }
    }

    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，在`<h2>`元素中添加`v-cloak`指令，作用是使`<h2>`先不显示，直到模板编译结束后才显示。接着还需在`<style>`标签中添加 CSS 规则`*[v-cloak] { display: none }`。

在浏览器中运行代码，效果如图2-6所示。

图2-6 v-cloak

2.3 v-bind

前面介绍的指令是用来给元素绑定内容的，元素除了绑定内容之外，还要绑定各种各样的属性。此时，可以使用v-bind指令。下面将详细介绍v-bind指令的使用。

2.3.1 绑定基本属性

在很多时候，元素的属性也是动态的，比如`<a>`元素的`href`属性、`<img>`元素的`src`属性等，通常需要动态插入值，这时可以使用`v-bind`来绑定这些属性。

新建`07_v-bind的基本使用.html`文件，使用`v-bind`指令绑定元素的基本属性，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <!-- 1.v-bind的基本使用 -->
    <img v-bind:src="imgUrl" alt="">
    <a v-bind:href="link">百度一下</a>

    <!-- 2. 冒号语法绑定属性是v-bind指令的语法糖(即简写) -->
    <img :src="imgUrl" alt="" sty>

    <!-- 3.直接赋值“imgUrl”字符串给src属性 -->
    <img src="imgUrl" alt="">
  </template>
```

```
  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          imgUrl: "https://v2.cn.vuejs.org/images/logo.svg",
          link: "https://www.baidu.com"
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，在`data`属性中定义了`imgUrl`和`link`变量，`imgUrl`存放的是一张图片的路径，`link`存放的是一个网址。首先使用`v-bind`指令（`:`是v-bind的简写）把`imgUrl`变量绑定到第一个`<img>`元素的`src`属性上，接着把`link`变量绑定到`<a>`元素的`href`属性上。最后，第二个`<img>`元素的`src`属性赋值“`imgUrl`”字符串。

在浏览器中运行代码，效果如图2-7所示。

图2-7 v-bind 的基本使用

2.3.2 绑定class属性

`v-bind`绑定元素或组件的`class`属性，支持绑定字符串、对象和数组类型。

下面详细地讲解这三种类型的绑定。

1.绑定字符串类型

直接给 class 属性绑定一个字符串，代码如下所示：

```
<!-- 1.绑定字符串，这里的冒号语法是v-bind的简写 -->
<div :class="'abc'">class绑定字符串</div>
<div :class="className">class绑定字符串</div>
```

可以看到，直接给class绑定一个字符串`'abc'`，也可给class绑定一个字符串类型的className变量。

2.绑定对象类型

新建`08_v-bind绑定class-对象语法.html`文件，使用`v-bind`指令给class属性绑定对象，代码如下所示：

```
  <body>
    <div id="app"></div>
    <template id="my-app">
      <!-- 1.绑定字符串语法 -->
      <div :class="'abc'">class绑定字符串1</div>
      <div :class="className">class绑定字符串2</div>
      <!-- 2.绑定对象，支持 {'active': boolean} or { active: boolean} 写法 -->
      <div :class="{'active': isActive}">class绑定对象1</div>
      <!-- 3.对象可以有多个键值对 -->
      <div :class="{active: isActive, title: true}">class绑定对象2</div>
      <!-- 4.默认的class和动态的class结合使用 -->
      <div class="abc cba" :class="{active: isActive, title: true}">
        默认的class和动态的class结合
      </div>
      <!-- 5.将对象放到一个单独的class属性中 -->
      <div class="abc cba" :class="classObj">绑定属性中的对象</div>
      <!-- 6.将返回的对象放到一个methods(或computed)中 -->
      <div class="abc cba" :class="getClassObj()">绑定methods/computed返回的对
象</div>

      <button @click="toggle">切换isActive</button>
    </template>

    <script src="./js/vue.js"></script>
    <script>
      const App = {
        template: "#my-app",
        data() {
          return {
            className: "coderwhy",
            isActive: true,
            classObj: {
              active: true,
              title: true
            },
          };
        },
        methods: {
          toggle() {
            this.isActive = !this.isActive;
          },
          getClassObj() {
            return {
              active: false,
              title: true
            }
          }
        },
      };
```

```
    Vue.createApp(App).mount("#app");
  </script>
</body>
```

可以看到，该案例代码稍微多一点，但内容并不复杂。上面代码共演示了六种绑定class的情况。

1. 给class直接绑定abc字符串和绑定字符串类型的`className`变量。
2. 给class绑定了一个对象`{'active': isActive}`，如`isActive`变量为`true`时，该对象中`active`将会绑定到`div`的`class`上，否则不会。（注意：`'active'`单引号可有可无，但是如有短横杠连接字符时必须要单引号，例如`'active-link'`必须要单引号）
3. 给class绑定对象，对象可以有多个键值对。
4. 先给class直接赋值字符串，再给class绑定了对象。这时直接赋值的字符串会和该对象中值为`true`的`key`进行合并，再绑定到`div`的`class`上。
5. 与第四种一样，区别是绑定的对象被抽取到名为`classObj`的变量中。
6. 与第五种基本一样，区别是给class绑定了`methods`中`getClassObj`函数返回的对象。（注意：`class`也支持绑定`computed`中函数返回的对象，有关于计算属性会在第3章节讲。）

在浏览器中运行代码，效果如图2-8所示。

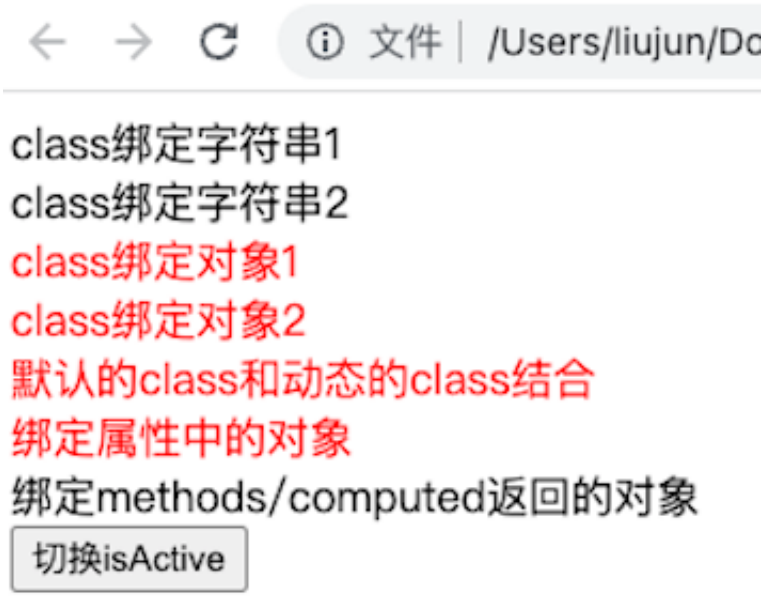

图2-8 v-bind绑定对象类型

3.绑定数组类型

新建`09_v-bind绑定class-数组语法.html`文件，用`v-bind`指令给class属性绑定数组，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <div :class="['abc', title]">v-bind绑定class-数组语法</div>
```

```
    <div :class="['abc', title, isActive ? 'active': '']">v-bind绑定class-数组语法</div>
    <div :class="['abc', title, {active: isActive}]">v-bind绑定class-数组语法</div>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          message: "Hello World",
          title: "cba",
          isActive: true
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，上面分别给三个`<div>`的`class`属性绑定了数组。

1. 第一个`<div>`元素的`class`属性绑定了一个字符串类型的数组。
2. 第二个`<div>`元素的`class`属性绑定的数组包含了三元运算符。
3. 第三个`<div>`元素的`class`属性绑定的数组包含了对象。

在浏览器中运行代码，效果如图2-9所示。

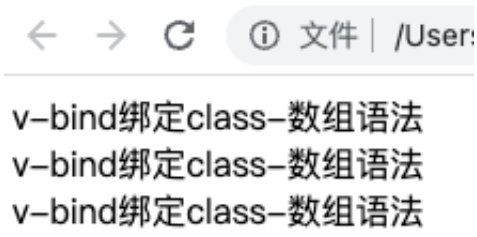

图2-9 v-bind绑定数组类型

2.3.3 绑定style属性

`v-bind`可以用于绑定元素或组件的`style`属性，支持绑定对象和数组类型。

下面详细地讲解这两种类型的绑定。

1.绑定对象类型

绑定对象的语法十分直观，例如：`:style="{color: 'red'}"`，对象中CSS属性的命名可采用Vue.js 3的约定，即小驼峰（camelCase）或短横线分隔（kebab-case）。其中，短横线分隔需用单引号括起来。

