第 3 章

React事件处理

本章介绍 React 事件处理概述，鼠标事件处理、焦点事件处理、键盘事件处理和图像事件处理的应用开发。

3.1 React 事件处理概述

3.1.1 事件

JavaScript 与 HTML 的交互是通过事件（Event）实现的。事件是指用户或浏览器执行的某种动作，如用户单击鼠标等。事件代表文档（Document）或浏览器窗口（Window）等某个有意义的运行时刻。

事件概念最早是在 IE3 和 Navigator2 中出现的，当时的用意是把某些表单处理工作从服务器转移到浏览器。DOM2 开始尝试以符合逻辑的方式来标准化 DOM 事件 API。目前，主流的浏览器都实现了 DOM2 事件系统的核心部分。浏览器事件系统非常复杂，即使所有的主流浏览器都实现了 DOM2，规范也没有涵盖到所有的事件类型。DOM3 新增的事件 API 又让这些问题更加复杂了。

DOM2 将事件流分为事件捕获、到达目标（即实际的目标元素接收到事件）和事件冒泡（即事件向上层节点传递）3 个阶段。事件捕获为提前拦截事件提供了可能，最迟要在事件冒泡阶段响应事件。为响应事件而调用的事件处理函数称为事件处理程序（或事件监听器）。

React 元素的事件处理和 DOM 元素的事件处理很相似。但是，React 事件的命名方式采用小驼峰式。对于由多个英文单词组成的事件名称的第一个英文单词全部小写，而从第二个单词起的每个单词首字母大写。而且，使用 JSX 语法时需要传入一个函数（而不是字符串）作为事件处理函数。使用 React 事件时，不能通过返回 false 的方式阻止其默认行为，而必须显式地使用 preventDefault 来阻止。一般不需要使用 addEventListener 为已创建的 DOM 元素添加监听器，只需要在该元素初始渲染的时候添加监听器即可。定义类组件

时，通常要将事件处理函数声明为类中的方法。

必须谨慎对待 JSX 回调函数中的 this。在 JavaScript 中，class 的方法默认不会绑定 this。例如，如果没有绑定 this.handleClick 并把它传入了 onClick，即 onClick={this.handleClick}；在调用这个函数的时候，this 的值为 undefined。在通常情况下，如果没有在方法后面添加 ()，就应该为这个方法绑定（bind）this。还可以使用实验性的类 field 语法或回调函数避免使用绑定。如果回调函数作为 props 传入子组件时，这些组件可能会进行额外的重新渲染。建议在构造方法中绑定或使用类 field 语法来避免这类性能问题。

在循环中，通常需要为事件处理函数传入额外的参数。例如，若 id 是要删除那一行的 ID，以下两种方式都可以向事件处理函数传入参数：

```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>
<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
```

上述两种方式是等价的，分别通过箭头函数（使用了=>）和 Function.prototype.bind() 方法（方法绑定）来实现。如果通过箭头函数的方式，事件对象必须显式地进行传递，React 的事件对象 e 会被作为第二个参数传递。通过方法绑定的方式，事件对象以及更多的参数将会被隐式地进行传递。

3.1.2 合成事件

合成事件即自定义事件。事件合成除了可以处理兼容性问题，还可以用来自定义高级事件，如 onChange 事件（表单元素统一的值变动事件）。虚拟 DOM 抽象了跨平台的渲染方式，SyntheticEvent 实例将被传递给事件处理函数，它是浏览器的原生事件的跨浏览器包装器。除兼容所有浏览器外，它还拥有和浏览器原生事件相同的接口，包括 stopPropagation() 方法和 preventDefault()方法等。SyntheticEvent 是合并而来。这意味着 SyntheticEvent 对象可能会被重用，而且在事件回调函数被调用后，所有的属性都会无效。

大部分事件最终绑定到了 Document，而不是 DOM 节点本身。这就简化了 DOM 事件处理逻辑，减少了内存开销。不同类型的事件有不同的优先级，如高优先级的事件可以中断渲染，让用户代码可以及时响应用户交互。

