React.js 小书

<-- 返回首页

动手实现 React-redux（四）：mapDispatchToProps

- 作者：胡子大哈
- 原文链接： http://huziketang.com/books/react/lesson39

（本文未审核）

在重构 `ThemeSwitch` 的时候我们发现，`ThemeSwitch` 除了需要 `store` 里面的数据以外，还需要 `store` 来 `dispatch`：

```
...
  // dispatch action 去改变颜色
  handleSwitchColor (color) {
    const { store } = this.context
    store.dispatch({
      type: 'CHANGE_COLOR',
      themeColor: color
    })
  }
...
```

目前版本的 `connect` 是达不到这个效果的，我们需要改进它。

想一下，既然可以通过给 `connect` 函数传入 `mapStateToProps` 来告诉它如何获取、整合状态，我们也可以想到，可以给它传入另外一个参数来告诉它我们的组件需要如何触发 `dispatch`。我们把这个参数叫 `mapDispatchToProps`：

```
const mapDispatchToProps = (dispatch) => {
  return {
    onSwitchColor: (color) => {
      dispatch({ type: 'CHANGE_COLOR', themeColor: color })
    }
  }
}
```

和 `mapStateToProps` 一样，它返回一个对象，这个对象内容会同样被 `connect` 当作是 `props` 参数传给被包装的组件。不一样的是，这个函数不是接受 `state` 作为参数，而是 `dispatch`，你可以在返回的对象内部定义一些函数，这些函数会用到 `dispatch` 来触发特定的 `action`。

调整 `connect` 让它能接受这样的 `mapDispatchToProps`：

```
export const connect = (mapStateToProps, mapDispatchToProps) => (WrappedComponen
  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
        ? mapStateToProps(store.getState(), this.props)
        : {} // 防止 mapStateToProps 没有传入
      let dispatchProps = mapDispatchToProps
        ? mapDispatchToProps(store.dispatch, this.props)
        : {} // 防止 mapDispatchToProps 没有传入
      this.setState({
        allProps: {
          ...stateProps,
          ...dispatchProps,
          ...this.props
        }
      })
    }

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

在 `_updateProps` 内部，我们把`store.dispatch` 作为参数传给 `mapDispatchToProps`，它会返回一个对象 `dispatchProps`。接着把 `stateProps`、`dispatchProps`、`this.props` 三者合并到 `this.state.allProps` 里面去，这三者的内容都会在 `render` 函数内全部传给被包装的组件。

另外，我们稍微调整了一下，在调用 `mapStateToProps` 和 `mapDispatchToProps` 之前做判断，让这两个参数都是可以缺省的，这样即使不传这两个参数程序也不会报错。

这时候我们就可以重构 `ThemeSwitch`，让它摆脱 `store.dispatch`：

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

class ThemeSwitch extends Component {
  static propTypes = {
    themeColor: PropTypes.string,
    onSwitchColor: PropTypes.func
  }

  handleSwitchColor (color) {
    if (this.props.onSwitchColor) {
      this.props.onSwitchColor(color)
    }
  }

  render () {
    return (
      <div>
        <button
          style={{ color: this.props.themeColor }}
          onClick={this.handleSwitchColor.bind(this, 'red')}>Red</button>
        <button
          style={{ color: this.props.themeColor }}
          onClick={this.handleSwitchColor.bind(this, 'blue')}>Blue</button>
      </div>
    )
  }
}

const mapStateToProps = (state) => {
  return {
    themeColor: state.themeColor
  }
}
const mapDispatchToProps = (dispatch) => {
  return {
    onSwitchColor: (color) => {
      dispatch({ type: 'CHANGE_COLOR', themeColor: color })
    }
  }
}
ThemeSwitch = connect(mapStateToProps, mapDispatchToProps)(ThemeSwitch)

export default ThemeSwitch
```

光看 `ThemeSwitch` 内部，是非常清爽干净的，只依赖外界传进来的 `themeColor` 和 `onSwitchColor`。但是 `ThemeSwitch` 内部并不知道这两个参数其实都是我们去 `store`

里面取的，它是 Dumb 的。这时候这三个组件的重构都已经完成了，代码大大减少、不依赖 context，并且功能和原来一样。

> 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（五）：Provider

上一节：动手实现 React-redux（三）：connect 和 mapStateToProps