

第 5 章

React状态管理

本章先简要介绍 `state`（状态）、`setState()`方法、`forceUpdate()`方法、状态提升等内容，再介绍状态的基础应用开发以及状态提升的应用开发等内容。

5.1 React 状态管理概述

5.1.1 state

组件中的 `state` 包含了随时可能发生变化的数据。`state` 与 `props` 类似，但 `state` 由用户自定义，是私有的，并且完全受控于当前组件。它是一个普通 JavaScript 对象。如果某些数据（值）未用于渲染或数据流，就不必将其设置为 `state`。此类数据可以在组件实例上定义。

如例 5-1 所示的类组件 `Clock`，在每次组件更新时，`render()`方法都会被调用，但只要相同的 DOM 节点中渲染 `<Clock />`，就只有一个 `Clock` 组件实例被创建。

【例 5-1】 类组件 `Clock` 的代码。

```
class Clock extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.props.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

可以通过三个步骤向类组件中添加局部 `state`：把 `render()`方法中的 `this.props` 替换成

`this.state`；添加一个构造方法，然后在该方法中为 `this.state` 赋初值；转移属性（从组件被调用时的 `props` 转换到组件内）。将例 5-1 中的 `date` 从 `props` 移动到 `state` 中，经过转换后的带 `state` 的类组件 `Clock` 代码如例 5-2 所示。

【例 5-2】 转换后的带 `state` 的类组件 `Clock` 的代码。

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }
  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

5.1.2 `setState()`方法

`setState(updater, [callback])`方法简称为 `setState()`方法。将对组件 `state` 的更改排入队列，并通知 `React` 使用更新后的 `state` 重新渲染此组件及其子组件。这是用于更新 `UI` 界面以响应事件和处理服务器数据的主要方式。

可以在组件的构造方法中给 `this.state` 赋值。除了在组件构造方法中直接给 `this.state` 赋值之外，应使用 `setState()`方法来修改 `state`，而不要直接修改 `state`（如赋值语句）。同时，可以把多个 `state` 的 `setState()`方法调用合并成一个方法。

在调用 `setState()`方法时，`React` 会先把提供的对象合并到当前的 `state`，再分别调用 `setState()` 方法来单独地更新它们。

不管是父组件或是子组件，都无法知道某个组件是否有 `state`，并且它们也并不关心这个组件是函数组件还是类组件。组件可以选择把它的 `state` 作为 `props` 向下传递到子组件中，但是组件本身无法知道它是来自组件（或父组件）的 `state` 或 `props`，还是手动输入的。这通常会被叫作自上而下的单向数据流。任何的 `state` 总是属于特定的组件，而且从该 `state` 派生的任何数据或 `UI` 界面只能影响组件树中低于它们的组件（如它们的子组件）。如果把一个以组件树想象成一个 `props` 的数据瀑布的话，那么每一个组件的 `state` 就像是在任意一点上给瀑布增加额外的水源，它只能向下流动。

在调用 `setState()`方法后，若立即读取 `this.state`，可能会有隐患。为了消除隐患，可以使用 `componentDidUpdate()`方法或者 `setState()`方法，这两种方式都可以保证在应用更新后触发。

除非 `shouldComponentUpdate()`方法返回 `false`，否则 `setState()`方法将始终执行重新渲染操作。如果使用可变对象，且无法在 `shouldComponentUpdate()` 方法中实现条件渲染，那么仅在新旧状态不同时调用 `setState()`方法，就可以避免不必要的重新渲染。

5.1.3 forceUpdate()方法

组件的 `forceUpdate(callback)` 方法简称为 `forceUpdate()` 方法。在默认情况下, 当组件的 `state` 或 `props` 发生变化时, 组件将重新渲染。如果 `render()` 方法依赖于其他数据, 就可以调用 `forceUpdate()` 方法强制让组件重新渲染。

调用 `forceUpdate()` 方法将会导致组件调用 `render()` 方法, 此操作会跳过该组件的 `shouldComponentUpdate()` 方法。但其子组件会触发正常的生命周期方法, 包括 `shouldComponentUpdate()` 方法。如果标签发生变化, `React` 仍将只更新 `DOM`。

通常, 应避免使用 `forceUpdate()` 方法, 尽量在 `render()` 方法中使用 `this.props` 和 `this.state`。

5.1.4 状态提升

当多个组件需要反映相同的变化数据时, 可以将共享状态提升到最近的共同父组件中去。这和面向对象设计(如 `Java` 开发)时将子类共同内容抽取到父类中的思想类似。在 `React` 中, 将多个组件中需要共享的 `state` 向上移动到它们最近共同父组件中, 便可实现多个组件共享 `state`。这就是所谓的“状态提升”。

`React` 中的任何可变数据应当只有一个相对应的唯一数据源。通常, `state` 都是先添加到需要渲染数据的组件中去。如果其他组件也需要这个 `state`, 那么就可以将它提升至这些组件最近共同父组件中。

虽然提升 `state` 方式比双向绑定方式需要编写更多的代码, 但带来的好处是, 排查和隔离 `bug` 所需的工作量将会变少。还可以使用自定义逻辑来拒绝或转换用户的输入。

在 `UI` 界面中发现错误时, 可以检查问题组件的 `props`, 并且按照组件树结构逐级向上搜寻, 直到定位到负责更新 `state` 的组件。



视频讲解

5.2 状态的基础应用

5.2.1 开发示例

在 `firstreact` 根目录下创建 `stateexample` 子目录, 在 `firstreact\stateexample` 目录下创建文件 `stateexample.html`, 代码与例 1-3 所示的代码相同。在 `firstreact\stateexample` 目录下创建文件 `index.js`, 代码如例 5-3 所示。

【例 5-3】 在 `firstreact\stateexample` 目录下创建的文件 `index.js` 的代码。