3.1.3 支持的事件类型

常见的鼠标事件有 onClick、onContextMenu、onDoubleClick、onDrag、onDragEnd、onDragEnter、onDragExit、onDragLeave、onDragOver、onDragStart、onDrop、onMouseDown、onMouseEnter、onMouseLeave、onMouseMove、onMouseOut、onMouseOver 和 onMouseUp 等。其中 onMouseEnter 和 onMouseLeave 事件从离开的元素向进入的元素传播，不是正常的冒泡，也没有捕获阶段。

常见的焦点事件 onFocus、onBlur 在 React DOM 上的所有元素都有效，不只是表单元素。onFocus 事件在元素（或其内部某些元素）聚焦时被调用。onBlur 事件在元素（或其内部某些元素）失去焦点时被调用。可以使用 currentTarget 和 relatedTarget 来区分聚焦和失去焦点是否来自父元素外部。

常见的键盘事件有 onKeyDown、onKeyPress 和 onKeyUp。常见的图像事件有 onLoad 和 onError。

常见的指针事件（并非每个浏览器都支持指针事件）有 onPointerDown、onPointerMove、onPointerUp、onPointerCancel、onGotPointerCapture、onLostPointerCapture、onPointerEnter、onPointerLeave、onPointerOver 和 onPointerOut。其中 onPointerEnter 和 onPointerLeave 事件从离开的元素向进入的元素传播，不是正常的冒泡，也没有捕获阶段。

常见的表单事件有 onChange、onInput、onInvalid、onReset 和 onSubmit 等。常见的选择事件有 onSelect。常见的剪贴板事件有 onCopy、onCut 和 onPaste。常见的复合（Composition）事件有 onCompositionEnd、onCompositionStart 和 onCompositionUpdate。常见的通用事件有 onError 和 onLoad。常见的触摸事件有 onTouchCancel、onTouchEnd、onTouchMove 和 onTouchStart。

常见的 UI 事件有 onScroll。注意，从 React 17.0 开始，onScroll 事件在 React 中不再冒泡。这与浏览器的行为一致，并且避免了当一个嵌套且可滚动的元素在其父元素触发事件时造成的混乱。常见的滚轮事件有 onWheel。

常见的媒体事件有 onAbort、onCanPlay、onCanPlayThrough、onDurationChange、onEmptied、onEncrypted、onEnded、onError、onLoadedData、onLoadedMetadata、onLoadStart、onPause、onPlay、onPlaying、onProgress、onRateChange、onSeeked、onSeeking、onStalled、onSuspend、onTimeUpdate、onVolumeChange 和 onWaiting。

常见的动画事件有 onAnimationStart、onAnimationEnd 和 onAnimationIteration。常见的过渡事件有 onTransitionEnd。还有一些其他事件，如 onToggle。


视频讲解

3.2　鼠标事件处理

3.2.1　开发示例

在项目 firstreact 根目录下创建 eventexample 子目录，在 firstreact\eventexample 目录下创建文件 eventexample.html，代码与例 1-3 所示的代码相同。在 firstreact/eventexample 目录下创建文件 index.js，代码如例 3-1 所示。