新建 `10-v-bind绑定style-对象语法.html` 文件，使用 `v-bind` 指令给style属性绑定对象，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <!-- 1.v-bind绑定style的基本用法 -->
    <div :style="{color: finalColor, 'font-size': '16px'}">v-bind绑定style-对象语法</div>
    <div :style="{color: finalColor, fontSize: '16px'}">v-bind绑定style-对象语法</div>
    <div :style="{color: finalColor, fontSize: finalFontSize + 'px'}">v-bind绑定style-对象语法</div>
    <!-- 2.绑定data中定义的对象类型变量 -->
    <div :style="finalStyleObj">绑定一个data中的属性</div>
    <!-- 3.绑定调用methods方法返回的对象 -->
    <div :style="getFinalStyleObj()">methods方法返回的对象</div>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          message: "Hello World",
          finalColor: 'red',
          finalFontSize: 16,
          finalStyleObj: {
            'font-size': '16px',
            fontWeight: 700,
            backgroundColor: '#ddd'
          }
        }
      },
      methods: {
        getFinalStyleObj() {
          return {
            'font-size': '16px',
            fontWeight: 700,
            backgroundColor: '#ddd'
          }
        }
      }
    }

    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，上面代码共演示了三种绑定style的情况。

1. 给`<div>`元素的`style`属性绑定了一个对象。对象中的key有多个单词时可用小驼峰或短横线分隔。
2. `key`的值如果是一个字符串，如`'16px'`是需要单引号。如果是变量，如`finalColor`时则不需要。
3. `key`的值支持表达式语法，如上面`<div>`元素的绑定的字体大小用了`finalFontSize + 'px'`表达式。
4. 给`<div>`元素的`style`属性绑定了一个对象类型的变量。
5. 给`<div>`元素的`style`属性绑定了一个返回对象类型的方法，也支持绑定`computed`。

在浏览器中运行代码，效果如图2-10所示。

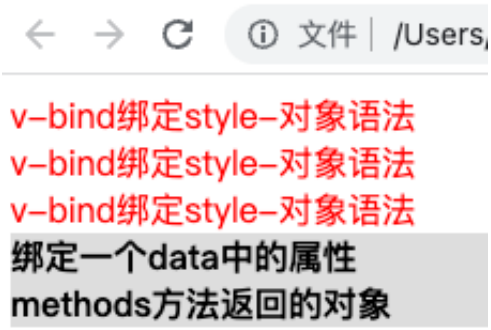

图2-10 v-bind 绑定对象类型

2.绑定数组

绑定数组的语法十分直观，例如：`:style="[{}, {}]"`，可将多个样式对象应用到同一个元素上。

新建`11_v-bind绑定style-数组语法.html`文件，使用`v-bind`指令给style属性绑定数组，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <div :style="[{color:'red', fontSize:'15px'}]">v-bind绑定style-数组语法
</div>
    <div :style="[style1Obj, style2Obj]">v-bind绑定style-数组语法</div>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          message: "Hello World",
          style1Obj: {
            color: 'red',
```

```
            fontSize: '16px'
          },
          style2Obj: {
            textDecoration: "underline"
          }
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，上面分别给两个`<div>`的`style`属性绑定了数组。

1. 第一个`<div>`元素的`style`属性绑定了一个数组，数组中每一项都是一个对象；
2. 第二个`<div>`元素的`style`属性绑定了一个数组，数组中每一项都是一个对象类型的变量。

在浏览器中运行代码，效果如图2-11所示。

图2-11 v-bind 绑定数组类型

2.3.4 动态绑定属性

前面介绍了如何给元素绑定内容和动态绑定属性，现在我们继续讲解如何动态绑定属性的名称和值，也就是属性的名称和值都是通过 JavaScript 表达式动态插入的。

新建`12_v-bind动态绑定属性名称和值.html`文件，使用 v-bind 绑定动态属性的名称和值，代码如下所示：

```
<body>
  <div id="app"></div>

  <template id="my-app">
    <!-- : 是 v-bind的简写 -->
    <div :[name]="value">v-bind动态绑定属性名称和值</div>
    <!-- 上面等价于下面的写法 -->
    <div v-bind:[name]="value">v-bind动态绑定属性名称和值</div>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
```

```
      data() {
        return {
          name: "username",
          value: "kobe"
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，上面代码给`<div>`元素绑定的属性名称和值都是动态。先在`data`中定义了`name`和`value`两个变量，`name`存放属性的名称，值为`username`字符串；`value`存放属性名称对应的值，值为`kobe`字符串。然后用`:key="value"`的语法，就可以实现动态绑定属性名称和值。

在浏览器中运行代码，效果如图2-12所示。

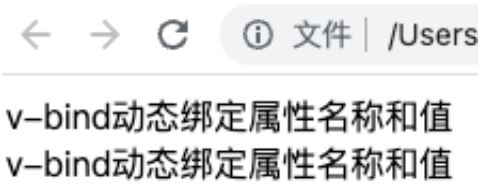

图2-12 v-bind 动态绑定属性

2.3.5 绑定一个对象

`v-bind`指令不仅可以绑定单个属性，还可以使用`v-bind`直接绑定一个对象，从而实现一次批量绑定多个属性。

新建`13_v-bind动态绑定一个对象.html`文件，使用 v-bind 直接绑定一个对象，代码如下所示：

```
<body>
  <div id="app"></div>

  <template id="my-app">
    <div v-bind="info">v-bind动态绑定一个对象</div>
    <!-- 下面是上面的简写，阅读性不好，不推荐 -->
    <div :="info">v-bind动态绑定一个对象</div>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
```

```
          info: {
            name: "why",
            age: 18,
            height: 1.88
          }
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，在`data`中定义了一个对象类型的`info`变量，接着在`template`中使用`v-bind`指令将`info`绑定到`div`元素中，`info`对象中的键值对会被拆解成`div`的各个属性，并绑定到`div`元素上。

在浏览器中运行代码，效果如图2-13所示。

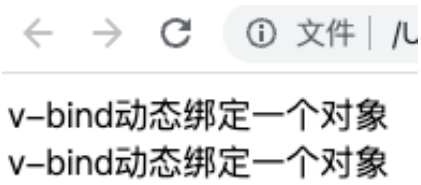

图2-13 v-bind 绑定一个对象

2.4 v-on

前面已经介绍了元素内容和属性的绑定。在前端开发中，交互是非常重要的一部分，例如处理单击、拖拽、键盘事件等。在Vue.js 3中，使用v-on指令可以实现对这些事件的监听。

v-on指令的语法如下：

- **指令的简写**：`@`
- **指令值类型**：`Function | Inline Statement | Object`
- **参数**：`event`
- **指令修饰符包括**：
  - `.stop` - 调用`event.stopPropagation()`。
  - `.prevent` - 调用`event.preventDefault()`。
  - `.capture` - 添加事件侦听器时使用 capture 模式。
  - `.self` - 只有事件是从侦听器绑定的元素本身触发时才触发回调。
  - `.{keyAlias}` - 仅当事件是从特定键触发时才触发回调。
  - .once - 只触发一次回调。
    - .left - 只有在单机鼠标左键时才会触发回调。

        - .right - 只有在单机鼠标右键时才会触发回调。
        - .middle - 只有在单机鼠标中键时才会触发回调。
    - `.passive` - 使用`{ passive: true }`模式添加侦听器。