```
const divReact = document.getElementById('root');
const infoMap = {
  title1info: '状态和生命周期的应用',
  nowInfo: '现在是: ',
  endInfo: '。',
  title2info: '数据流应用示例 1',
  title3info: 'setState()方法示例',
  countInfo: '现在的计数是: ',
```

```

    helloInfo: 'Hello React!',
    btn1info: '箭头函数',
    btn2info: 'bind()方法',
    title4info: '数据流应用示例 2',
    title5info: '数据流应用示例 3',
  }
}
class ClockReactComp extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }
  //生命周期的方法
  componentDidMount() {
    this.timerId = setInterval(
      () => this.tick(),
      1000
    );
  }
  componentWillUnmount() {
    clearInterval(this.timerId);
  }
  tick() {
    this.setState({
      date: new Date()
    });
  }
  render() {
    return (
      <span>
        <h4>{infoMap.title1info}</h4>
        <p>{infoMap.nowInfo}{this.state.date.toLocaleTimeString()}
{infoMap.endInfo}</p>
      </span>
    );
  }
}
function FormattedDate(props) {
  return <h4>{infoMap.nowInfo}{props.date.toLocaleTimeString(){infoMap.
endInfo}</h4>;
}
class ClockReactComp2 extends React.Component {
  static defaultProps = {
    propsDate: new Date()
  };
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }
  componentDidMount() {

```

```

        this.timerId = setInterval(
            () => this.tick(),
            1000
        );
    }
    componentWillUnmount() {
        clearInterval(this.timerId);
    }
    tick() {
        this.setState({
            date: new Date()
        });
    }
    render() {
        return (
            <span>
                <h3>{infoMap.title2info}</h3>
                <FormattedDate date={this.state.date} />
                <FormattedDate date={new Date()} />
                <FormattedDate date={this.props.propsDate} />
            </span>
        );
    }
}
class ClockReactComp3 extends React.Component {
    static defaultProps = {
        propsDate: new Date()
    };
    constructor(props) {
        super(props);
        this.state = {date: new Date()};
    }
    componentDidMount() {
        this.timerId = setInterval(
            () => this.tick(),
            1000
        );
    }
    componentWillUnmount() {
        clearInterval(this.timerId);
    }
    tick() {
        this.setState({
            date: new Date()
        });
    }
    render() {
        return (
            <span>

```

