CHAPTER 5



第 5 章　React 的合成事件

合成事件(Synthetic Events)是 React 框架提供的一种针对原生 DOM 事件的跨浏览器兼容性封装。React 为了简化事件处理,并确保事件在不同浏览器中的行为一致性,创造了一套自己的事件系统。

合成事件使用了与原生 DOM 事件相似的 API,但有一些细微的差别,例如合成事件 API 的名称需使用小驼峰命名法(如使用 onClick 而非 onclick)。本章将介绍一些常用的 React 事件处理 API。

5.1　鼠标事件

5.1.1　onClick

onClick 事件是当鼠标单击一个元素时触发的事件。

【例 5-1】　修改 App.jsx,内容如下所示。

```
function App() {
  const handleClick = () => {
    console.log("Button clicked!");
  };

  return <button onClick={handleClick}>Click me</button>;
}

export default App;
```

在 App 组件内部,定义一个名为 handleClick 的箭头函数。这个函数被调用时会在浏览器的控制台输出“Button clicked!”。return 语句返回一个 JSX 表达式,在< button >按钮元素上,通过花括号语法将 handleClick 函数绑定到 React 合成事件的单击事件 onClick 上。

单击 Click me 按钮,触发 onClick 事件,handleClick 函数被调用,浏览器控制台将会打印出“Button clicked!”,如图 5-1 所示。

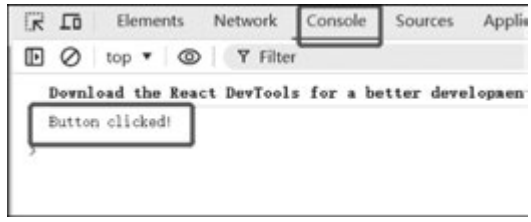


图 5-1　单击 Click me 按钮,查看打印信息

5.1.2 onDoubleClick

onDoubleClick 事件是指当鼠标双击一个元素时触发的事件。

【例 5-2】 修改 App.jsx，内容如下所示。

```
function App() {
  const handleDoubleClick = () => {
    console.log("按钮被双击了!");
  };

  return <button onDoubleClick={handleDoubleClick}>Click me</button>;
}

export default App;
```

双击 Click me 按钮，触发 onDoubleClick 事件，handleDoubleClick 函数被调用，控制台打印“按钮被双击了!”。

5.1.3 onMouseOver

onMouseOver 事件是指当鼠标指针移动到一个元素上时触发的事件。

【例 5-3】 修改 App.jsx，内容如下所示。

```
function App() {
  const handleMouseOver = () => {
    console.log("鼠标移入了按钮!");
  };

  return <button onMouseOver={handleMouseOver}>鼠标移入我</button>;
}

export default App;
```

当鼠标指针移入到按钮时，触发 onMouseOver 事件，handleMouseOver 函数被调用，控制台会打印“鼠标移入了按钮!”。

5.1.4 onMouseLeave

onMouseLeave 事件是指当鼠标指针从元素上移出时触发的事件。

【例 5-4】 修改 App.jsx，内容如下所示。

```
function App() {
  const handleMouseLeave = () => {
    console.log("鼠标离开了按钮!");
  };

  return <button onMouseLeave={handleMouseLeave}>鼠标离开我</button>;
}

export default App;
```

当鼠标指针离开按钮时，触发 onMouseLeave 事件，handleMouseLeave 函数被调用，控制台会打印“鼠标离开了按钮！”。

5.2 键盘事件

5.2.1 onKeyDown

onKeyDown 事件是指键盘上的按键被按下的事件。当用户按下一个键时，onKeyDown 事件会被触发。

【例 5-5】 修改 App.jsx，内容如下所示。

```
function App() {
  const handleKeyDown = (event) => {
    console.log("按键信息:", event.key);
  };

  return <input type="text" onKeyDown={handleKeyDown} />;
}

export default App;
```

