React.js 小书

<-- 返回首页

动手实现 React-redux（三）：connect 和 mapStateToProps

- 作者：胡子大哈
- 原文链接： http://huziketang.com/books/react/lesson38

（本文未审核）

我们来观察一下刚写下的这几个组件，可以轻易地发现它们有两个重大的问题：

1. **有大量重复的逻辑**：它们基本的逻辑都是，取出 context，取出里面的 store，然后用里面的状态设置自己的状态，这些代码逻辑其实都是相同的。
2. **对 context 依赖性过强**：这些组件都要依赖 context 来取数据，使得这个组件复用性基本为零。想一下，如果别人需要用到里面的 `ThemeSwitch` 组件，但是他们的组件树并没有 context 也没有 store，他们没法用这个组件了。

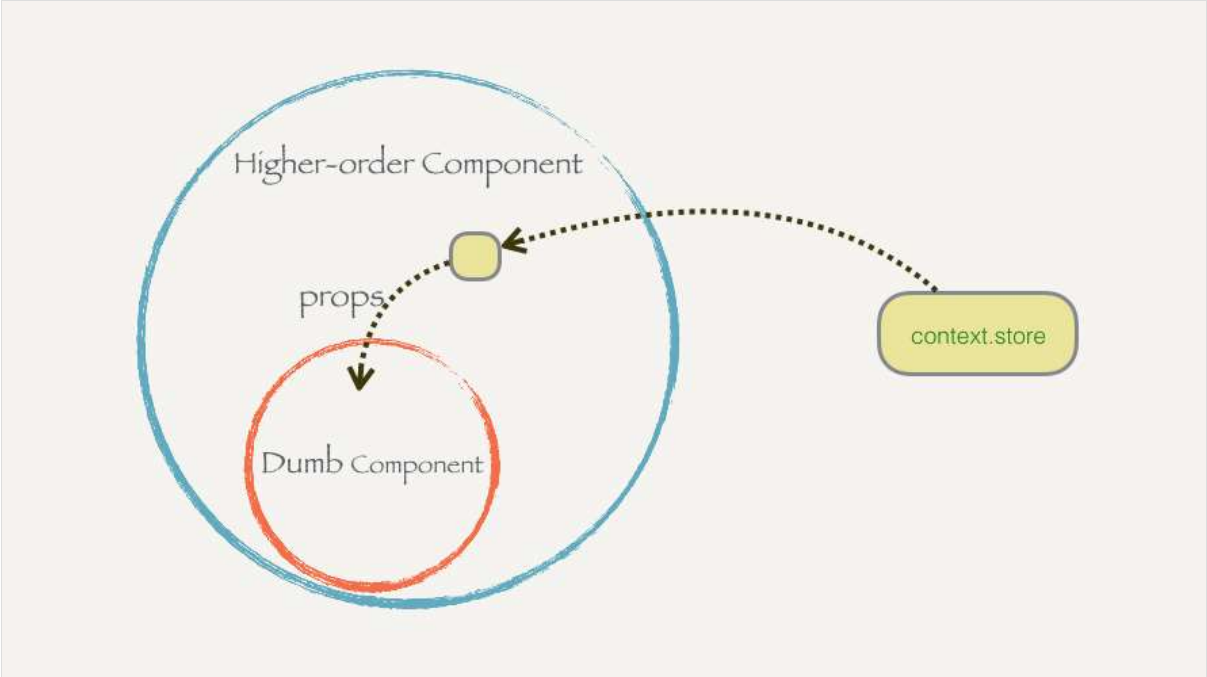
对于第一个问题，我们在 高阶组件 的章节说过，可以把一些可复用的逻辑放在高阶组件当中，高阶组件包装的新组件和原来组件之间通过 `props` 传递信息，减少代码的重复程度。

对于第二个问题，我们得弄清楚一件事情，到底什么样的组件才叫复用性强的组件。如果一个组件对外界的依赖过于强，那么这个组件的移植性会很差，就像这些严重依赖 context 的组件一样。