```

        <FormattedDate date={this.state.date} />
      </span>
    );
  }
}
//数据流
function ClockDataFlow() {
  return (
    <div>
      <ClockReactComp3/>
      <ClockReactComp3/>
    </div>
  );
}
class ClockReactComp4 extends React.Component {
  static defaultProps = {
    propsDate: new Date()
  };
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }
  componentDidMount() {
    this.timerId = setInterval(
      () => this.tick(),
      Math.ceil(Math.random()*9)*1000
    );
  }
  componentWillUnmount() {
    clearInterval(this.timerId);
  }
  tick() {
    this.setState({
      date: new Date()
    });
  }
  render() {
    return (
      <span>
        <FormattedDate date={this.state.date} />
      </span>
    );
  }
}
function ClockDataFlow2() {
  return (
    <div>
      <ClockReactComp4 />
      <ClockReactComp4 />
    </div>
  );
}

```

```

        </div>
    );
}
class CountReactComp extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            count: 0
        };
    }
    componentDidMount() {
        this.timerId = setInterval(
            () => this.count(),
            1000
        );
    }
    count() {
        this.setState((prevState, props) => ({
            count: prevState.count + props.increment
        }));
    }
    componentWillUnmount() {
        clearInterval(this.timerId);
    }
    render() {
        return (
            <span>
                <div>{infoMap.title3info}</div>
                <div>{infoMap.countInfo}{this.state.count}{infoMap.
endInfo}</div>
            </span>
        );
    }
}
//注意代码的顺序
//defaultProps 应用
CountReactComp.defaultProps = {
    increment: 1
};
class ParamsEventComp extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            name: infoMap.helloInfo
        };
    }
    passParamsClick(name, e) { //事件对象 e 要放在最后
        e.preventDefault();
        alert(name);
    }
}

```

```

    }
    render() {
      return (
        <div>
          { /* 通过箭头函数方法传递参数 */ }
          <button onClick={ (e) => this.passParamsClick(this.state.name,
e)}}>
            {infoMap.btn1info}
          </button>
        </div>
      );
    }
  }
class ParamsEventComp2 extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: infoMap.helloInfo
    };
  }
  passParamsClick(name, e) {
    e.preventDefault();
    alert(name);
  }
  render() {
    return (
      <div>
        { /* 通过 bind()方法绑定传递参数 */ }
        <button onClick={this.passParamsClick.bind(this, this.state.
name)}}>
          {infoMap.btn2info}
        </button>
      </div>
    );
  }
}
const exampleState= (
  <span>
    <ClockReactComp/>
    <hr/>
    <ClockReactComp2/>
    <hr/>
    <h4>{infoMap.title4info}</h4>
    <ClockDataFlow />
    <hr/>
    <h4>数据流应用示例 3</h4>
    <ClockDataFlow2 />
    <hr/>
    <CountReactComp/>

```