- **指令用法**：绑定事件监听器。

下面来看看 v-on 指令的实际使用。

2.4.1 绑定事件

在开发中，经常需要处理如单击、拖拽等操作。在Vue.js 3中我们可以用v-on指令来绑定所需的事件并进行监听。

新建`14_v-on的基本使用-绑定事件.html`文件，使用 v-on绑定事件 ，代码如下所示：

```
<head>
  ......
  <style>
    .area {
      width: 100px;
      height: 100px;
      background: #ddd;
      margin-bottom: 4px;
    }
  </style>
</head>
<body>
  <div id="app"></div>
  <template id="my-app">
    <!-- 1.绑定事件完整写法，v-on:监听的事件="methods中方法" -->
    <button v-on:click="btn1Click">监听按钮单击(完整写法)</button>
    <div class="area" v-on:mousemove="mouseMove">监听鼠标移动事件</div>
    <!-- 2. @ 是 v-on 的语法糖 -->
    <button @click="btn1Click">监听按钮单击(简写)</button>
    <!-- 3.绑定一个表达式: inline statement -->
    <button @click="counter++">单击+1:  {{counter}}</button>
    <!-- 4.绑定一个对象 -->
    <div class="area" v-on="{click: btn1Click, mousemove: mouseMove}">监听鼠标移
动事件</div>
    <!-- 5.是上面v-on的简写，阅读性不好，不推荐 -->
    <div class="area" @="{click: btn1Click, mousemove: mouseMove}">监听鼠标移动事
件（简写） </div>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
```

```
        return {
          message: "Hello World",
          counter: 100
        }
      },
      methods: {
        btn1Click() {
          console.log("按钮1发生了单击");
        },
        mouseMove() {
          console.log("鼠标移动");
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，上面代码共演示了五种 v-on 的使用情况：

1. 用`v-on`监听了`<button>`元素的单击事件，当单击`<button>`时，会触发`btn1Click`函数的回调；同时还用`v-on`监听了`<div>`元素上鼠标移动事件，当鼠标在该`<div>`元素上移动时，会触发`mouseMove`函数的回调。
2. 用`v-on`的语法糖冒号`@`来监听了`<button>`元素的单击事件。
3. 用`v-on`监听了`<button>`元素的单击事件，但是这次绑定的是一个表达式，而不是回调函数，当单击`<button>`时，会触发`counter`变量自增。
4. 用`v-on`绑定一个对象，对象中的key是监听事件的名称，key对应的值是事件的回调函数。该对象中包含了单击`click`事件和鼠标移动`mousemove`事件。
5. 用`v-on`的简写`@`语法来绑定一个对象，阅读性不好，不推荐该写法。

注意：为了方便演示，上面所有回调函数都用了`btn1Click`和`mouseMove`这两个函数。

在浏览器中运行代码，效果如图2-14所示。

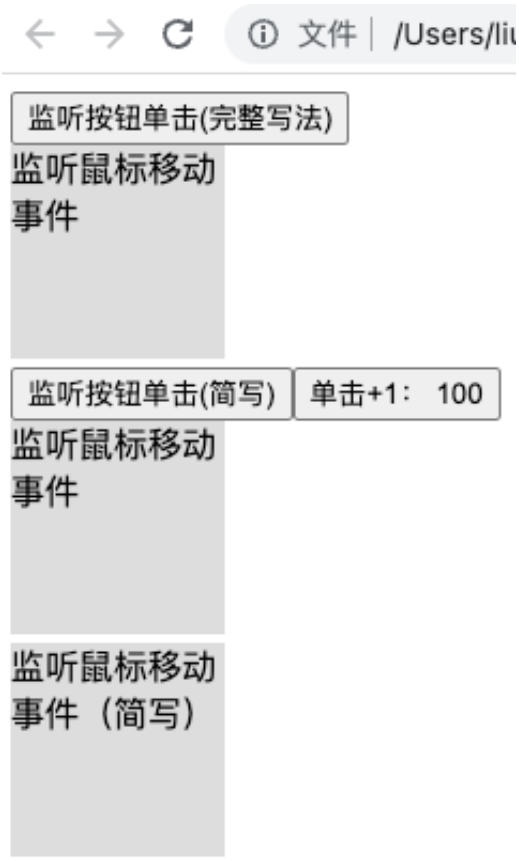

图2-14 v-on 的基本使用

2.4.2 事件对象和传递参数

事件在发生时会产生事件对象，可以在事件回调函数中获取**事件对象**。新建`15_v-on事件对象和参数传递.html`文件，使用`v-on`指令获取事件对象和参数传递，代码如下所示：

```
<body>
  <div id="app"></div>

  <template id="my-app">
    <!-- 1.Vue.js内部会自动传入event对象，可在方法中接收 -->
    <button @click="btn1Click">自动传入event对象</button>
    <!-- 2.手动传入事件对象：$event。$event是固定的语法 -->
    <button @click="btn2Click($event, 'coderwhy', 18)">手动传入event对
象</button>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          message: "Hello World"
```

```
        }
      },
      methods: {
        btn1Click(event) {
          console.log(event); // 打印自动传入的事件对象
        },
        btn2Click(event, name, age) {
          console.log(event, name, age); // 打印事件对象和其他传入的参数
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，这里编写了两个`<button>`元素，并设置了单击事件。

1. 第一个`<button>`元素监听了单击事件，单击时会触发`btn1Click`函数调用。如果没有手动传递参数，Vue.js 3会自动将`event`对象传递给`btn1Click`函数，所以该函数可以接收`event`事件对象参数。
2. 第二个`<button>`标签也监听了单击事件，单击时会触发`btn2Click`函数调用。这次手动传递了三个参数：`$event`、`"coderwhy"`和`18`。所以在该函数中可以接收这三个参数。

需要注意的是：`$event`是固定写法，专门用来获取事件对象。另外，该参数的位置不是固定的。

在浏览器中运行代码，分别单击两个按钮，效果如图2-15所示。

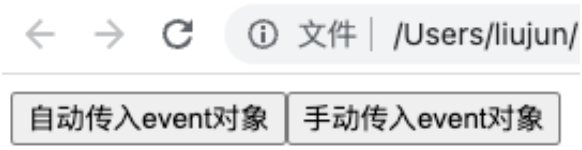

图2-15 v-on 获取事件对象

2.4.3 修饰符

事件冒泡，在JavaScript中可通过`event.stopPropagation()`来阻止；在`Vue.js 3`中可使用`v-on`指令的`.stop`修饰符来阻止事件冒泡（注意：有关于v-on指令的更多修饰符，可看本章2.4小节）。

新建`16_v-on绑定事件时添加修饰符.html`文件，使用v-on指令的`.stop`和`.enter`修饰符，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <div
      @click="divClick"
      :style="{width: '100px', 'height': '65px',backgroundColor:'#ddd'}"
    >
```

```
      div
      <button @click.stop="btnClick">button按钮</button>
    </div>
    <input type="text" @keyup.enter="enterKeyup">
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          message: "Hello World"
        }
      },
      methods: {
        divClick() {
          console.log("divClick");
        },
        btnClick() {
          console.log('btnClick');
        },
        enterKeyup(event) {
          console.log("keyup", event.target.value);
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，上面代码共使用了`v-on`的`.stop`和`.enter`两个修饰符。

- `stop`修饰符的应用：给`<div>`和`<button>`两个元素都绑定了单击事件，单击`<button>`时，因为事件会冒泡，所以`btnClick`和`divClick`函数都会被触发。如果想在单击`<button>`时，只触发`btnClick`函数，需要在`<button>`的`@click`后加上`.stop`修饰符。这时再单击`<button>`，事件就不会继续冒泡到`<div>`上。
- `enter`修饰符的应用：给`<input>`元素监听了`keyup`键盘抬起事件，每输入一个字符，就会触发一次`enterKeyup`函数。如果想当用户按`Enter`键时再触发`enterKeyup`函数，需要在`@keyup`后加上`.enter`修饰符。这时在`<input>`中输入值，只有当按`Enter`键才会触发`enterKeyup`函数回调。

在浏览器中运行代码，单击`<button>`只会触发`btnClick`函数回调，在`<input>`输入`coderwhy`，按`Enter`键后才会触发`enterKeyup`函数回调，效果如图2-16所示。

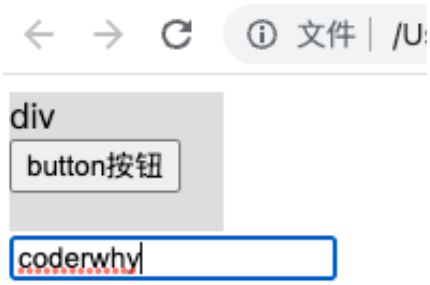

图2-16 v-on 修饰符

2.5 条件渲染

在前端开发中，有时需要根据当前条件来决定是否渲染特定的元素或组件。Vue.js 3 提供了 v-if、v-else、v-else-if 和 v-show 指令来实现条件判断。

2.5.1 v-if和v-else等

1.v-if指令的使用

`v-if`指令用于根据条件来渲染某一块内容。该指令是惰性的，当条件为false时，判断的内容完全不会被渲染或被销毁；当条件为true时，才会真正渲染条件块中的内容。

新建`17_v-if条件渲染的基本使用.html`文件，使用v-if来渲染某一快内容，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <h2 v-if="isShow">v-if条件渲染的基本使用</h2>
    <button @click="toggle">单击切换显示和隐藏内容</button>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          isShow: true
        }
      },
      methods: {
        toggle() {
          this.isShow = !this.isShow;
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
```

