React.js 小书

<-- 返回首页

动手实现 React-redux（六）：React-redux 总结

- 作者：胡子大哈
- 原文链接： http://huziketang.com/books/react/lesson41

（本文未审核）

到这里大家已经掌握了 React-redux 的基本用法和概念，并且自己动手实现了一个 React-redux，我们回顾一下这几节都干了什么事情。

React.js 除了状态提升以外并没有更好的办法帮我们解决组件之间共享状态的问题，而使用 context 全局变量让程序不可预测。通过 Redux 的章节，我们知道 store 里面的内容是不可以随意修改的，而是通过 dispatch 才能变更里面的 state。所以我们尝试把 store 和 context 结合起来使用，可以兼顾组件之间共享状态问题和共享状态可能被任意修改的问题。

第一个版本的 store 和 context 结合有诸多缺陷，有大量的重复逻辑和对 context 的依赖性过强。我们尝试通过构建一个高阶组件 `connect` 函数的方式，把所有的重复逻辑和对 context 的依赖放在里面 `connect` 函数里面，而其他组件保持 Pure（Dumb） 的状态，让 `connect` 跟 context 打交道，然后通过 `props` 把参数传给普通的组件。

而每个组件需要的数据和需要触发的 action 都不一样，所以调整 `connect`，让它可以接受两个参数 `mapStateToProps` 和 `mapDispatchToProps`，分别用于告诉 `connect` 这个组件需要什么数据和需要触发什么 action。

最后为了把所有关于 context 的代码完全从我们业务逻辑里面清除掉，我们构建了一个 `Provider` 组件。`Provider` 作为所有组件树的根节点，外界可以通过 `props` 给它提供 store，它会把 store 放到自己的 context 里面，好让子组件 connect 的时候都能够获取到。

这几节的成果就是 `react-redux.js` 这个文件里面的两个内容：`connect` 函数和 `Provider` 容器组件。这就是 React-redux 的基本内容，当然它是一个残疾版本的 React-redux，很多地方需要完善。例如上几节提到的性能问题，现在不相关的数据变化的时候其实所有组件都会重新渲染的，这个性能优化留给读者做练习。

通过这种方式大家不仅仅知道了 React-redux 的基础概念和用法，而且还知道这些概念到底是解决什么问题，为什么 React-redux 这么奇怪，为什么要 connect，为什么要 mapStateToProps 和 mapDispatchToProps，什么是 Provider，我们通过解决一个个问题就知道它们到底为什么要这么设计的了。

> 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：使用真正的 Redux 和 React-redux

上一节：动手实现 React-redux（五）：Provider