在 App 组件内部，定义了一个名为 handleKeyDown 的箭头函数。参数 event 是一个事件对象，它包含了与按键事件相关的所有信息。这个函数被调用时会在浏览器的控制台输出被按下的键的值。

当在文本输入框中按下键盘上的任意键时，会触发 onKeyDown 事件，handleKeyDown 函数被调用，在控制台打印出按键信息。

5.2.2 onKeyUp

onKeyUp 事件是指键盘上的按键被释放的事件。当用户按下一个键并释放时，onKeyUp 事件会被触发。

【例 5-6】 修改 App.jsx，内容如下所示。

```
function App() {
  const handleKeyUp = (event) => {
    console.log("按键信息:", event.key);
  };

  return <input type="text" onKeyUp={handleKeyUp} />;
}

export default App;
```

当用户在文本输入框中释放键盘上的任意键时，触发 onKeyUp 事件，handleKeyUp 函数被调用，在控制台打印出按键信息。

5.3 焦点事件

5.3.1 onFocus

onFocus 事件用于监听元素获得焦点的情况。当元素被单击或通过程序获得焦点时，onFocus 事件将被触发。

【例 5-7】 修改 App.jsx，内容如下所示。

```
function App() {
  const handleFocus = () => {
    console.log("获取焦点");
  };

  return <input type="text" onFocus={handleFocus} />;
}

export default App;
```

鼠标单击输入框，输入框获得焦点，触发 onFocus 事件，handleFocus 函数被调用，控制台会打印“获取焦点”。

5.3.2 onBlur

onBlur 事件在元素失去焦点时触发，它是与 onFocus 事件相对应的事件。当用户将焦点从一个元素移开时，会触发该元素的 onBlur 事件。

【例 5-8】 修改 App.jsx，内容如下所示。

```
function App() {
  const handleBlur = () => {
    console.log("失去焦点");
  };

  return <input type="text" onBlur={handleBlur} />;
}

export default App;
```

当输入框失去焦点时，触发 onBlur 事件，handleBlur 函数被调用，控制台会打印“失去焦点”。

5.4 表单事件

5.4.1 onChange

onChange 事件在表单元素的值发生改变时触发。它适用于文本输入框、复选框、单选框、下拉列表等表单元素。

【例 5-9】 修改 App.jsx，内容如下所示。

```
function App() {
  const handleChange = (event) => {
    console.log(event.target.value);
  };

  return < input type = "text" onChange = {handleChange} />;
}

export default App;
```

在 App 组件内部，定义了一个名为 handleChange 的箭头函数。参数 event 是一个事件对象，它包含了与值改变事件相关的所有信息。通过访问 event.target.value 属性，可以获取输入值的内容。

当用户输入或修改输入框的内容时，会触发 onChange 事件，handleChange 函数被调用，并在控制台中打印出当前输入的值。

5.4.2 onSubmit

onSubmit 事件用于处理表单提交。当用户在表单中单击提交按钮或按下回车键时，该事件会触发。

【例 5-10】 修改 App.jsx，内容如下所示。

```
function App() {
  const handleSubmit = (event) => {
    // 阻止浏览器重新加载页面
    event.preventDefault();

    // 读取表单数据
    const form = event.target;
    const formData = new FormData(form);

    //读取 name = "myInput"的输入框数据
    const myInput = formData.get("myInput");
    console.log(myInput);
  };

  return (
    < form method = "post" onSubmit = {handleSubmit}>
      < label >
        文本框:
        < input name = "myInput" defaultValue = "Some initial value" />
      </label >
      < button type = "reset">重置表单</button >
      < button type = "submit">提交表单</button >
    </form >
  );
}

export default App;
```