```
</body>
```

可以看到，上面的`<h2>`元素使用`v-if`指令绑定了`isShow`变量，如果`isShow`为`true`时显示`<h2>`元素，为`false`时则隐藏该元素。接着通过单击`<button>`改变`isShow`的值，来控制`<h2>`元素的显示和隐藏。

在浏览器中运行代码，单击`<button>`时`<h2>`元素会隐藏，再单击便会显示，效果如图2-17所示。

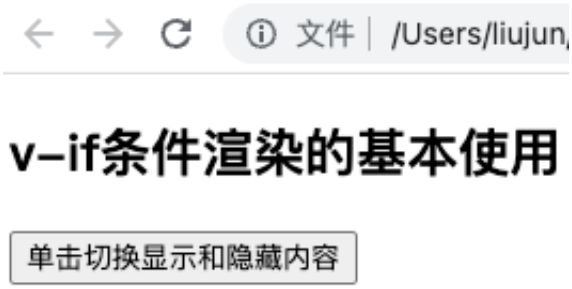

图2-17 v-if 的基本使用

2.v-if、v-else、v-else-if一起使用

`v-if`、`v-else`、`v-else-if`指令都是用于根据条件来渲染某一块的内容。只有在条件为`true`时，对应的内容才会被渲染出来。需要注意的是，这三个指令类似于 JavaScript 的条件语句`if`、`else`、`else if`。

新建`18_v-if多个条件的渲染.html`文件，演示这三个指令的使用，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <input type="text" v-model="score">
    <h2 v-if="score >= 90">优秀</h2>
    <h2 v-else-if="score >= 60">良好</h2>
    <h2 v-else>不及格</h2>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          score: 95
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，首先在data属性中定义了score变量，并将其默认值设为95分。随后，在`<input>`元素上使用`v-model`指令来双向绑定score变量，把`<input>`输入的数据绑定到score变量中（`v-model`是Vue.js 3中用于实现表单输入和应用程序状态之间双向绑定的重要指令，该指令会在第4章单独讲解）。

接着，在`<h2>`元素上使用`v-if`指令绑定表达式`score >= 90`，如果分数大于或等于90分，则显示“优秀”。然后使用`v-else-if`指令绑定表达式`score >= 60`，如果分数大于或等于60分，则显示“良好”。最后，使用`v-else`指令来处理分数小于60的情况，显示“不及格”。

在浏览器中运行代码，在`<input>`元素中依次输入：95，80，50分时，分别显示**优秀**，**良好**，**不及格**，效果如图2-18所示。

图2-18 v-if 多个条件渲染

2.5.2 v-if和template结合使用

使用`v-if`指令时必须添加到某一个元素上，例如`<div>`元素。但如果希望显示和隐藏多个元素，常见有两种实现方式。

1. 用`<div>`元素包裹多个元素，然后使用v-if指令来控制该`<div>`元素的显示和隐藏即可。缺点是`<div>`元素也会被渲染。
2. 用 HTML5 的`<template>`元素包裹多个元素，然后使用v-if指令来控制`<template>`元素的显示和隐藏即可。优点是`<template>`元素不会被渲染出来（推荐）。

以下是`v-if`和`<template>`元素的结合使用。新建`19_v-if和template结合使用.html`文件，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <template v-if="isShow">
      <h3>v-if控制多个h3标签显示隐藏</h3>
      <h3>v-if控制多个h3标签显示隐藏</h3>
      <h3>v-if控制多个h3标签显示隐藏</h3>
    </template>

    <template v-else>
      <h4>v-if控制多个h4标签显示隐藏</h4>
      <h4>v-if控制多个h4标签显示隐藏</h4>
      <h4>v-if控制多个h4标签显示隐藏</h4>
    </template>
  </template>
```

```
  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
            isShow: true
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，上面代码分别使用了`<template>`元素包含了3个`<h3>`元素和3个`<h4>`元素，接着在`<template>`元素上分别绑定了`v-if`和`v-else`指令。如果`isShow`为`true`时，会显示3个`<h3>`元素，否则显示3个`<h4>`元素。

在浏览器中运行代码，先将`isShow`设为 true，然后设为 false，效果如图2-19所示。

图2-19 v-if 和 template 结合使用

2.5.3 v-show

`v-show`指令也是用来控制显示和隐藏某一块内容，用法和`v-if`指令一致。

新建`20_v-show条件渲染.html`文件，使用v-show来显示和隐藏某一快内容，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <h4 v-show="isShow">isShow条件渲染基本使用</h4>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
```

```
      data() {
        return {
          isShow: true
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，上面代码直接在`<h4>`元素上使用`v-show`指令来控制该元素的显示和隐藏。当`isShow`为`true`时，显示该元素，否则隐藏。

2.5.4 v-show和v-if的区别

前面介绍的`v-if`和`v-show`都可用来控制显示和隐藏某一块内容，下面介绍一下它们的区别。

- `v-show`不支持在`<template>`标签上使用。
- `v-show`不可与`v-else`一起使用。
- `v-show`控制的元素无论是否需要显示到浏览器上，它的DOM都会渲染。本质是通过CSS的`display`属性来控制显示和隐藏。
- 当`v-if`的条件为false时，对应的元素是不会被渲染到DOM中的。

在开发过程中的选择如下。

- 如元素需要在显示和隐藏之间频繁切换，则使用`v-show`。
- 如不需频繁切换显示和隐藏，则使用`v-if`。

新建`21_v-show和v-if的区别.html`文件，演示一下v-show和v-if区别，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <h2 v-if="isShow">v-if控制显示和隐藏</h2>
    <h2 v-show="isShow">v-show控制显示和隐藏</h2>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          isShow: false
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
```

```
</body>
```

可以看到，两个`<h2>`元素分别使用`v-if`和`v-show`来控制显示和隐藏。当`isShow`为`false`时，第一个`<h2>`不会被渲染到DOM中，而第二个`<h2>`通过CSS的`display`属性来控制显示和隐藏。

在浏览器中运行代码，效果如图2-20所示。

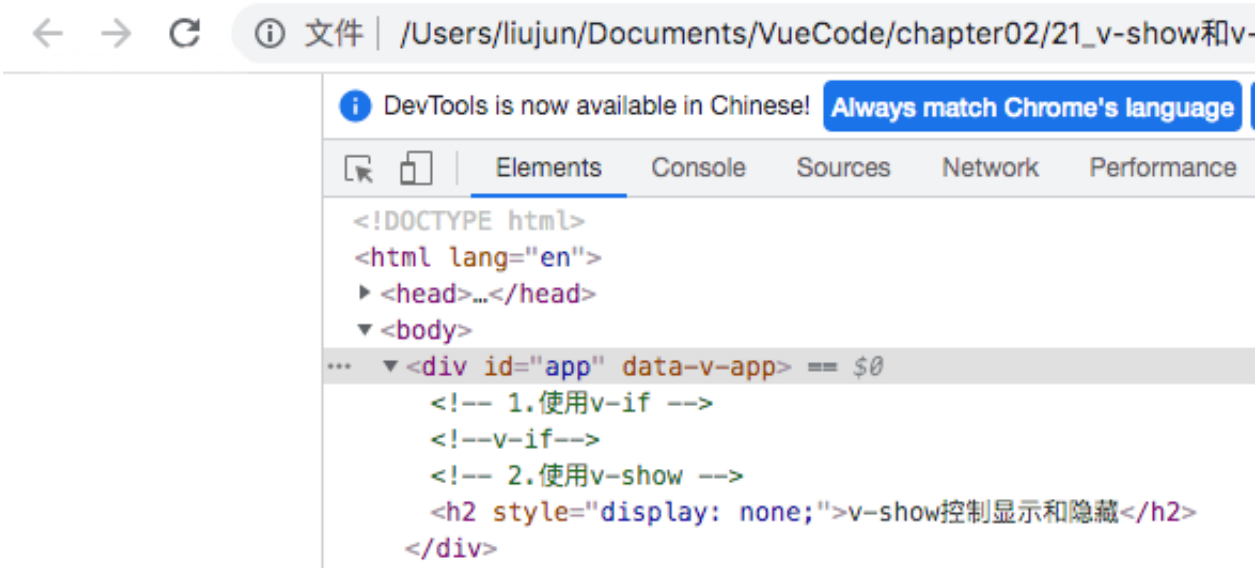

图2-20 v-show和v-if区别

2.6 列表渲染

在真实的开发中，通常需要从服务器获取一组数据并渲染到页面上。这时可以使用Vue.js 3中的v-for指令来实现。v-for指令类似于JavaScript中的for循环，可以用于遍历一组数据并将每个元素渲染到页面上。下面我们来看一下v-for指令的使用（注意：在React中，渲染一组数据通常使用map函数）。

2.6.1 v-for的基本使用

在 Vue.js 3 中，使用 v-for 指令语法的方式为：`v-for="item in 数组"` 或 `v-for="(item, index) in 数组"`。

- 数组：通常来自于 data 或 prop，也可以来自 methods 和 computed。
- item：可以给数组中每一项元素起一个别名，这个别名可以自行命名。
  - item在循环过程中代表当前遍历到的数组元素。
- index：表示当前元素在数组中的索引位置。

新建`22_v-for的基本使用.html`文件，演示`v-for`指令的使用，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <h4>1.电影列表（遍历数组[]）</h4>
    <ul>
      <li v-for="movie in movies">{{movie}}</li>
    </ul>
```

```
    <h4>2.电影列表（遍历数组[]）</h4>
    <ul>
      <li v-for="(movie, index) in movies">{{index+1}}.{{movie}}</li>
    </ul>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          movies: [
            "星际穿越",
            "盗梦空间",
            "少年派"
          ]
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，先在`data`中定义一个`movies`数组变量存放三部电影名称。接着，在两个`<li>`元素使用了`v-for`指令来遍历`movies`数组。

1. 在第一个`<li>`元素中，使用`v-for`指令来遍历`movies`数组，在`in`操作符前面的`movie`便可拿到数组中每一项内容。最后用插值语法将`movie`绑定在`<li>`元素中（注意：`v-for`遍历数组时除了可用`in`操作符，也支持使用`of`操作符）。
2. 在第二个`<li>`元素中，在遍历`movies`时，我们在`in`操作符前拿到了数组中每一项内容`movie`和索引`index`，但是必须要用括号括起来。最后，使用插值语法将`movie`和`index`绑定在`<li>`元素中。