```

    <hr/>
    <div>{infoMap.btn1info}</div>
    <ParamsEventComp />
    <hr/>
    <div>{infoMap.btn2info}</div>
    <ParamsEventComp2 />
  </span>
);
ReactDOM.createRoot(divReact).render(exampleState);

```

5.2.2 运行效果

运行文件 `stateexample.html`，效果如图 5-1 所示。注意，由于图 5-1 所示中输出的时间是运行程序的时间，读者在运行该程序时的结果会与图 5-1 不同，读者多次运行同一程序时的结果也不会相同，只要正确运行即可。单击图 5-1 所示中的“箭头函数”按钮，弹出的对话框如图 5-2 所示。单击图 5-1 中的“bind()方法”按钮，弹出的对话框如图 5-3 所示。

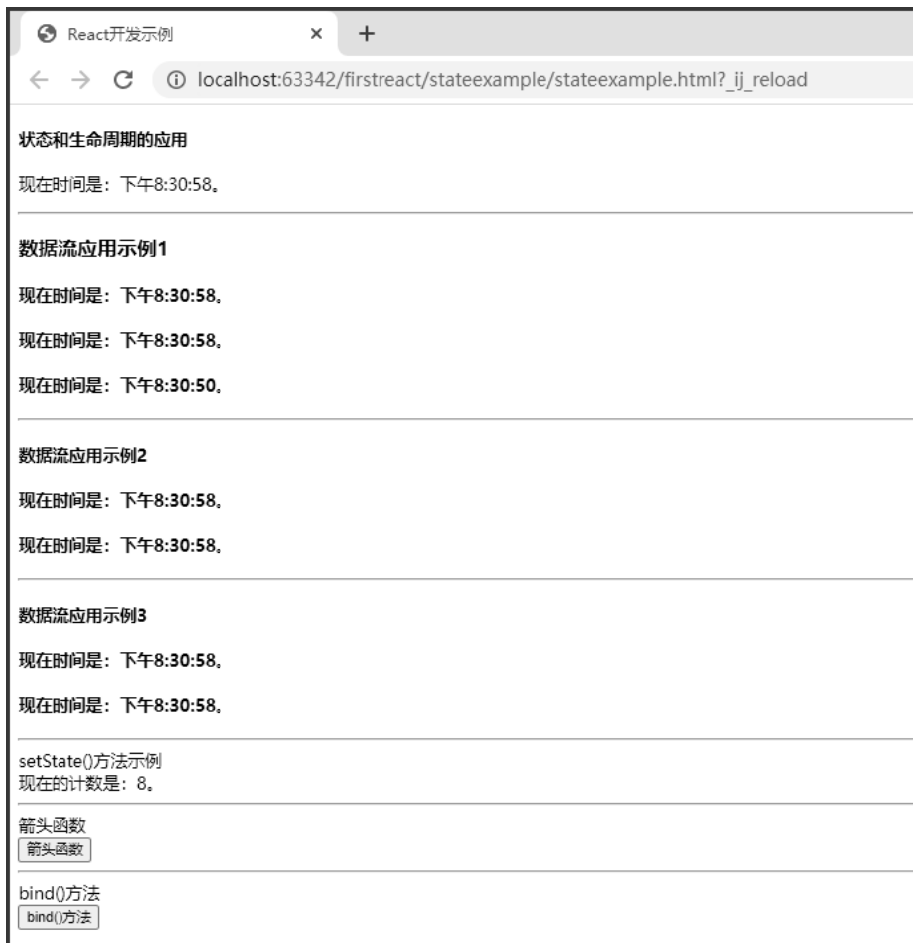


图 5-1 运行文件 `stateexample.html` 的效果

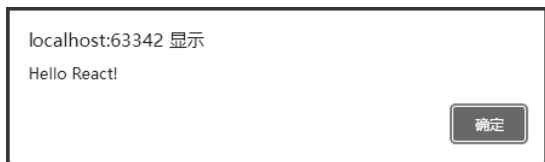


图 5-2 单击图 5-1 所示中的“箭头函数”按钮后弹出的对话框



图 5-3 单击图 5-1 所示中的“bind()方法”按钮后弹出的对话框

5.3 状态的提升应用

5.3.1 开发示例

在项目 firstreact 根目录下创建 advancedstateexample 子目录，在 firstreact\advancedstateexample 目录下创建文件 advancedstateexample.html，代码与如例 1-3 所示的代码相同。在 firstreact\advancedstateexample 目录下创建文件 index.js，代码如例 5-4 所示。

【例 5-4】 在 firstreact\advancedstateexample 目录下创建的文件 index.js 的代码。

```
const divReact = document.getElementById('root');
const infoMap= {
  cTempType: '摄氏度',
  fTempType: '华氏度',
  coolInfo: '今天真冷。',
  warmInfo: '今天气温舒适。',
  hotInfo: '今天有点热。',
  veryHotInfo: '今天真热。',
  howInfo: '今天气温如何？',
  cCoolStandard: 0,
  cWarmStandard: 20,
  cHotStandard: 38,
  fCoolStandard: 32,
  fWarmStandard: 68,
  fHotStandard: 100,
  outputInfo: '请输入气温（',
  outputInfoEnd: '): ',
  title1info: '状态提升示例',
  title2info: '示例 1: 两个文本框的气温数值相同（不考虑摄氏度和华氏度的区别）',
  title3info: '示例 2: 两个文本框的气温相等（考虑了摄氏度和华氏度之间的转换关系）',
}
const tempType = {
  c: infoMap.cTempType,
```



视频讲解