【例 3-1】 在 firstreact\eventexample 目录下创建的文件 index.js 的代码。

```
const divReact = document.getElementById('root');
const infoMap= {
    clickAlertInfo:'发生了鼠标单击事件',
    linkPreventInfo:'即将访问 React 官网',
    reactWebsiteInfo:'https://reactjs.org/',
    gotoReactWebsiteInfo:'访问 React 官网',
    button2AlertInfo:'类事件处理正常',
    button2Value:'类的事件',
    callbackAlertInfo:'使用了箭头函数',
    callbackValue:'回调操作',
    eventWithPropsAlertInfo:'传递的参数为:name=',
```

```
    eventWithPropsValue:'带参数的事件处理',
    onOffStateAlertInfo:'现有的状态为',
    onOffButtonValue1:'开关按钮（初始为开）',
    onOffButtonValue2:'开关按钮（初始为关）',
    titleInfo:'事件处理的简单示例',
    button1Info:'鼠标单击事件示例：',
    button1Value:'单击鼠标事件',
    linkExInfo:'超链接访问示例：',
    button2Info:'类的事件处理示例：',
    callbackInfo:'回调操作示例：',
    eventWithPropsInfo:'带参数的事件处理示例：',
    onOffStateInfo:'bool 状态的变化示例：',
    preventDefaultInfo:'使用 preventDefault 阻止默认行为',
    actionLinkInfo:'单击 Link'
}
{/*鼠标单击事件处理*/}
function onBtnClick() {
    alert(infoMap.clickAlertInfo);
}
{/*阻止事件的默认行为*/}
function PreventLink() {
    function handleClick(e) {
        alert(infoMap.linkPreventInfo);
    }
    return (
        <a href={infoMap.reactWebsiteInfo} onClick={handleClick}>
            {infoMap.gotoReactWebsiteInfo}
        </a>
    );
}
class BtnClickComp extends React.Component {
    constructor(props) {
        super(props);
    }
    handleClick2() {
        alert(infoMap.button2AlertInfo);
    }
    render() {
        return (
            <button onClick={this.handleClick2}>
                {infoMap.button2Value}
            </button>
        );
    }
}
{/*使用了箭头函数，回调*/}
class CallBackBtnClickComp extends React.Component {
    constructor(props) {
        super(props);
```

```
    }
    handleClick = () => {
        alert(infoMap.callbackAlertInfo);
    };
    render() {
        // 此语法确保 handleClick()方法内的 this 已被绑定
        return (
            <button onClick={(e) => this.handleClick(e)}>
                {infoMap.callbackValue}
            </button>
        );
    }
}
{/*在事件处理方法中传递参数*/}
class ParamsEventComp extends React.Component {
    constructor(props) {
        super(props);
        this.name =  props.name ;
    }
    passParamsClick(name, e) {//事件对象 e 要放在最后
        e.preventDefault();
        alert(infoMap.eventWithPropsAlertInfo+" "+name);
    }
    render() {
        return (
            <div>
                {/* 通过箭头函数方法传递参数 */}
                <button onClick={(e) => this.passParamsClick(this.name, e)}>
                    {infoMap.eventWithPropsValue}
                </button>
            </div>
        );
    }
}
class ToggleBtnComp extends React.Component {
    constructor(props) {
        super(props);
        this.isToggleOn=props.isToggleOn;
        //为了在回调中使用 this, 这里的绑定是必不可少的
        this.handleClick = this.toggleBtnClick.bind(this);
    }
    toggleBtnClick(isToggleOn, e) {
        e.preventDefault();
        alert(infoMap.onOffStateAlertInfo+" " + this.isToggleOn  );
        this.isToggleOn = ! this.isToggleOn;
    }
    render() {
        return (
            <button onClick={(e)=>this.toggleBtnClick(this.isToggleOn, e)}>
```

```
                {this.isToggleOn ? infoMap.onOffButtonValue1 : infoMap.
onOffButtonValue2}
            </button>
        );
    }
}
function ActionLink() {
    //e 是合成事件，React 根据 W3C 规范来定义合成事件，开发时无须考虑跨浏览器兼容问题
    function handleClick(e) {
        e.preventDefault();
        alert(infoMap.actionLinkInfo);
    }
    return (
        <a href="#" onClick={handleClick}>
            {infoMap.actionLinkInfo}
        </a>
    );
}
const exampleEvent = (
    <span>
        <h2 align="center">{infoMap.titleInfo}</h2>
        <span>{infoMap.button1Info}</span>
<button onClick={onBtnClick}>{infoMap.button1Value}</button>
         <hr/>
         <span>{infoMap.linkExInfo}</span><PreventLink/>
        <hr/>
         <span>{infoMap.button2Info}</span><BtnClickComp/>
         <hr/>
        <span>{infoMap.callbackInfo}</span><CallBackBtnClickComp/>
        <hr/>
        <span>{infoMap.eventWithPropsInfo}</span><ParamsEventComp name=
{'zd'}/>
        <hr/>
        <span>{infoMap.onOffStateInfo}</span><ToggleBtnComp isToggleOn=
{true}/>
        <hr/>
        <span>{infoMap.preventDefaultInfo}</span><ActionLink/>
    </span>
);
ReactDOM.createRoot(divReact).render(exampleEvent);
```

3.2.2 运行效果

运行文件 eventexample.html，效果如图 3-1 所示。

单击图 3-1 所示中的“单击鼠标事件”按钮，弹出如图 3-2 所示的对话框。单击图 3-1 所示中的超链接“访问 React 官网”，弹出如图 3-3 所示的对话框。单击该对话框中的“确定”按钮，结果如图 3-4 所示。单击图 3-1 所示中的“类的事件”按钮，弹出如图 3-5 所示