如果一个组件的渲染只依赖于外界传进去的 `props` 和自己的 `state`，而并不依赖于其他的外界的任何数据，也就是说像纯函数一样，给它什么，它就吐出（渲染）什么出来。这种组件的复用性是最强的，别人使用的时候根本不用担心任何事情，只要看看 `PropTypes` 它能接受什么参数，然后把参数传进去控制它就行了。

我们把这种组件叫做 Pure Compoent，因为它就像纯函数一样，可预测性非常强，对参数（`props`）以外的数据零依赖，也不产生副作用。这种组件也叫 Dumb Component，因为它们呆呆的，让它干啥就干啥。写组件的时候尽量写 Dumb Component 会提高我们的组件的可复用性。

到这里思路慢慢地变得清晰了，我们需要高阶组件帮助我们从 context 取数据，我们也需要写 Dumb 组件帮助我们提高组件的复用性。所以我们尽量多地写 Dumb 组件，然后用高阶组件把它们包装一层，高阶组件和 context 打交道，把里面数据取出来通过 `props` 传给 Dumb 组件。



我们把这个高阶组件起名字叫 `connect`，因为它把 Dumb 组件和 context 连接（connect）起来了：

```
import React, { Component, PropTypes } from 'react'

export connect = (WrappedComponent) => {
  class Connect extends Component {
    static contextTypes = {
      store: PropTypes.object
    }

    // TODO: 如何从 store 取数据？

    render () {
      return <WrappedComponent />
    }
  }

  return Connect
}
```

`connect` 函数接受一个组件 `WrappedComponent` 作为参数，把这个组件包含在一个新的组件 `Connect` 里面，`Connect` 会去 context 里面取出 store。现在要把 store 里面的数据取出来通过 `props` 传给 `WrappedComponent`。

但是每个传进去的组件需要 store 里面的数据都不一样的，所以除了给高阶组件传入 Dumb 组件以外，还需要告诉高级组件我们需要什么数据，高阶组件才能正确地去取数据。为了解决这个问题，我们可以给高阶组件传入类似下面这样的函数：

```
const mapStateToProps = (state) => {
  return {
    themeColor: state.themeColor,
```

```
    themeName: state.themeName,
    fullName: `${state.firstName} ${state.lastName}`
    ...
  }
}
```

这个函数会接受 `store.getState()` 的结果作为参数，然后返回一个对象，这个对象是根据 `state` 生成的。`mapStateTopProps` 相当于告知了 `Connect` 应该如何去 store 里面取数据，然后可以把这个函数的返回结果传给被包装的组件：

```
import React, { Component, PropTypes } from 'react'

export const connect = (mapStateToProps) => (WrappedComponent) => {
  class Connect extends Component {
    static contextTypes = {
      store: PropTypes.object
    }

    render () {
      const { store } = this.context
      let stateProps = mapStateToProps(store.getState())
      // {...stateProps} 意思是把这个对象里面的属性全部通过 `props` 方式传递进去
      return <WrappedComponent {...stateProps} />
    }
  }

  return Connect
}
```

`connect` 现在是接受一个参数 `mapStateToProps`，然后返回一个函数，这个返回的函数才是高阶组件。它会接受一个组件作为参数，然后用 `Connect` 把组件包装以后再返回。 `connect` 的用法是：

```
...
const mapStateToProps = (state) => {
  return {
    themeColor: state.themeColor
  }
}
Header = connect(mapStateToProps)(Header)
...
```

> 有些朋友可能会问为什么不直接 `const connect = (mapStateToProps, WrappedComponent)`，而是要额外返回一个函数。这是因为 React-redux 就是这么设计的，而个人观点认为这是一个 React-redux 设计上的缺陷，这里有机会会在关于函数编程的章节再给大家科普，这里暂时不深究了。

我们把上面 `connect` 的函数代码单独分离到一个模块当中，在 `src/` 目录下新建一个 `react-redux.js`，把上面的 `connect` 函数的代码复制进去，然后就可以在 `src/Header.js` 里面使用了：

```
import React, { Component, PropTypes } from 'react'
import { connect } from './react-redux'

class Header extends Component {
  static propTypes = {
    themeColor: PropTypes.string
  }

  render () {
    return (
      <h1 style={{ color: this.props.themeColor }}>React.js 小书</h1>
    )
  }
}

const mapStateToProps = (state) => {
  return {
    themeColor: state.themeColor
  }
}
Header = connect(mapStateToProps)(Header)

export default Header
```