在 App 组件内部，定义了一个名为 handleSubmit 的箭头函数。参数 event 是一个事件对象，包含

了与提交事件相关的所有信息。函数内部首先调用 event.preventDefault 方法阻止了浏览器的默认行为(即页面刷新)。通过 event.target 获取触发事件的目标元素,即当前的< form >元素。使用 FormData 对象从表单中提取数据。然后从 FormData 对象中读取名为 myInput 的输入框的值,即在 JSX 中 input 标签的 name 属性的值。最后将获取的 myInput 值打印到浏览器的控制台。

当用户在输入框中输入内容后,单击提交按钮或按下回车键时,会触发 onSubmit 事件,并将输入框的值打印到浏览器的控制台。这种模式常用于用户注册和登录等应用场景。

5.5 滚动事件

滚动事件是指在网页或元素内容区域滚动时触发的事件。

【例 5-11】 修改 App.css,内容如下所示。

```
.scroll_container {
  width: 400px;
  height: 300px;
  overflow: auto;
}

.scroll_content {
  height: 800px;
  background-color: #f0f0f0;
}
```

App.css 定义了两个类选择器.scroll-container 和.scroll-content,分别用于控制网页中元素的布局和样式。

修改 App.jsx,内容如下所示。

```
import "./App.css";

function App() {
  const handleScroll = (event) => {
    const container = event.target;
    const scrollHeight = container.scrollHeight;
    const scrollTop = container.scrollTop;
    const clientHeight = container.clientHeight;

    if (scrollHeight - scrollTop === clientHeight) {
      // 当滚动到底部时执行相应逻辑
      console.log("已滚动到底部");
    }
  };

  return (
    <div className="scroll_container" onScroll={handleScroll}>
      <div className="scroll_content"></div>
    </div>
  );
}

export default App;
```

在这个示例中，创建了一个滚动容器 scroll_container，并给容器绑定了滚动事件 onScroll。当滚动容器的内容发生滚动时，会触发 onScroll 事件，handleScroll 函数被调用。

在 handleScroll 函数中，通过事件对象 event 获取滚动容器的滚动高度 scrollHeight、滚动位置 scrollTop 以及可见区域的高度 clientHeight。通过这些信息，可以判断滚动是否到达了底部（scrollHeight-scrollTop===clientHeight），并执行相应的逻辑。在示例中，当滚动到底部时，控制台会打印出“已滚动到底部”的消息。

5.6 事件传参

【例 5-12】 给函数传参是个常见的业务场景。修改 App.jsx，内容如下所示。

```
function App() {
  const handleClick = (message) => {
    console.log(message); // 输出 'hello'
  };

  return <button onClick={() => handleClick("hello")}>按钮</button>;
}

export default App;
```

在 App 组件中，定义一个名为 handleClick 的事件处理函数，它接收一个参数 message。当这个函数被调用时，它会在控制台打印出传入的 message 参数值。

按钮上绑定了一个 onClick 事件处理器，该处理器是一个匿名箭头函数，当按钮被单击时，这个匿名函数会被执行。这个匿名函数内部调用了 handleClick 函数，并传入了字符串"hello"作为参数。因此，当单击按钮时，浏览器控制台会打印出 hello。

5.7 阻止事件冒泡

5.7.1 什么是事件冒泡

事件冒泡是 JavaScript 中的一种事件传播机制。当一个 HTML 元素上的事件被触发时（如单击按钮），该事件不仅会在当前元素上被处理，而且会逐级向上传递到它的所有祖先元素，直到文档的根元素。这种自下而上的事件传递过程称为事件冒泡。