的对话框。单击图 3-1 所示中的“回调操作”按钮，弹出如图 3-6 所示的对话框。单击图 3-1 所示中的“带参数的事件处理”按钮，弹出如图 3-7 所示的对话框。单击图 3-1 所示中的“开关按钮（初始为开）”按钮，弹出如图 3-8 所示的对话框。单击图 3-1 所示中的超链接“单击 Link”，弹出如图 3-9 所示的对话框。

图 3-1　运行文件 eventexample.html 的效果

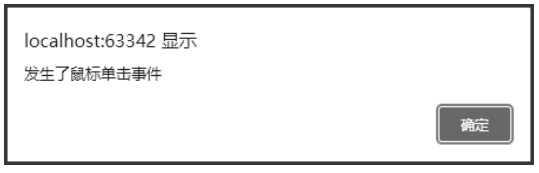

图 3-2　单击图 3-1 所示中的“单击鼠标事件”按钮的效果

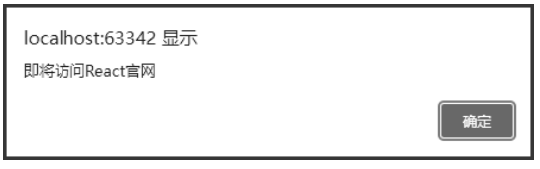

图 3-3　单击图 3-1 所示中的超链接“访问 React 官网”的效果

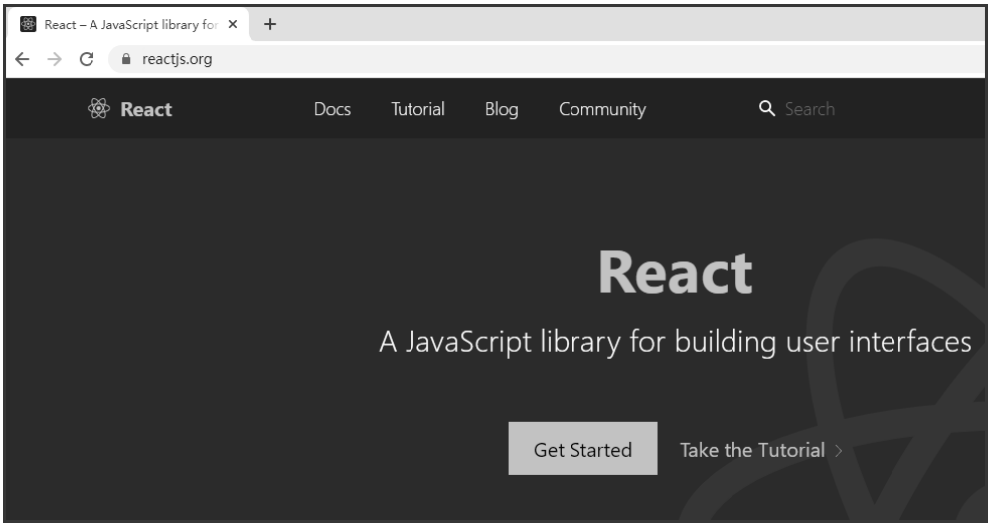

图 3-4　单击图 3-3 所示中的“确定”按钮的效果

图 3-5　单击图 3-1 所示中的“类的事件”按钮的效果

图 3-6　单击图 3-1 所示中的“回调操作”按钮的效果

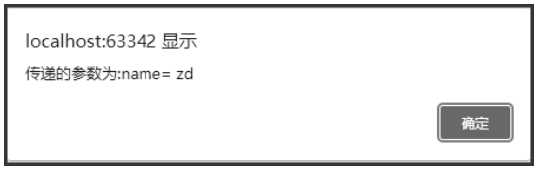

图 3-7　单击图 3-1 所示中的“带参数的事件处理”按钮的效果

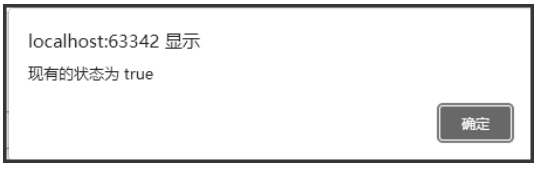

图 3-8　单击图 3-1 所示中的“开关按钮（初始为开）”按钮的效果

图 3-9　单击图 3-1 所示中的超链接“单击 Link”的效果


视频讲解

3.3　焦点事件处理

3.3.1　开发示例

在 firstreact 根目录下创建 inputexample 子目录，在 firstreact\inputexample 目录下创建文件 inputexample.html，代码与例 1-3 所示的代码相同。在 firstreact\inputexample 目录下创建文件 index.js，代码如例 3-2 所示。

【例 3-2】　在 firstreact\inputexample 目录下创建的文件 index.js 的代码。

```
const divReact = document.getElementById('root');
const infoMap= {
    input1focusAlertInfo:'第一个文本框获得焦点',
    input1changeAlertInfo:'第一个文本框内容变为',
    input1blurAlertInfo:'第一个文本框失去焦点',
    input2focusAlertInfo:'第二个文本框获得焦点',
    input2NotFocusAlertInfo:'第二个文本框失去焦点',
    titleInfo:'文本框事件简单示例',
}
class InputListenerComp extends React.Component {
    constructor(props) {
        super(props);
        this.inputVal=props.inputVal;
        this.inputTextFocus = this.inputTextFocus.bind(this);
        this.inputTextChange = this.inputTextChange.bind(this);
```

```
        this.inputTextBlur = this.inputTextBlur.bind(this);
    }
    inputTextFocus(inputVal) {