需要注意的是：`in`操作符前面的参数顺序不能互换，但参数名称可自行命名。例如上面`movie`和`index`可自行命名。

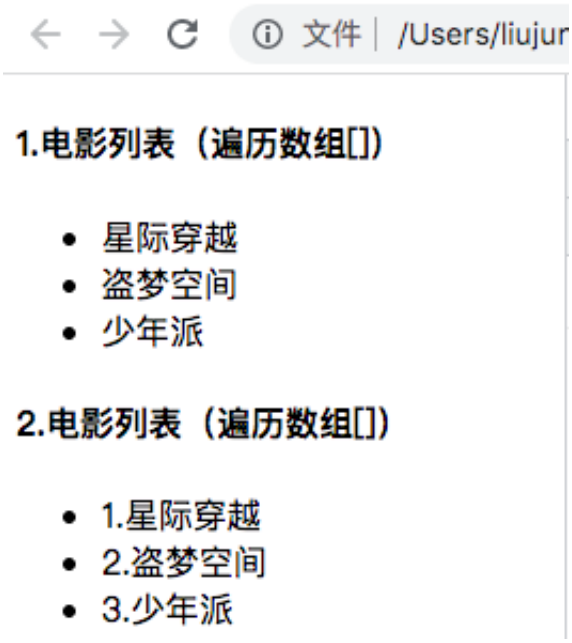

图2-21 v-for 的基本使用

2.6.2 v-for支持的类型

`v-for` 指令不仅支持数组的遍历，也支持对象类型和数字类型的遍历。具体使用方式如下。

1. **对象类型遍历：**

- 遍历值： `v-for="value in object"`
- 遍历键值对： `v-for="(value, key) in object"`
- 遍历键值对及索引： `v-for="(value, key, index) in object"`

2. **数字类型遍历：**

- 遍历值： `v-for="value in number"`
- 遍历值及索引： `v-for="(value,index) in number"`

新建 `23_v-for支持的类型.html` 文件，使用 `v-for` 遍历对象和数字类型，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <h4>1.个人信息（遍历对象{}）</h4>
    <ul>
      <li v-for="(value, key, index) in info">{{value}}-{{key}}-{{index}}</li>
    </ul>
    <h4>2.遍历数字number</h4>
    <ul>
      <li v-for="(num, index) in 3">{{num}}-{{index}}</li>
    </ul>
  </template>
```

```
  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          info: {
            name: "why",
            age: 18,
            height: 1.88
          }
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，先在`data`中定义一个`info`对象变量，该对象有`name`、`age`和`height`三个属性。

- 接着，在第一个`<li>`元素中使用`v-for`指令来遍历`info`对象。在`in`操作符前面拿到对象中的每一个键值对`value`、`key`和索引`index`，并使用插值语法将`value`、`key`和`index`绑定在`<li>`元素中。
- 然后，在第二个`<li>`元素中使用`v-for`指令来遍历数字3。在`in`操作符前面拿到从1数到3的每一个数字`num`和索引`index`，并使用插值语法将`num`和`index`绑定在`<li>`元素中。

同样，这里`in`操作符前面的参数顺序也不能互换，但参数名称可自行命名。

在浏览器中运行代码，效果如图2-22所示。

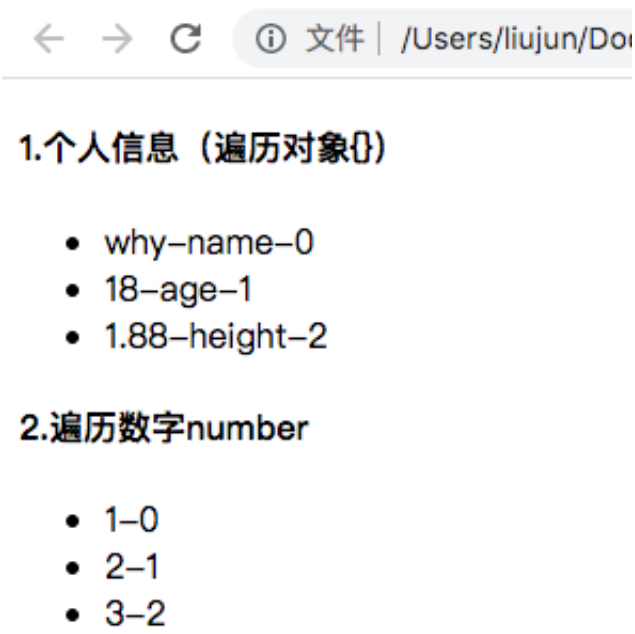

图2-22 v-for 遍历对象和数字

2.6.3 v-for和template结合使用

类似于`v-if`，`v-for`同样可以使用`<template>`元素来循环渲染一段包含多个元素的内容（注意：这里的`<template>`元素也是不会被渲染出来）。

新建`24_v-for和template结合使用.html`文件，使用template元素对多个元素进行包裹，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <ul>
      <template v-for="(value, key) in info">
        <li>{{key}}</li>
        <li>{{value}}</li>
        <!-- 下面是实现一条分割线 -->
        <li :style="{height:'2px', backgroundColor:'#ddd'}"></li>
      </template>
    </ul>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          info: {
            name: "why",
            age: 18,
            height: 1.88
          }
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，在`<template>`元素上使用`v-for`指令来遍历`info`对象，然后使用插值语法将遍历对象的`value`和`key`绑定到`<li>`元素上。

在浏览器中运行代码，效果如图2-23所示。

图2-23 v-for 和 template结合使用

2.6.4 数组的更新检测

在`data`中定义的变量属于响应式变量，修改这些变量时会自动触发视图的更新。对于定义为数组类型的响应式变量，在调用`filter()`、`concat()`和`slice()`方法时不会触发视图更新，而调用`push()`、`pop()`等方法则会。

下面详细介绍一下数组更新检测的规则。

1.调用数组的变更方法

在Vue.js 3中，系统会对被侦听的数组的变更方法进行封装，因此，当执行数组变更方法时，系统会自动触发视图更新。这些被封装的方法包括：`push()`、`pop()`、`shift()`、`unshift()`、`splice()`、`sort()`和`reverse()`。

新建`25-数组更新的检测.html`文件，演示数组更新检测，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <h4>电影列表</h4>
    <ul>
      <li v-for="(movie, index) in movies">{{index+1}}.{{movie}}</li>
    </ul>
    <input type="text" v-model="newMovie">
    <button @click="addMovie">添加电影</button>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          newMovie: "",
          movies: [
            "星际穿越",
```

```
            "盗梦空间"
          ]
        }
      },
      methods: {
        addMovie() {
          this.movies.push(this.newMovie);
          this.newMovie = "";
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，在 `data` 中定义了一个 `movies` 数组。在 `<li>` 元素中使用 `v-for` 指令遍历该数组，并用插值语法将数组中的每一项内容和索引绑定到 `<li>` 元素中。

接着，在 `<input>` 元素中使用 `v-model` 指令绑定 `newMovie` 变量，将 `<input>` 元素输入的值绑定到 `newMovie` 变量中。当单击 `<button>` 按钮时，会触发 `addMovie` 函数回调。该函数调用 `movies` 数组的 `push` 方法，将 `<input>` 元素输入的值 `this.newMovie` 添加到 `movies` 数组中（调用 `push` 方法修改数组会触发视图更新）。最后，清空 `<input>` 的输入。

另外需要注意的是：这里只演示了 `push` 方法，调用上面提到的 `pop`、`shift` 等方法也会触发视图更新。

在浏览器中运行代码，在输入框中输入“流浪地球”后，单击“添加电影”，电影列表就会自动刷新，效果如图2-23所示。

图2-24 数据更新检测

2.新数组替换旧数组

数组除了以上提到的方法外，常用的还有 `filter()`、`concat()` 和 `slice()` 方法。调用这些方法时默认不会触发视图更新，因为它们返回的是一个新的数组。只有将这个新数组用于替换原来的数组时，才可以触发视图更新。

新建`26-数组更新的检测-替换旧数组.html`文件，用新数组替换旧数组来触发视图更新，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <h4>电影列表</h4>
    <ul>
      <li v-for="(movie, index) in movies">{{movie}}</li>
    </ul>
    <button @click="showTopThreeMovie">显示前三名电影</button>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          movies: [
            "1.星际穿越",
            "2.盗梦空间",
            "3.大话西游",
            "4.教父",
            "5.少年派"
          ]
        }
      },
      methods: {
        showTopThreeMovie() {
          this.movies = this.movies.filter((movie, index) => index < 3 )
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，当单击`<button>`按钮时会触发`showTopThreeMovie`函数回调，该函数调用了`movies`数组的`filter`方法，并将该方法返回的新数组赋值给`movies`变量。当`movies`变量的值被替换之后，便会触发视图更新。