事件冒泡分为三个阶段：捕获阶段、目标阶段和冒泡阶段。

（1）捕获阶段：事件从最外层元素（通常是 window 对象）开始向下传播到目标元素。

（2）目标阶段：事件到达并由实际的目标元素处理。

（3）冒泡阶段：事件从目标元素开始向上逐级传播回最外层元素。这个阶段像水里的气泡一样从下向上传播。

简单来说，当使用鼠标单击某个元素时，不仅会触发该元素自身的事件，还会依次触发其所有父元素及祖先元素上绑定的相同事件。下面通过一个简单示例进行说明。

【例 5-13】 修改 App.jsx，内容如下所示。

```
export default function App() {
  const handleBox1Click = () => {
    console.log("box1");
  };

  const handleBox2Click = () => {
    console.log("box2");
  };

  const handleBox3Click = () => {
    console.log("box3");
  };

  return (
    <div className="box1" style={{ width: 200, height: 200, border: "1px solid" }}
onClick={handleBox1Click}>
      box1
      <div className="box2" style={{ width: 100, height: 100, border: "1px solid" }}
onClick={handleBox2Click}>
        box2
        <div className="box3" style={{ width: 50, height: 50, border: "1px solid" }}
onClick={handleBox3Click}>
          box3
        </div>
      </div>
    </div>
  );
}
```

保存代码后，可以看到页面有三块区域。单击 box3 区域，浏览器控制台打印结果为“box3 box2 box1”，如图 5-2 所示。

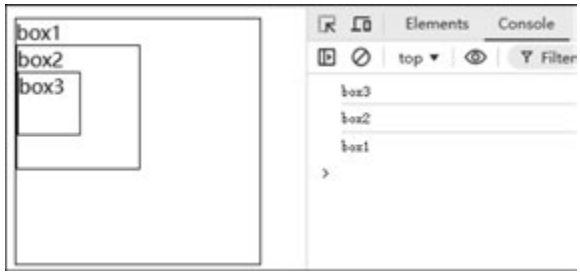


图 5-2 单击 box3 区域，查看打印信息

当单击 box3 区域时，尽管只进行了一次单击操作，但控制台却显示了三次单击事件的触发记录。这实际上是由事件冒泡机制引起的。具体过程如下：当单击 box3 区域时，事件首先从最外层元素开始向下传播到目标元素（捕获阶段）。当事件到达目标元素 box3 区域时，单击事件被触发（目标阶段），控制台打印“box3”。之后，事件从目标元素开始向上逐级传播回最外层元素（冒泡阶段），依次触发上一层的 box2 区域单击事件和上上层的 box1 区域单击事件，从而导致控制台打印了三次。

5.7.2 阻止事件冒泡

在大多数情况下，不希望一次单击触发多个事件。因此，需要手动阻止事件冒泡来避免这种情况。

【例 5-14】 在 React 合成事件中，使用 e.stopPropagation()阻止事件冒泡。修改 App.jsx，内容如下所示。

```
export default function App() {
  const handleBox1Click = () => {
    console.log("box1");
  };

  const handleBox2Click = () => {
    console.log("box2");
  };

  const handleBox3Click = (e) => {
    e.stopPropagation() //阻止合成事件冒泡
    console.log("box3");
  };

  return (
    <div className="box1" style={{ width: 200, height: 200, border: "1px solid" }}
onClick={handleBox1Click}>
      box1
      <div className="box2" style={{ width: 100, height: 100, border: "1px solid" }}
onClick={handleBox2Click}>
        box2
        <div className="box3" style={{ width: 50, height: 50, border: "1px solid" }}
onClick={(e) => handleBox3Click(e)}>
          box3
        </div>
      </div>
    </div>
  );
}
```

此时再单击 box3 区域，浏览器控制台只会打印“box3”，成功阻止了事件冒泡。

本章小结

（1）五种常见的合成事件类型：通过实例介绍了鼠标事件、键盘事件、焦点事件、表单事件和滚动事件的使用方法。

（2）事件处理函数传递参数的方法：展示了如何灵活地向事件处理函数传递参数，以实现对组件行为的更精细控制。

（3）解析了事件冒泡及其阻止方法：解释了什么是事件冒泡以及在 React 中如何有效地阻止事件冒泡，以避免不必要的事件传播。

通过本章的学习，读者应该已经熟悉了合成事件及其使用规则。接下来的章节将介绍 React 组件的相关概念和用法。