        alert(infoMap.input1focusAlertInfo);
    }
    inputTextChange(inputVal) {
        alert(infoMap.input1changeAlertInfo+inputVal);
    }
    inputTextBlur(inputVal) {
        alert(infoMap.input1blurAlertInfo);
    }
    render() {
        const inputVal = this.inputVal;
        return (
            <input type="text" value={inputVal}
                  onFocus={this.inputTextFocus(inputVal)}
                  onChange={this.inputTextChange(inputVal)}
                  onBlur={this.inputTextBlur(inputVal)} />
        );
    }
}
class InputFocusComp extends React.Component {
    constructor(props) {
        super(props);
        this.inputTextFocus = this.inputTextFocus.bind(this);
    }
    inputTextFocus(inputTextFocus) {
        if (inputTextFocus) {
            alert(infoMap.input2focusAlertInfo);
            inputTextFocus =! inputTextFocus;
        }
        alert(infoMap.input2NotFocusAlertInfo);
    }
    render() {
        return (
            <input type="text" onFocus={this.inputTextFocus} />
        );
    }
}
const exampleInputEvent = (
    <span>
        <h2 align="center">{infoMap.titleInfo}</h2>
        {/*注意"true"和 true 的差别*/}
        <InputListenerComp inputVal={"true"}/>
        <InputFocusComp  inputTextFocus={true}/>
    </span>
);
ReactDOM.createRoot(divReact).render(exampleInputEvent);
```

3.3.2 运行效果

运行文件 inputexample.html，效果如图 3-10 所示。单击图 3-10 所示中的“确定”按钮，弹出如图 3-11 所示的对话框。单击图 3-11 所示中的“确定”按钮，弹出如图 3-12 所示的对话框。单击图 3-12 所示中的“确定”按钮，效果如图 3-13 所示。单击图 3-13 所示中的第二个文本框（横向从左至右），弹出如图 3-14 所示的对话框。单击图 3-14 所示中的“确定”按钮，弹出如图 3-15 所示的对话框。单击图 3-15 所示中的“确定”按钮，弹出如图 3-16 所示的对话框。之后在图 3-15 和图 3-16 之间来回转换。