在浏览器中运行代码，当单击"显示前三名电影"按钮时，电影列表会自动刷新，效果如图2-25所示。

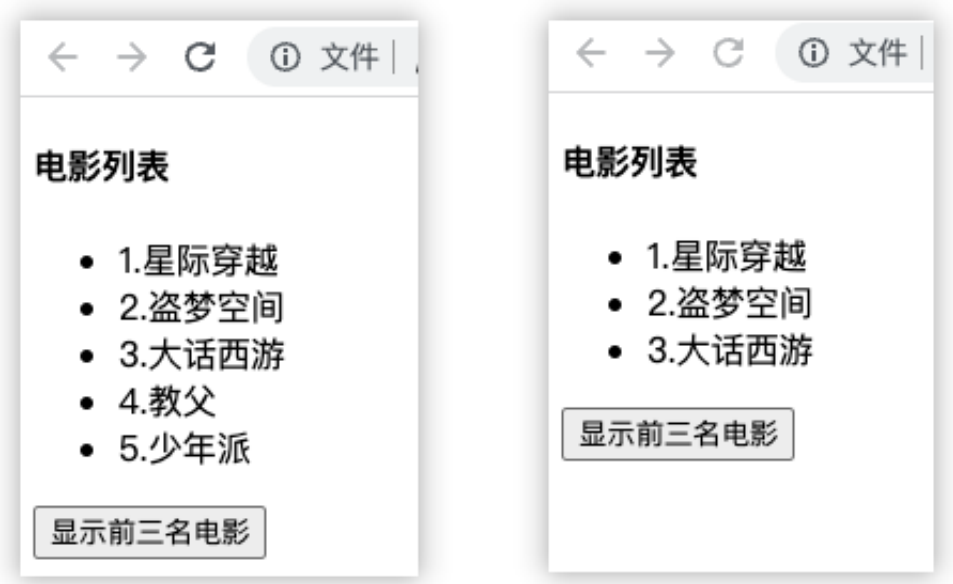

图2-25 数组更新检测

2.7 key和diff算法

在Vue.js 3中，使用v-for进行列表渲染时，官方建议给元素或组件绑定一个key属性。这样做的主要目的是为了更好地进行diff算法，并确定需要删除、新增、位置移动的元素。由于Vue.js 3会根据每个元素的key值判断此类操作，因此key的作用非常重要。合适的key可以提高DOM更新速度，同时减少不必要的DOM操作。

需要注意的是，key的值可以是number或string类型，但必须保证其唯一性。

2.7.1 认识VNode和VDOM

初学者对于key的解析可能会对以下问题感到困惑：

- 新旧nodes是什么？VNode又是什么？
- 如果没有key，怎样才能尝试修改和复用节点？
- 如果有key，如何按照key重新排列节点？

Vue.js 3官网对key属性作用具体解释如下。

- key属性主要用在 Vue.js 3 虚拟DOM算法中。在新旧节点（nodes）对比时，用于辨识VNodes。
- 如果不用key属性，Vue.js 3会尝试使用一种算法，最大限度减少动态元素并尽可能就地修改或复用相同类型的元素。
- 如果用了key属性，Vue.js 3将根据key属性的值重新排列元素的顺序，并移除或销毁那些不存在key属性的元素。

为了更好的理解key的作用，先介绍一下VNode的概念。

- VNode的全称是Virtual Node，也就是虚拟节点。
- 事实上，无论是组件还是元素，它们最终在Vue3中表示出来的都是一个个VNode。
- VNode本质是一个JavaScript的对象，如下代码所示。

```
<div class="title" style="font-size: 30px; color: red;">哈哈哈</div>
```

上面的`<div>`元素在Vue.js 3中会被转化，并创建出一个`VNode`对象：

```
const vnode = {
  type: 'div',
  props: {
    'class': 'title',
    style: {
      'font-size': '30px',
      color: 'red'
    }
  },
  children: '哈哈哈'
}
```

在Vue.js 3内部获取到VNode对象后，会对该对象进行处理，并将其渲染成真实的DOM。 具体渲染过程如图2-26所示。

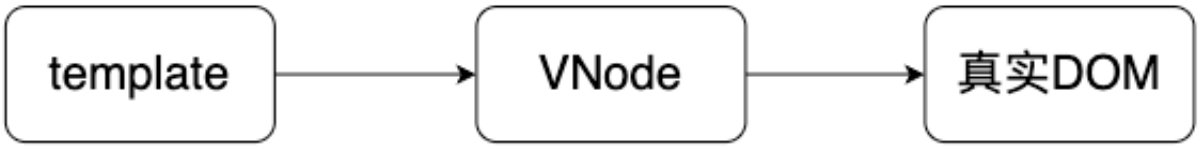

图2-26 Vue.js 3 DOM渲染过程

如果页面不仅仅是一个简单的`<div>`，而是包含大量元素的话，如下代码所示：

```
<div>
  <p>
    <i>哈哈哈哈</i>
    <i>哈哈哈哈</i>
  </p>
  <span>嘻嘻嘻嘻</span>
  <strong>呵呵呵呵</strong>
</div>
```

上面的代码会形成一个 **VNode Tree**（也称 **Virtual DOM**），如 图2-27 所示。

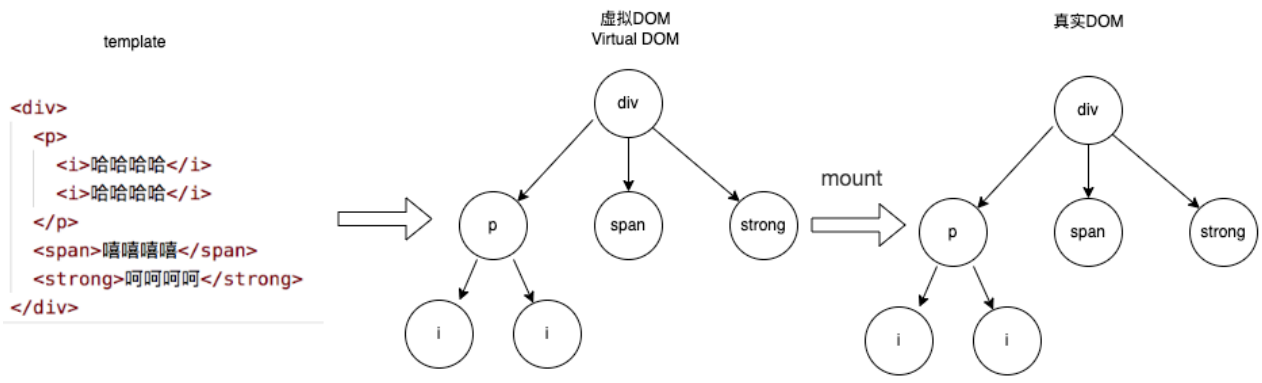

图2-27 Virtual DOM

2.7.2 key的作用和diff算法

介绍了Vue.js 3中的VNode（虚拟节点）和Virtual DOM（虚拟DOM）后，接下来我们需要具体分析它们与key之间的关系。

先看一个例子，单击 "button" 按钮，会在列表中间插入一个 "f" 字符串。新建`27_v-for中key的案例-插入f元素.html`文件，代码如下所示：

```
<body>
  <div id="app"></div>
  <template id="my-app">
    <ul>
      <li v-for="item in letters" :key="item">{{item}}</li>
    </ul>
    <button @click="insertF">插入F元素</button>
  </template>

  <script src="./js/vue.js"></script>
  <script>
    const App = {
      template: '#my-app',
      data() {
        return {
          letters: ['a', 'b', 'c', 'd']
        }
      },
      methods: {
        insertF() {
          this.letters.splice(2, 0, 'f')
        }
      }
    }
    Vue.createApp(App).mount('#app');
  </script>
</body>
```

可以看到，在`<li>`元素中使用`v-for`指令遍历`letters`数组来展示"a"、"b"、"c"、"d"字符，并给`<li>`元素绑定了`key`属性，`key`对应的值是数组的每一项（`key`的值要保证唯一）。

接着，当单击`<button>`时，会回调`insertF`函数，然后在该函数中调用`letters`数组的`splice`方法在索引为2处插入"f"，将会触发视图更新。

下面我们来分析一下Vue.js 3列表更新的原理。

- Vue.js 3会根据列表项有没有`key`，调用不同的方法来更新列表。
- 有`key`，调用`patchKeyedChildren`方法更新列表。如图2-28所示，第 1621 行代码。
- 没有`key`，调用`patchUnkeyedChildren`方法更新列表。如图2-28所示，第 1635 行代码。