可以看到 `Header` 删掉了大部分关于 context 的代码，它除了 `props` 什么也不依赖，它是一个 Pure Component，然后通过 `connect` 取得数据。我们不需要知道 `connect` 是怎么和 context 打交道的，只要传一个 `mapStateToProps` 告诉它应该怎么取数据就可以了。同样的方式修改 `src/Content.js`：

```
import React, { Component, PropTypes } from 'react'
import ThemeSwitch from './ThemeSwitch'
import { connect } from './react-redux'

class Content extends Component {
  static propTypes = {
    themeColor: PropTypes.string
  }

  render () {
    return (
      <div>
        <p style={{ color: this.props.themeColor }}>React.js 小书内容</p>
        <ThemeSwitch />
      </div>
    )
  }
}
```

```
const mapStateToProps = (state) => {
  return {
    themeColor: state.themeColor
  }
}
Content = connect(mapStateToProps)(Content)

export default Content
```

`connect` 还没有监听数据变化然后重新渲染，所以现在点击按钮只有按钮会变颜色。我们给 `connect` 的高阶组件增加监听数据变化重新渲染的逻辑，稍微重构一下 `connect`：

```
export const connect = (mapStateToProps) => (WrappedComponent) => {
  class Connect extends Component {
    static contextTypes = {
      store: PropTypes.object
    }

    constructor () {
      super()
      this.state = { allProps: {} }
    }

    componentWillMount () {
      const { store } = this.context
      this._updateProps()
      store.subscribe(() => this._updateProps())
    }

    _updateProps () {
      const { store } = this.context
      let stateProps = mapStateToProps(store.getState(), this.props) // 额外传入
      this.setState({
        allProps: { // 整合普通的 props 和从 state 生成的 props
          ...stateProps,
          ...this.props
        }
      })
    }

    render () {
      return <WrappedComponent {...this.state.allProps} />
    }
  }

  return Connect
}
```

我们在 `Connect` 组件的 `constructor` 里面初始化了 `state.allProps`，它是一个对象，用来保存需要传给被包装组件的所有的参数。生命周期 `componentWillMount` 会调用调用 `_updateProps` 进行初始化，然后通过 `store.subscribe` 监听数据变化重新调用 `_updateProps`。

为了让 connect 返回新组件和被包装的组件使用参数保持一致，我们会把所有传给 `Connect` 的 `props` 原封不动地传给 `WrappedComponent`。所以在 `_updateProps` 里面会把 `stateProps` 和 `this.props` 合并到 `this.state.allProps` 里面，再通过 `render` 方法把所有参数都传给 `WrappedComponent`。

`mapStateToProps` 也发生点变化，它现在可以接受两个参数了，我们会把传给 `Connect` 组件的 `props` 参数也传给它，那么它生成的对象配置性就更强了，我们可以根据 `store` 里面的 `state` 和外界传入的 `props` 生成我们想传给被包装组件的参数。

现在已经很不错了，`Header.js` 和 `Content.js` 的代码都大大减少了，并且这两个组件 connect 之前都是 Dumb 组件。接下来会继续重构 `ThemeSwitch`。

> 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：动手实现 React-redux（四）：mapDispatchToProps

上一节：动手实现 React-redux（二）：结合 context 和 store

3 条评论，3 人参与。

8

我有话说...

使用社交帐号登录

发布前先点击左边的按钮登录

最新评论

manong • 昨天 10:30

非常感谢。以前搞不清楚为啥用connector，看了这篇就明白了

顶 • 回复 • 分享»

可鱼不是鱼 • 6月28日 11:01

高级函数搞晕了

顶 • 回复 • 分享»

_chad • 5月2日 01:14

棒棒的

顶 • 回复 • 分享»

友言?