图 3-10　运行文件 inputexample.html 的效果

图 3-11　单击图 3-10 所示中的“确定”按钮的效果

图 3-12　单击图 3-11 所示中的“确定”按钮的效果

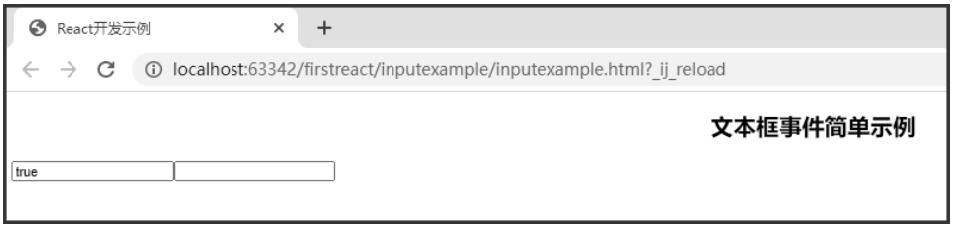

图 3-13　单击图 3-12 所示中的“确定”按钮的效果

图 3-14　单击图 3-13 所示中的第二个文本框（横向从左至右）的效果

图 3-15　单击图 3-14 所示中的“确定”按钮的效果

图 3-16　单击图 3-15 所示中的“确定”按钮的效果

3.4　键盘事件处理

3.4.1　开发示例

在 firstreact 根目录下创建 keyboardeventexample 子目录，在 firstreact\ keyboardeventexample 目录下创建文件 keyboardeventexample.html，代码与例 1-3 所示的代码相同。在 firstreact\ keyboardeventexample 目录下创建文件 index.js，代码如例 3-3 所示。

【例 3-3】 在 firstreact\keyboardeventexample 目录下创建的文件 index.js 的代码。

```
const divReact = document.getElementById('root');
const infoMap={
   msgInit:'通过键盘在文本框中输入字符后在浏览器 Console 中显示对应的 ASCII 码值',
   alertInfo:'按回车键，msg 值为：'
}
class KeyBind extends React.Component {
   constructor(props) {
      super(props)
      this.state = {
```

```
            msg: infoMap.msgInit
        }
    }
    keyUp = (e) => {
        console.log(e.keyCode)
        if (e.keyCode === 13) {
            alert(infoMap.alertInfo+e.target.value)
        }
    }
    render() {
        return (
            <div>
                <h2>{infoMap.msgInit}</h2>
                <input onKeyUp={this.keyUp}/>
            </div>
        )
    }
}
const keyboardEx = (
    <span>
        <KeyBind/>
    </span>
)
ReactDOM.createRoot(divReact).render(keyboardEx);
```

3.4.2 运行效果

运行文件 keyboardeventexample.html，效果如图 3-17 所示。通过在图 3-17 所示的文本框中输入字符（如“1”“a”），在浏览器的 Console 中输出字符对应的 ASCII 码值（如 49、65），效果如图 3-18 所示。在图 3-18 所示中，通过按回车键，弹出的对话框（包含文本框中存有的内容，如“1a”）如图 3-19 所示。

图 3-17　运行文件 keyboardeventexample.html 的效果

图 3-18　通过键盘在文本框中输入字符后在浏览器 Console 中显示对应的 ASCII 码值

图 3-19　在图 3-18 所示中通过按回车键后弹出的对话框

3.5　图像事件处理

3.5.1　开发示例

在 firstreact 根目录下创建 imageeventexample 子目录，在 firstreact\imageeventexample 目录下创建文件 imageeventexample.html，代码与例 1-3 所示的代码相同。在 firstreact\imageeventexample 目录下创建文件 index.js，代码如例 3-4 所示。还需要在 firstreact\imageeventexample 目录下创建 img 子目录，并且在 firstreact\imageeventexample\img 目录下准备图片文件 1.jpg、2.jpg、3.jpg 和 4.jpg。

