State&生命周期

state

初始值在构造函数中，通过 this.state 赋初值。在 class 组件其他地方通过 this.state.xx 获取该对象的值。

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};  // 在构造函数中赋初始值
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>  {/*在 class 组件其他地方使用*/}
      </div>
    );
  }
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Clock />);
```

生命周期方法

在具有许多组件的应用程序中，当组件被销毁时释放所占用的资源是非常重要的。

- componentDidMount() 方法会在组件已经被渲染到 DOM 中后运行。
- componentWillUnmount() 生命周期方法中清除计时器。

使用 this.setState() 来时刻更新组件 state。调用 setState()，React 能够知道 state 已经改变了，然后会重新调用 render() 方法来确定页面上该显示什么。

正确使用 State

1. 不要直接修改 state

```
// 错误的
this.state.comment = 'Hello';

// 正确的
this.setState({comment: 'Hello'});
```

构造函数是唯一可以给 this.state 赋值的地方

2. State 的更新可能是异步的

处于性能考虑，React 可能会把多个 setState() 调用合并成一个调用。

因为 this.props 和 this.state 可能会异步更新，所以不要依赖他们的值来更新下一个状态。

```
// 错误的写法，不要依赖 this.props 和 this.state 来更新下一个状态
this.setState({
  counter: this.state.counter + this.props.increment,
});
```

要解决这个问题，可以让 setState() 接收一个函数而不是一个对象。这个函数用上一个 state 作为第一个参数，将此次更新被应用时的 props 做为第二个参数：

```
this.setState((state, props) => ({
  counter: state.counter + props.increment
}));
```

3. State 的更新会被合并

当调用 setState() 的时候，React 会把提供的对象合并到当前的 state。

```
constructor(props) {
    super(props);
    this.state = {
        posts: [],
        comments: []
    };
}
```

上面代码中，在构造函数中给 this.state 赋值两个变量 posts 和 comments。

```
componentDidMount() {
    fetchPosts().then(response => {
      this.setState({
        posts: response.posts
      });
    });

    fetchComments().then(response => {
      this.setState({
        comments: response.comments
      });
```

```
      });
    }
```

上面代码在 componentDidMount() 生命周期方法中分别异步设置了 posts 和 comments 的值。

数据流是乡向下流动的

“自上而下”或“单向”数据流。

子组件在其 props 中接收参数 date，但是组件本身无法知道它是来自于父组件的 state 还是 props 还是手动输入的。

任何的 state 总是所属于特定的组件，而且从该 state 派生的任何数据或 UI 只能影响树中“低于”它们的组件。