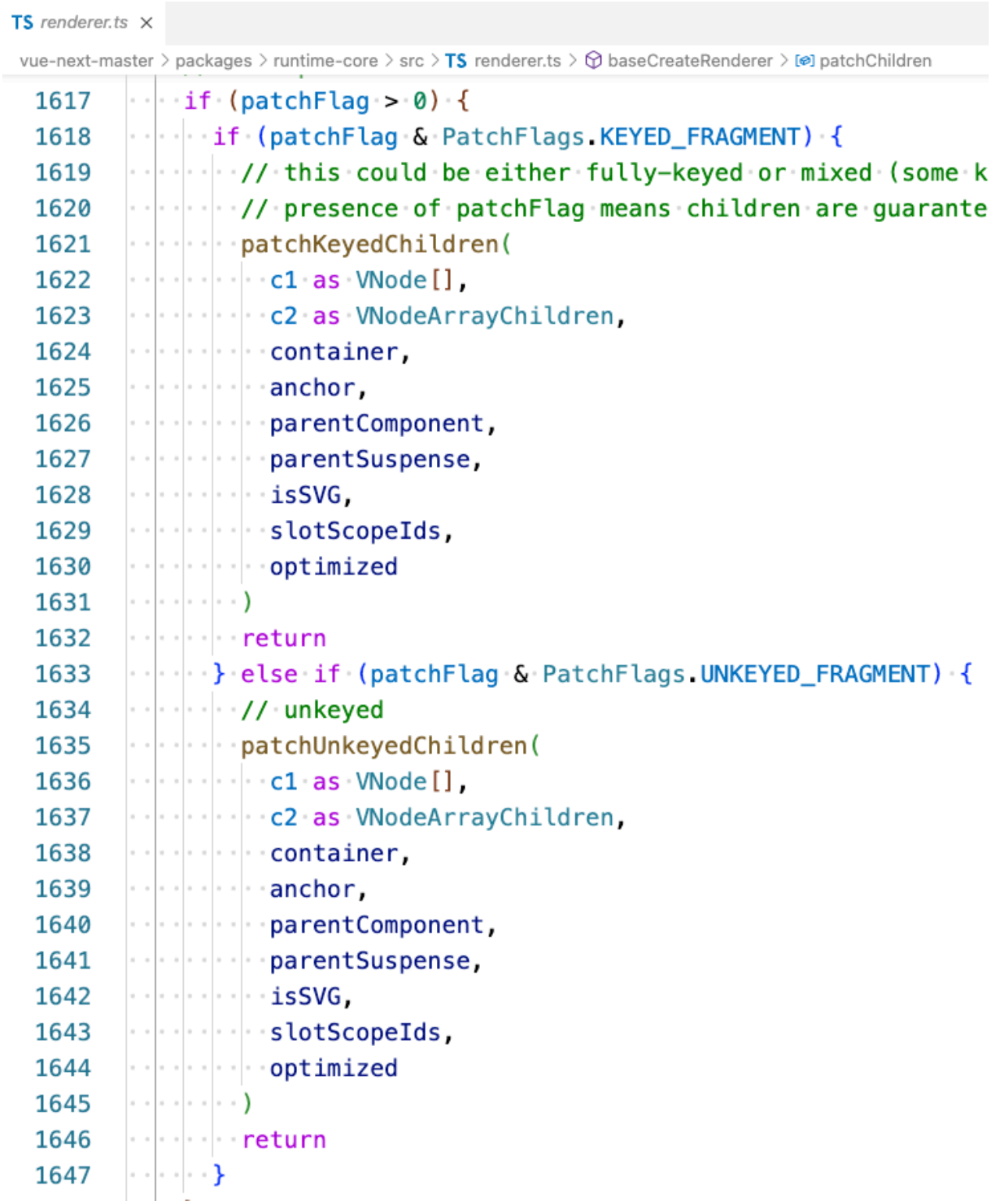

图2-28 v-for 有key和没有key的操作

上面介绍的`patchKeyedChildren`和`patchUnkeyedChildren`方法，其实就是 Vue.js 3 中的差异算法（也称为diff算法）的内容。下面我们来探讨一下这两个方法是怎么操作的。

2.7.3 没key时diff的操作

没有`key`时`patchUnkeyedChildren`方法对应源代码如下（省略了大部分代码）。

```
const patchUnkeyedChildren = (
    c1: VNode[], // 旧 VNodes [a, b, c, d]
```

```
    c2: VNodeArrayChildren, // 新 VNodes [a, b, f, c, d]
    container: RendererElement,
  ......
  ) => {
    c1 = c1 || EMPTY_ARR
    c2 = c2 || EMPTY_ARR
    // 1.获取旧节点的长度
    const oldLength = c1.length
    // 2.获取新节点的长度
    const newLength = c2.length
    // 3.获取最小那个的长度
    const commonLength = Math.min(oldLength, newLength)
    let i
    // 4.从0位置开始依次patch比较
    for (i = 0; i < commonLength; i++) {
      const nextChild = (c2[i] = optimized
        ? cloneIfMounted(c2[i] as VNode)
        : normalizeVNode(c2[i]))
      // 依次patch比较
      patch(
        c1[i], // 旧 VNode 节点
        nextChild, // 新 VNode 节点
        container,
        ......
      )
    }
    // 5.如果旧的节点数大于新的节点数
    if (oldLength > newLength) {
      // remove old
      // 5.1移除剩余的节点
      unmountChildren(
        c1,
        parentComponent,
        ......
      )
    } else {
      // mount new
      // 5.2创建新的节点
      mountChildren(
        c2,
        container,
        ......
      )
    }
  }
```

上面`patchUnkeyedChildren`方法对应的`diff`操作稍微有点难。为了更好地理解上面的`diff`操作过程，下面提供了一张`diff`操作过程图，如图2-29所示。

- 首先，旧`VNode`列表是：`[a,b,c,d]`，新`VNode`列表是：`[a,b,f,c,d]`。
- 接下来，旧的`a`和新的`a`进行`patch`对比，发现一样，不需要任何改动，接着旧的`b`和新的`b`进行`patch`对比，也一样，不需要任何改动。
- 然后，轮到旧的`c`和新的`f`进行`patch`对比，发现不一样，需要进行DOM操作把旧的`c`更新为`f`，接着旧的`d`和新的`c`进行`patch`对比，发现不一样，需要进行DOM操作把旧的`d`更新为`c`。
- 最后，新`VNode`列表还新增了一个`d`节点，直接新增`d`节点即可。

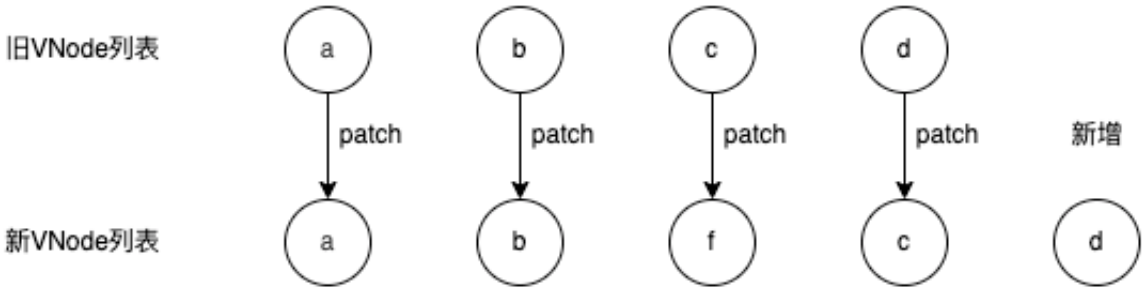

图2-29 没有key时diff的操作

上面的diff算法效率并不高，因为对于`c`和`d`来说，它们事实上并不需要有任何的改动。但是这里的`c`被`f`用了，后续所有的内容都要跟着进行改动。接下来，我们来看一下当存在key时的diff算法操作。

2.7.4 有key时diff的操作

有key时`patchKeyedChildren`方法对应的源代码如下（省略了大部分代码）。

```
const patchKeyedChildren = (
  c1: VNode[],  // 旧 VNodes [a, b, c, d]
  c2: VNodeArrayChildren, // 新 VNodes [a, b, f, c, d]
  ......
) => {
  ......
  // 1. sync from start
  // 从头部开始遍历，遇到相同的节点就继续，遇到不同的节点就跳出
  // (a b) c
  // (a b) d e
  while (i <= e1 && i <= e2) {
    const n1 = c1[i] // 旧节点
    const n2 = c2[i] // 新节点
    // 1.1 如果节点相同(isSameVNodeType：判断节点的类型和key都相同)，那么就继续遍历
    if (isSameVNodeType(n1, n2)) {
      patch(n1, n2, ......)
    } else {
      // 1.2 节点不同就直接跳出循环
      break
    }
    i++
  }
```

```
    // 2. sync from end
    // 从尾部开始遍历，遇到相同的节点就继续，遇到不同的节点就跳出
    // a (b c)
    // d e (b c)
    while (i <= e1 && i <= e2) {
      const n1 = c1[e1]
      const n2 = c2[e2]
      // 2.1 如果节点相同，就继续遍历
      if (isSameVNodeType(n1, n2)) {
        patch(n1, n2, ......)
      } else {
        // 2.2 不同就直接跳出循环
        break
      }
      e1--
      e2--
    }

    // 3. common sequence + mount
    // 如果旧节点遍历完了，依然有新的节点，那么新的节点就是添加(mount)
    // (a b)
    // (a b) c
    if (i > e1) {
      if (i <= e2) {
        while (i <= e2) {
          patch(null, c2[i],  ......)
          i++
        }
      }
    }
    // 4. common sequence + unmount
    // 如果新的节点遍历完了，还有旧的节点，那么旧的节点就是移除的
    // (a b) c
    // (a b)
    else if (i > e2) {
      while (i <= e1) {
        unmount(c1[i], ......)
        i++
      }
    }
    // 5. unknown sequence
    // 如果中间存在不知道如何排列的位置序列，那么就使用key建立索引图，最大限度的使用旧节点
    // a b [c d e] f g
    // a b [e d c h] f g
    else {
    ......
    }
  }
```