【例 3-4】 在 firstreact\imageeventexample 目录下创建的文件 index.js 的代码。

```
const divReact = document.getElementById('root');
const infoMap = {
    imgPath: './img',
    imgEx:'.jpg',
    file1Name:'1',
    file2Name:'2',
    file3Name:'3',
    file4Name:'4',
    alertInfo: '装载图像'
}
//图像 onLoad 事件处理函数（方法）
function loadPic(){
    alert(infoMap.alertInfo)
}
class Img extends React.Component{
    constructor(props) {
        super(props);
        this.state={
            listImg:[
                './img/1.jpg',
                './img/2.jpg',
                './img/3.jpg',
                './img/4.jpg',
            ],
            index:0
        }
    }
//计时器执行
```

```
        indexChange(){
            if(this.state.index == this.state.listImg.length-1){
                this.setState({
                    index:0
                })
            }else{
                this.setState({
                    index:this.state.index+1
                })
                console.log(this.state.index); //在浏览器 Console 中输出图片序号
            }
        }
        componentDidMount(){
            setInterval(()=>{
                this.indexChange();
            },2000)
        }
        render(){
            let {listImg,index} = this.state;
            let imgList=listImg.map((item,imgIndex)=>{
                return
    <img src={item} key={imgIndex} style={{'display':index == imgIndex ?'block' :
'none'}} className='img' onLoad={loadPic}/>
            })
            let liList=listImg.map((item2,imgIndex2)=>{
                return
                 <li  key={imgIndex2} style={{'listStyleType':index == imgIndex2 ?
'initial' :'circle'}}></li>
            })
            return (
                <div>
                    {imgList}
                    <div>
                        <ul >
                            {liList }
                        </ul>
                    </div>
                </div>
            )
        }
    }
    const loadImageEx = (
        <span>
            <Img/>
        </span>
    )
    ReactDOM.createRoot(divReact).render(loadImageEx);
```

3.5.2　运行效果

运行文件 imageeventexample.html，效果如图 3-20 所示。单击图 3-20 所示中的“确定”按钮，即可开始装载图像（图片），如图 3-21 所示。在装载后，显示轮播信息（如图 3-21 所示中的 4 个圆点对应的图片）。装载之后，就在 4 幅图片之间循环播放，如图 3-22 所示。

