React Props

React 组件生命周期

React 组件 API

在本章节中我们将讨论 React 组件 API。我们将讲解以下7个方法:

- 设置状态：setState
- 替换状态：replaceState
- 设置属性：setProps
- 替换属性：replaceProps
- 强制更新：forceUpdate
- 获取DOM节点：findDOMNode
- 判断组件挂载状态：isMounted

设置状态:setState

```
setState(object nextState[, function callback])
```

参数说明

- **nextState**，将要设置的新状态，该状态会和当前的**state**合并
- **callback**，可选参数，回调函数。该函数会在**setState**设置成功，且组件重新渲染后调用。

合并nextState和当前state，并重新渲染组件。setState是React事件处理函数中和请求回调函数中触发UI更新的主要方法。

关于setState

不能在组件内部通过this.state修改状态，因为该状态会在调用setState()后被替换。

setState()并不会立即改变this.state，而是创建一个即将处理的state。setState()并不一定是同步的，为了提升性能React会批量执行state和DOM渲染。

setState()总是会触发一次组件重绘，除非在shouldComponentUpdate()中实现了一些条件渲染逻辑。

实例

React 实例

```
class Counter extends React.Component{
constructor(props) {
super(props);
this.state = {clickCount: 0};
this.handleClick = this.handleClick.bind(this);
}
handleClick() {
this.setState(function(state) {
```

```
return {clickCount: state.clickCount + 1};
});
}
render () {
return (<h2 onClick={this.handleClick}>点我! 点击次数为: {this.state.clickCount}</h2>);
}
}
ReactDOM.render(
<Counter />,
document.getElementById('example')
);
```

尝试一下 »

实例中通过点击 h2 标签来使得点击计数器加 1。

替换状态：replaceState

```
replaceState(object nextState[, function callback])
```

- **nextState**，将要设置的新状态，该状态会替换当前的**state**。
- **callback**，可选参数，回调函数。该函数会在**replaceState**设置成功，且组件重新渲染后调用。

replaceState()方法与**setState()**类似，但是方法只会保留**nextState**中状态，原**state**不在**nextState**中的状态都会被删除。

设置属性：setProps

```
setProps(object nextProps[, function callback])
```

- **nextProps**，将要设置的新属性，该状态会和当前的**props**合并
- **callback**，可选参数，回调函数。该函数会在**setProps**设置成功，且组件重新渲染后调用。

设置组件属性，并重新渲染组件。

props相当于组件的数据流，它总是会从父组件向下传递至所有的子组件中。当和一个外部的JavaScript应用集成时，我们可能会需要向组件传递数据或通知**React.render()**组件需要重新渲染，可以使用**setProps()**。

更新组件，我可以在节点上再次调用**React.render()**，也可以通过**setProps()**方法改变组件属性，触发组件重新渲染。

替换属性：replaceProps

```
replaceProps(object nextProps[, function callback])
```

- **nextProps**，将要设置的新属性，该属性会替换当前的**props**。
- **callback**，可选参数，回调函数。该函数会在**replaceProps**设置成功，且组件重新渲染后调用。

replaceProps()方法与**setProps**类似，但它会删除原有 props。

强制更新：forceUpdate

```
forceUpdate([function callback])
```

参数说明

- **callback**，可选参数，回调函数。该函数会在组件**render()**方法调用后调用。

forceUpdate()方法会使组件调用自身的render()方法重新渲染组件，组件的子组件也会调用自己的render()。但是，组件重新渲染时，依然会读取this.props和this.state，如果状态没有改变，那么React只会更新DOM。
forceUpdate()方法适用于this.props和this.state之外的组件重绘（如：修改了this.state后），通过该方法通知React需要调用render()
一般来说，应该尽量避免使用forceUpdate()，而仅从this.props和this.state中读取状态并由React触发render()调用。

获取DOM节点：findDOMNode

```
DOMElement findDOMNode()
```

- 返回值：DOM元素DOMElement

如果组件已经挂载到DOM中，该方法返回对应的本地浏览器 DOM 元素。当**render**返回**null** 或 **false**时，**this.findDOMNode()**也会返回**null**。从DOM 中读取值的时候，该方法很有用，如：获取表单字段的值和做一些 DOM 操作。

判断组件挂载状态：isMounted

```
bool isMounted()
```

- 返回值：**true**或**false**，表示组件是否已挂载到DOM中

isMounted()方法用于判断组件是否已挂载到DOM中。可以使用该方法保证了**setState()**和**forceUpdate()**在异步场景下的调用不会出错。

> *本文参考：http://itbilu.com/javascript/react/EkACBdqKe.html*

React Props

React 组件生命周期

关于 setState() 这里有三件事情需要知道。

1、不要直接更新状态

例如，此代码不会重新渲染组件：

```
// Wrong
this.state.comment = 'Hello';
```

应当使用 setState():

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

构造函数是唯一能够初始化 this.state 的地方。

2、状态更新可能是异步的

React 可以将多个 setState() 调用合并成一个调用来提高性能。

因为 this.props 和 this.state 可能是异步更新的，你不应该依靠它们的值来计算下一个状态。

例如，此代码可能无法更新计数器：

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

要修复它，请使用第二种形式的 setState() 来接受一个函数而不是一个对象。 该函数将接收先前的状态作为第一个参数，将此次更新被应用时的props做为第二个参数：

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

上方代码使用了箭头函数，但它也适用于常规函数：

```
// Correct
this.setState(function(prevState, props) {
  return {
    counter: prevState.counter + props.increment
  };
});
```

3、状态更新合并

当你调用 setState() 时，React 将你提供的对象合并到当前状态。

例如，你的状态可能包含一些独立的变量：

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

```
    };
  }
```

你可以调用 setState() 独立地更新它们：

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

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

这里的合并是浅合并，也就是说 this.setState({comments}) 完整保留了 this.state.posts，但完全替换了 this.state.comments。

疾风拂晓 2个月前 (01-20)

isMounted 的方法在 ES6 中已经废除。主要的原因是它经过实际使用与测试可能不足以检测组件是否挂载，尤其是对于有一些异步的程序情况，以及逻辑上造成混乱。现在用以下方法代替：

```
componentDidMount() {
    this.mounted = true;
}

componentWillUnmount() {
    this.mounted = false;
}
```

lujin 3周前 (02-24)