看完上面的代码后，下面具体的分析一下有key时diff对应的操作。

第一步：从头开始进行遍历、比较。

- 先是，旧的 `a` 和新的 `a` 是相同节点（它们类型和 key 都相同），会进行 patch 比较，发现一样则无需修改。
- 接着，旧的 `b` 和新的 `b` 是相同节点，会进行 patch 比较，发现一样则也无需任何修改。
- 接着，旧的 `c` 和新的 `f` 它们 key 不一样，不是相同节点，会 break 跳出循环，如图2-30所示。

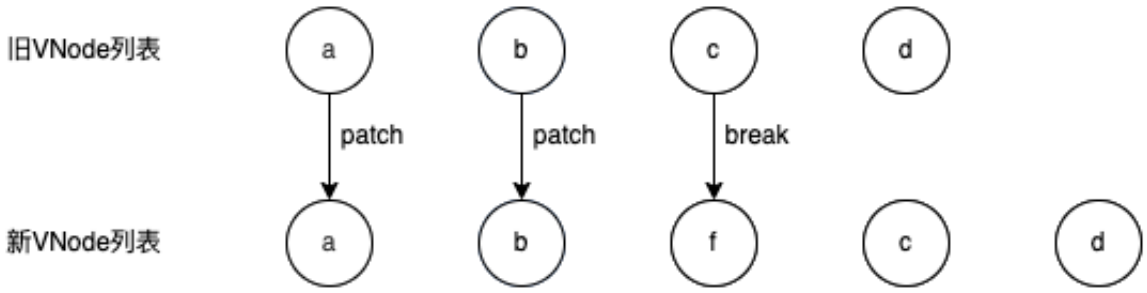

图2-30 从头开始进行遍历、比较

第二步：从尾部开始进行遍历、比较。

- 先是旧的 `d` 和新的 `d` 是相同节点，会进行 patch 比较，发现一样则无需任何修改。
- 接着旧的 `c` 和新的 `c` 是相同节点，会进行 patch 比较，发现一样则无需任何修改。
- 接着旧的 `b` 和新的 `f`，它们 key 不一样，不是相同节点，会 break 跳出循环，如图 2-31 所示。

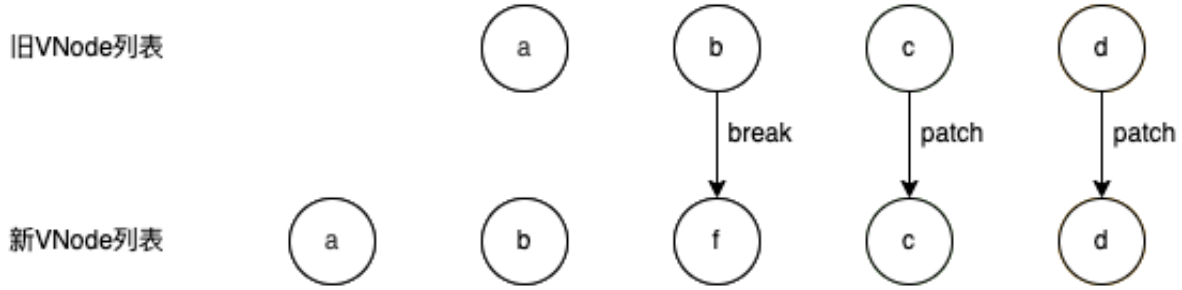

图2-31 从尾部开始进行遍历、比较

第三步：如旧节点遍历完毕，但是依然有新的节点，那就新增节点。

如前面两步已经把旧VNode列表遍历完毕，但是新VNode列表依然有个 `f` 节点，接着就是新增 `f` 节点，如图2-32所示。

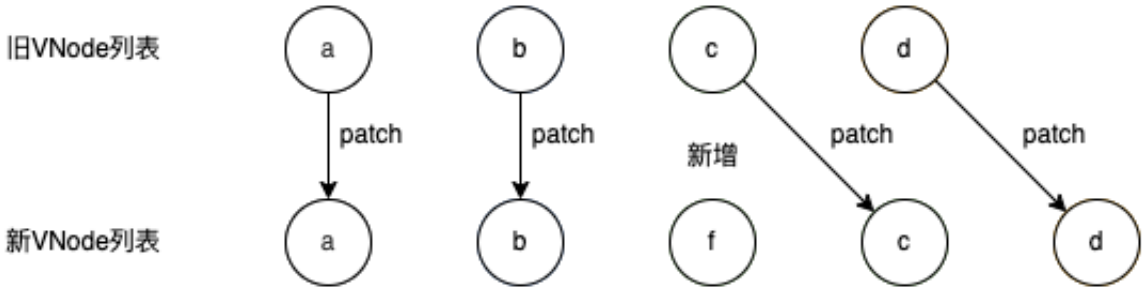

图2-32 新增节点

第四步：如新的节点遍历完毕，但是依然有旧的节点，那就移除旧节点。

如最前面的两步已经把新VNode列表遍历完毕，但是旧的VNode列表依然有个 `c` 节点，接着就是移除 `c` 节点，如图2-33所示。

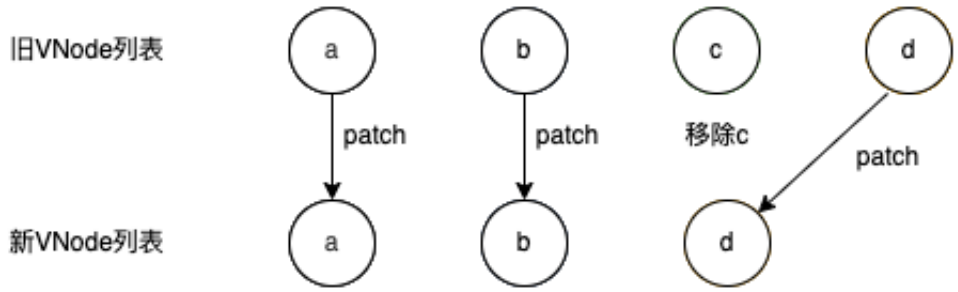

图2-33 移除旧节点

第五步：特殊情况，中间还有很多未知的或乱序的节点。

- 如前面四步已经执行完毕，但是新旧VNode列表中间依然有很多未知的或乱序的节点。
- 接着就是根据key建立map索引图，方便后面需要根据key复用元素和重新排列元素顺序。
- 遍历剩下的旧节点，进行新旧节点对比， 移除不使用的旧节点（如：i），那么相同的节点就直接复用。
- 最后如果新VNode列表还有剩下的节点就新增（如：f），如图2-34所示。

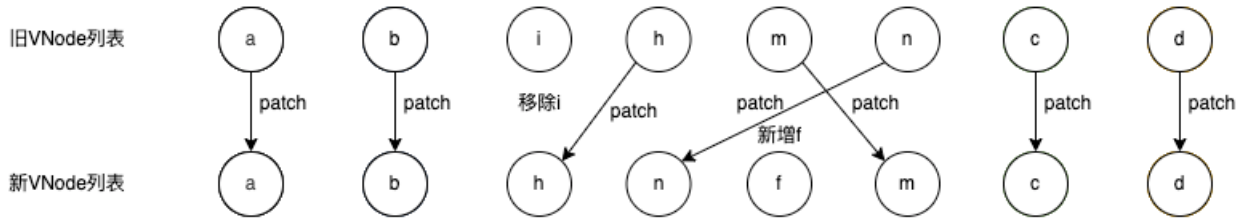

图2-34 处理未知的或乱序的节点

可以发现，Vue.js 3在进行 diff 算法时，会尽量利用 key 来进行优化操作。在没有 key 时，diff 算法效率是非常低效的，因为需要进行很多的 DOM 操作。因此，在进行插入或重置顺序时，保持相同的 key 可以让 diff 算法更加的高效。

2.8 本章小结

本章内容如下。

- Vue.js 3 模板语法：Mustache 插值语法。
- Vue.js 3 内置指令包括基本指令、v-bind、v-on、v-if、v-show和v-for。
- 在 for 循环中使用 key 可以帮助 Vue.js 3 虚拟 DOM 算法识别 VNodes，从而提高对 nodes 的复

用。