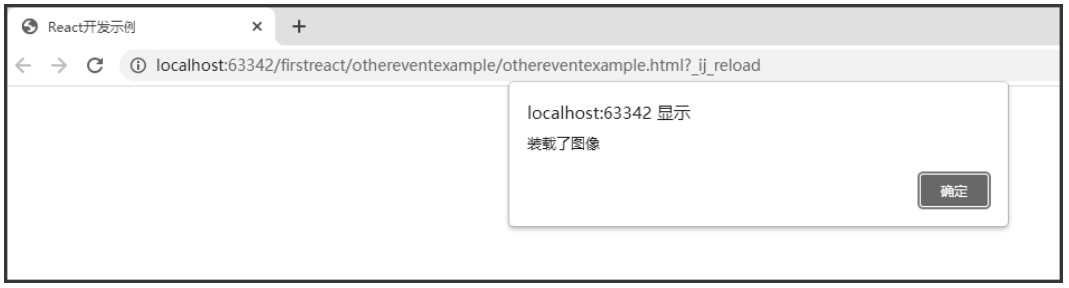

图 3-20　运行文件 imageeventexample.html 的效果

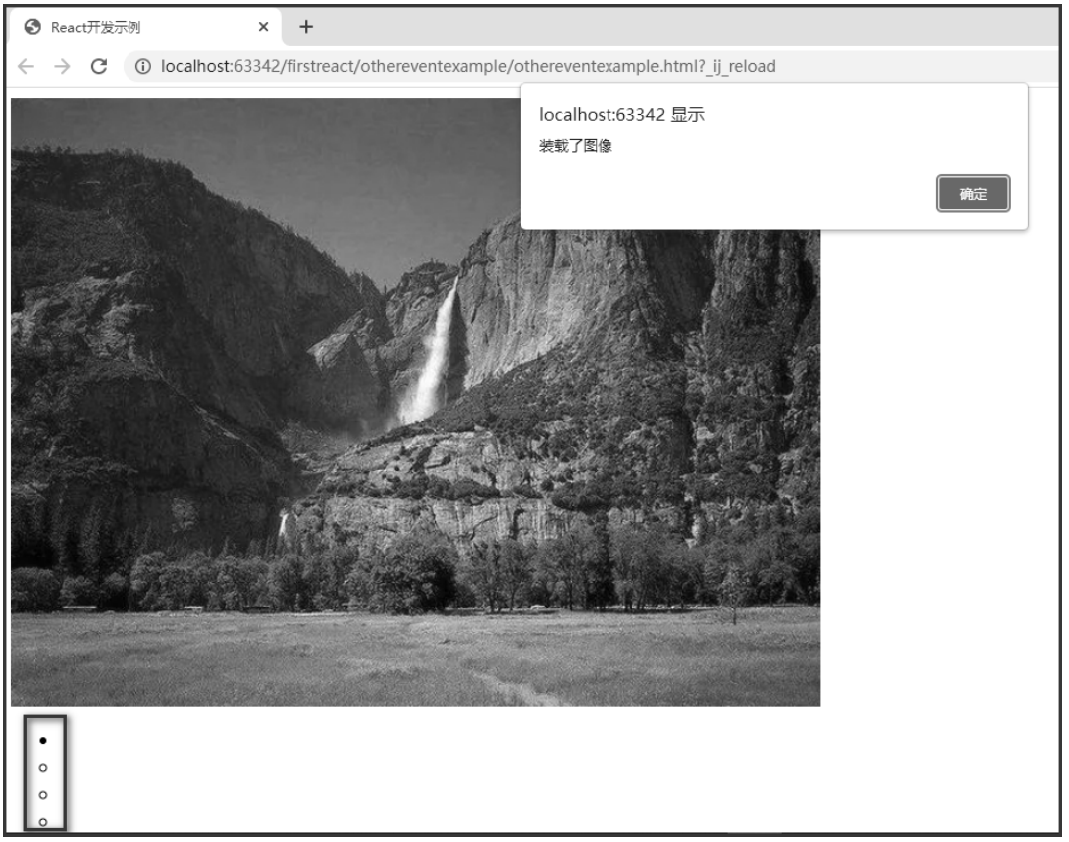

图 3-21　装载图像后显示的轮播信息

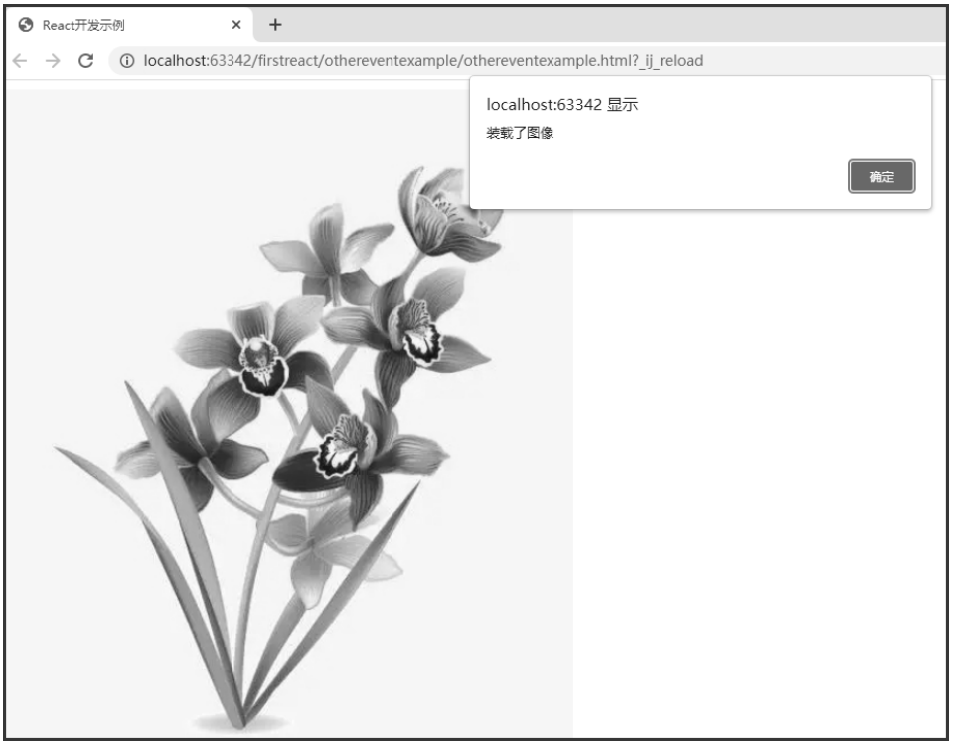

图 3-22　装载后在 4 幅图片之间播放的过程

习题 3

一、简答题

1. 简述对事件的理解。
2. 简述对合成事件的理解。

二、实验题

1. 实现鼠标事件处理的应用开发。
2. 实现焦点事件处理的应用开发。
3. 实现键盘事件处理的应用开发。
4. 实现图像事件处理的应用开发。