**把函数式组件转化为类组件**

class Clock extends React.Component {

    render() {

        return (

            <div>

                <h1>Hello, world!</h1>

                <h2>It is {this.props.date.toLocaleTimeString()}.</h2>

            </div>

        );

    }

}

Clock 现在被定为类组件，而不是函数式组件。

类允许我们在其中添加本地状态(state)和生命周期钩子。

**在类组件中添加本地状态(state)**

添加一个 [类构造函数(class constructor)](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes" \l "Constructor) 初始化 this.state:

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>

        );

    }

}

**State 的更新可能是异步的**

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

如下代码index最终的渲染结果是1

class App extends react.Component {

    state = {

      index: 0

    }

    componentDidMount() {

      this.setState({index: this.state.index + 1});

      this.setState({index: this.state.index + 1});

    }

    render(){

      return <div>{ this.state.index }</div>

    }

}

我们修改代码，让其最终的选择结果为2（注意看参数说明）

class App extends react.Component {

    state = {

      index: 0

    }

    componentDidMount() {

      this.setState({index: this.state.index + 1});

      // setState 可接受一个函数

      // state 是上一次 setState 的结果，props 是本次 setState 使用的 props

      this.setState((state, props) => ({index: state.index + 1}));

    }

    render(){

      return <div>{ this.state.index }</div>

    }

}

**在类中添加生命周期方法**

在ES6中，一个React组件是用一个class来表示的（具体可以参考[官方文档](https://link.jianshu.com?t=http://facebook.github.io/react/docs/reusable-components.html" \l "es6-classes" \t "_blank)），如下：

// 定义一个TodoList的React组件，通过继承React.Component来实现

class TodoList extends React.Component {

    // 组件即将挂载时调用

    componentWillMount(){

    }

}

这几个生命周期相关的函数有：

// 构造函数，在创建组件的时候调用一次。

constructor(props, context)

// 在组件挂载之前调用一次。

// 如果在这个函数里面调用setState，本次的render函数可以看到更新后的state，并且只渲染一次。

void componentWillMount()

// 在组件挂载之后调用一次。这个时候，子主键也都挂载好了，可以在这里使用refs。

void componentDidMount()

// 父组件发生render的时候子组件就会调用componentWillReceiveProps（不管props有没有更新，也不管父子组件之间有没有数据交换）

void componentWillReceiveProps(nextProps)

// 每次调用setState后都会调用shouldComponentUpdate判断是否需要重新渲染组件。默认返回true

bool shouldComponentUpdate(nextProps, nextState)

// shouldComponentUpdate返回true或者调用forceUpdate之后，componentWillUpdate会被调用。

void componentWillUpdate(nextProps, nextState)

// 除了首次render之后调用componentDidMount，其它render结束之后都是调用componentDidUpdate。

void componentDidUpdate()

// 必须函数，不要在render里面修改state。

ReactElement render()

// 组件被卸载的时候调用。一般在componentDidMount里面注册的事件需要在这里删除。

void componentWillUnmount()

作者：linjinhe  
链接：https://www.jianshu.com/p/4784216b8194  
來源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。