H2 Redux和Vuex核心状态管理

H3 回忆一下第一周老袁讲了什么

1. Redux/React自身良好的架构、先进的理念，加上一系列优秀的第三方插件的支持，是React/Redux成功的关键所在。
2. 可以将React看作输入为state，输出为view的“纯”函数。
3. 范畴论将世界抽象为对象和对象之间的联系，Redux将所有事件抽象为action。
4. Container 中含有 **value 和 map 两个属性，而修改** value 的方法只有 map，在操作完 value 后将新值放回 Container 中。

```
// 如何操作或修改 value 由 f 给出。
store -> container
currentState -> __value
action -> f
currentReducer -> map
middleware -> IO functor (解决异步操作的各种问题。)
```

5. store 是一个容器含有 state 和 reducer，

> reducer是一个纯函数，它可以查看之前的状态，执行一个action并且返回一个新的状态。

这从 store 的创建语句 enhancer(createStore)(reducer, preloadedState) 可以很明显的得出。而修改 store 中的 currentState 的唯一方法是使用 currentReducer，并且 currentState 在修改完后将新值依然存放在 store 内。

H3 如何修改 currentState 是根据用户操作 action 。

1. applyMiddlewar.js 使用自定义的 middleware 来扩展 Redux
2. bindActionCreators.js 把 action creators 转成拥有同名 keys 的对象,使用时可以直接调用
3. combineReducers.js 一个比较大的应用，需要对 reducer 函数 进行拆分，拆分后的每一块独立负责管理 state 的一部分
4. compose.js 从右到左来组合多个函数，函数编程中常用到
5. createStore.js 创建一个 Redux Store 来放所有的state
6. utils/warnimng.js 控制台输出一个警告，我们可以不用看
7. React可以看做纯函数 固定的state输入输出组件

H4 Redux Store的基础

`store` 是一个单一对象：

- 管理应用的 state
- 通过 store.getState() 可以获取 state
- 通过 store.dispatch(action) 来触发 state 更新
- 通过 store.subscribe(listener) 来注册 state 变化监听器
- 通过 createStore(reducer, [initialState]) 创建

H4 React-Redux的原理

Provider 其实就只是一个外层容器，它的作用就是通过配合 **connect** 来达到跨层级传递数据。使用时只需将Provider定义为整个项目最外层的组件，并设置好store。那么整个项目都可以直接获取这个store。它的原理其实是通过React中的Context来实现的。它大致的核心代码如下：

```
import React, {Component} from 'react'
import {PropTypes} from 'prop-types'

export default class Provider extends Component {
    getChildContext() {
        return {store: this.props.store}
    }

    constructor() {
        super()

        this.state = {}
    }

    render() {
        return this.props.children
    }
}

Provider.childContextTypes = {
    store: PropTypes.object
}
```

connect 的作用是连接React组件与 Redux store，它包在我们的容器组件的外一层，它接收上面 Provider 提供的 store 里面的 state 和 dispatch，传给一个构造函数，返回一个对象，以属性形式传给我们的容器组件。

它共有四个参数mapStateToProps, mapDispatchToProps, mergeProps以及options。

mapStateToProps 的作用是将store里的state（数据源）绑定到指定组件的props中 **mapDispatchToProps** 的作用是将store里的action（操作数据的方法）绑定到指定组件的props中 另外两个方法一般情况下使用不到，这里就不做介绍。。

那么 **connect** 是怎么将React组件与 Redux store连接起来的呢？其主要逻辑可以总结成以下代码：

```
import {Component} from "react";
import React from "react";
import {PropTypes} from 'prop-types'
```

```
const connect = (mapStateToProps, mapDispatchToProps) =>
(WrappedComponent => {
    class Connect extends Component {
        constructor() {
            super()

            this.state = {}

        }

        componentWillMount() {
            this.unSubscribe = this.context.store.subscribe(() => {

 this.setState(mapStateToProps(this.context.store.getState()))
            })
        }

        componentWillUnmount() {
            this.unSubscribe()
        }

        render() {
            return <WrappedComponent  {...this.state}

{...mapDispatchToProps(this.context.store.dispatch)}/>
        }
    }

    Connect.contextTypes = {
        store: PropTypes.object
    }
    return Connect
})

export default connect
```