UniqueWeb 2020 夏令营 前端第3期任务

Intro

在前端应用的开发中，状态管理一直是炙手可热的话题。流行的现代前端框架，大多是建立了从状态（state）到视图（view）的映射，前端开发在某种意义上即是应用状态的维护。应用的路由，用户的输入，组件的数据……都可以算是状态。

当应用规模较小时，在视图模型中对状态进行手动变更是可行的。然而随着应用规模增长，应用功能复杂，组件的抽象程度变高，需要被维护的状态也飞速增加。举例来说，实现一个数独游戏只需要管理棋盘大小和棋盘落子的状态；而实现大富翁桌游则需要管理玩家、地皮、回合、牌堆相关的全部状态。这些状态之间可能存在着复杂的逻辑关系，某一个组件的状态会被另一个组件使用。若此时状态与视图仍高度耦合，那么维护组件树将变得十分困难（考虑一个多层组件树间各组件状态的互相依赖），对状态进行有效管理所带来的心智负担会相当可观。

将分散的应用状态集中管理，对状态的更新进行合理按需分发，便是状态管理库的主要存在意义之一。随着主流前端框架的发展和对状态管理需求的变化，诸如 `flux`、`redux` 和 `mobx` 这般优秀的状态管理工具层出不穷。

本期任务将假设（事实上也是如此）你掌握了任意一门主流前端框架以及相关状态管理工具的基本用法（例如 Vue + Vuex / React + Redux），并要求你通过手写一个简单的状态管理工具 `uniquedux` 来掌握这一概念。

本次任务的 DDL 是 2020 年 7 月 26 日 18:00。

目标

1. 学习和掌握基本的 HTML 和 CSS 知识。
2. 初步了解 JavaScript 的语s言特性。
3. 理解 DOM 和 XHR 。

Redux 和其他类 Flux 库

阅读 Redux 官方文档 中 *Introduction* 的**全部小节**与 *Basics* 的**前四小节**。

提示：可以适当做笔记。

提示：运行 *Introduction* 一章 *Example* 中 Counter Vanilla，尝试修改代码并观察运行结果。理解 这个例子后续任务很有帮助。

提示：如果你熟悉 Vue 和 Vuex，你可能会对 Redux 的设计模式和 API 风格感到熟悉。

实战：uniquedux

实现一个 redux 风格的状态管理工具 uniquedux。

要求实现：

1. combineReducers 方法，接受若干个 reducer 作为参数，将其合并为单独一个 reducer 并返回。
2. Store(reducer, initState) 类，以**对象**的形式存储应用的状态:
    1. 具有 Store.getState() 方法，获取整个状态对象。
    2. 具有 Store.dispatch(action) 方法，为 reducer 分发一个 action。
    3. 具有 Store.subscribe(handler) 方法，接收一个函数 handler。当状态发生改变时调用该函数（注意一个 store 可以被多次订阅）。
    4. 具有 Store.replaceReducer(newReducer) 方法，改变 store 的 reducer。
3. 使用 uniquedux 替换 redux，改写示例 Counter Vanilla。

可选： connector

我们期望 uniquedux 可以在框架的实际应用中更加灵活和简洁。

在 React 生态中，往往会使用 react-redux 包提供的 connect 方法，来实现从 store 到组件 props 的连接，这大大减少了样版代码的数量并简化了对状态的操作。

在 Vue 生态中，vuex 也实现类似的逻辑，例如：

```
new Vue({
  el: '#app',
  store: store,
})
```

基于你熟悉的技术栈，尝试写一个 uniquedux-connector 来实现从 uniquedux 到组件的连接。

Tips

1. 若存在任何对题目的疑问和学习上的困难，feel free 向我们寻求帮助。