React.js 小书

<-- 返回首页

组件的 state 和 setState

- 作者：胡子大哈
- 原文链接： http://huziketang.com/books/react/lesson10
- 转载请注明出处，保留原文链接和作者信息。

state

我们前面提到过，一个组件的显示形态是可以由它数据状态和配置参数决定的。一个组件可以拥有自己的状态，就像一个点赞按钮，可以有“已点赞”和“未点赞”状态，并且可以在这两种状态之间进行切换。React.js 的 `state` 就是用来存储这种可变化的状态的。

我们还是拿点赞按钮做例子，它具有已点赞和未点赞两种状态。那么就可以把这个状态存储在 state 中。修改 `src/index.js` 为：

```
import React, { Component } from 'react'
import ReactDOM from 'react-dom'
import './index.css'

class LikeButton extends Component {
  constructor () {
    super()
    this.state = { isLiked: false }
  }

  handleClickOnLikeButton () {
    this.setState({
      isLiked: !this.state.isLiked
    })
  }
```

```
  render () {
    return (
      <button onClick={this.handleClickOnLikeButton.bind(this)}>
        {this.state.isLiked ? '取消' : '点赞'} 👍
      </button>
    )
  }
}
...
```

`isLiked` 存放在实例的 `state` 对象当中，这个对象在构造函数里面初始化。这个组件的 `render` 函数内，会根据组件的 `state` 的中的`isLiked`不同显示“取消”或“点赞”内容。并且给 `button` 加上了点击的事件监听。

最后构建一个 `Index` ，在它的 `render` 函数内使用 `LikeButton` 。然后把 `Index` 渲染到页面上：

```
...
class Index extends Component {
  render () {
    return (
      <div>
        <LikeButton />
      </div>
    )
  }
}

ReactDOM.render(
  <Index />,
  document.getElementById('root')
)
```

setState 接受对象参数

在 `handleClickOnLikeButton` 事件监听函数里面，大家可以留意到，我们调用了 `setState` 函数，每次点击都会更新 `isLiked` 属性为 `!isLiked`，这样就可以做到点赞和取消功能。

`setState` 方法由父类 `Component` 所提供。**当我们调用这个函数的时候，React.js 会更新组件的状态 `state` ，并且重新调用 `render` 方法，然后再把 `render` 方法所渲染的最新的内容显示到页面上。**

注意，当我们要改变组件的状态的时候，不能直接用 `this.state = xxx` 这种方式来修改，如果这样做 React.js 就没办法知道你修改了组件的状态，它也就没有办法更新页面。所以，一定要使用 React.js 提供的 `setState` 方法，**它接受一个对象或者函数作为参数。**

传入一个对象的时候，这个对象表示该组件的新状态。但你只需要传入需要更新的部分就可以了，而不需要传入整个对象。例如，假设现在我们有另外一个状态 `name` ：

```
...
  constructor (props) {
    super(props)
    this.state = {
      name: 'Tomy',
      isLiked: false
    }
  }

  handleClickOnLikeButton () {
    this.setState({
      isLiked: !this.state.isLiked
    })
  }
...
```

因为点击的时候我们并不需要修改 `name`，所以只需要传入 `isLiked` 就行了。Tomy 还是那个 Tomy，而 `isLiked` 已经不是那个 `isLiked` 了。

setState 接受函数参数

这里还有要注意的是，当你调用 `setState` 的时候，**React.js 并不会马上修改 state**。而是把这个对象放到一个更新队列里面，稍后才会从队列当中把新的状态提取出来合并到 `state` 当中，然后再触发组件更新。这一点要好好注意。可以体会一下下面的代码：

```
...
  handleClickOnLikeButton () {
    console.log(this.state.isLiked)
    this.setState({
      isLiked: !this.state.isLiked
    })
    console.log(this.state.isLiked)
  }
...
```

你会发现两次打印的都是 `false`，即使我们中间已经 `setState` 过一次了。这并不是什么 bug，只是 React.js 的 `setState` 把你的传进来的状态缓存起来，稍后才会帮你更新到 `state` 上，所以你获取到的还是原来的 `isLiked`。

所以如果你想在 `setState` 之后使用新的 `state` 来做后续运算就做不到了，例如：

```
...
  handleClickOnLikeButton () {
    this.setState({ count: 0 }) // => this.state.count 还是 undefined
```

```
    this.setState({ count: this.state.count + 1}) // => undefined + 1 = NaN
    this.setState({ count: this.state.count + 2}) // => NaN + 2 = NaN
  }
...
```

上面的代码的运行结果并不能达到我们的预期，我们希望 `count` 运行结果是 `3` ，可是最后得到的是 `NaN`。但是这种后续操作依赖前一个 `setState` 的结果的情况并不罕见。

这里就自然地引出了 `setState` 的第二种使用方式，可以接受一个函数作为参数。React.js 会把上一个 `setState` 的结果传入这个函数，你就可以使用该结果进行运算、操作，然后返回一个对象作为更新 `state` 的对象：

```
...
  handleClickOnLikeButton () {
    this.setState((prevState) => {
      return { count: 0 }
    })
    this.setState((prevState) => {
      return { count: prevState.count + 1 } // 上一个 setState 的返回是 count 为
    })
    this.setState((prevState) => {
      return { count: prevState.count + 2 } // 上一个 setState 的返回是 count 为
    })
    // 最后的结果是 this.state.count 为 3
  }
...
```

这样就可以达到上述的**利用上一次 `setState` 结果进行运算**的效果。

setState 合并

上面我们进行了三次 `setState`，但是实际上组件只会重新渲染一次，而不是三次；这是因为在 React.js 内部会把 JavaScript 事件循环中的消息队列的同一个消息中的 `setState` 都进行合并以后再重新渲染组件。

深层的原理并不需要过多纠结，你只需要记住的是：在使用 React.js 的时候，并不需要担心多次进行 `setState` 会带来性能问题。

课后练习

- 不能摸的狗（二）

> 因为第三方评论工具有问题，对本章节有任何疑问的朋友可以移步到 React.js 小书的论坛 发帖，我会回答大家的疑问。

下一节：配置组件的 props

上一节：事件监听

如果你觉得小书写得还不错，可以请胡子大哈喝杯茶 :)

赞赏

或者传播一下知识也是一个很好的选择