day07-Redux状态管理

1. Redux

### 简介

随着单页面应用的普及，web app内部需要管理的状态越来越多，这些状态可能来自服务器端，用户输入的数据，用户交互数据，当前UI状态，本地的缓存数据等等。如何能够有条理的管理这些数据，成为前端开发中一个难题.

Redux 是 JavaScript 状态管理库， 提供可预测的状态管理。

那什么是可预测，可以理解为根据一个固定的输入，必然会得到一个固定的结果.

### 三大原则

**单一数据源**, 整个应用的 state 被储存在一棵 object tree 中，并且这个 object tree 只存在于唯一一个 store 中.

console.log( store.getState() )

**State 是只读的**, 唯一改变 state 的方法就是触发 action，action 是一个用于描述已发生事件的普通对象

store.dispatch({

type: 'DELETE\_TODO',

index: 1

})

**使用纯函数来执行修改**, 描述 action 如何改变 state tree ，你需要编写 reducers。

### 使用Redux

#### store

store是redux应用的唯一数据源，我们调用createStore Api创建store.

我们知道 action 用来描述“发生了什么”， reducers 用来根据 action 更新 state;

const store = createStore( reducer, initState, applyMiddleware( thrunk ) );

Store 就是把它们联系到一起的对象。Store 有以下功能：

1. 提供 getState() 方法获取 state；

2. 提供 dispatch(action) 方法更新 state；

3. 通过 subscribe(listener) 注册监听器;

4. 通过 subscribe(listener) 返回的函数注销监听器。

#### action

Action 本质上是 JavaScript 普通对象。我们约定，action 内必须使用一个字符串类型的 type 字段来表示将要执行的动作。多数情况下，type 会被定义成字符串常量。当应用规模越来越大时，建议使用单独的模块或文件来存放 action。

除了 type 字段外，action 对象的结构完全由你自己决定;

我们应该尽量减少在 action 中传递的数据。比如下面的例子，传递 index 就比把整个任务对象传过去要好。

{

type: 'ADD\_TODO',

index: 5

}

#### action creator

Action 创建函数 就是生成 action 的方法;

在 Redux 中的 action 创建函数只是简单的返回一个 action:

function addTodo(text) {

return {

type: ADD\_TODO,

text

}

}

#### reducer

reducer 就是一个纯函数，接收旧的 state 和 action，返回新的 state。

(prevState, action) => newState

保持 reducer 纯净非常重要。永远不要在 reducer 里做这些操作：

1. 修改传入参数；

2. 执行有副作用的操作，如 API 请求和路由跳转；

3. 调用非纯函数，如 Date.now() 或 Math.random()。

function todoApp(state = initialState, action) {

switch (action.type) {

case SET\_VISIBILITY\_FILTER:

return Object.assign({}, state, {af: action.filter});

case ADD\_TODO:

return Object.assign({}, state, {})

case TOGGLE\_TODO:

return Object.assign({}, state, )

})

default: return state;

}

}

注意:

1. 不要修改 state。 使用 Object.assign() 新建了一个副本。不能这样使用 Object.assign(state, { visibilityFilter: action.filter })，因为它会改变第一个参数的值。你必须把第一个参数设置为空对象。你也可以使用 { ...state, ...newState } 达到相同的目的。

2. 在 default 情况下返回旧的 state。遇到未知的 action 时，一定要返回旧的 state。

### 数据流

严格的单向数据流是 Redux 架构的设计核心。

Redux 应用中数据的生命周期遵循下面 4 个步骤：

1. store调用 store.dispatch(action)。

你可以在任何地方调用 store.dispatch(action)，包括组件中、XHR 回调中、甚至定时器中;

2. store 调用传入的 reducer 函数。

Store 会把两个参数传入 reducer： 当前的 state 树和 action。

3. 根 reducer 把多个子 reducer 输出合并成一个单一的 state 树.

Redux 原生提供combineReducers()辅助函数，来把根 reducer 拆分成多个函数，用于分别处理 state 树的一个分支。

4. store 保存了根 reducer 返回的完整 state 树。

这个新的树就是应用的下一个 state！所有订阅 store.subscribe(listener) 的监听器都将被调用；监听器里可以调用 store.getState() 获得当前 state。

现在，可以应用新的 state 来更新 UI。如果你使用了 React Redux 这类的绑定库，这时就应该调用 this.setState(newState) 来更新。

### 代码模块化

当数据量过大时统一到一个store文件下会变得臃肿和难以管理, 可以合理拆分为几个模块

index.js、actionTypes.js、actionCreators.js、reducer.js

### react-redux

react-redux库是对 react与redux绑定的一个库，这样是为了保证核心功能做到最大程度的跨平台复用。

react-redux提供了一个组件和一个API帮助React与Redux进行绑定，一个是React组件 <Provider/>，一个是connect()。

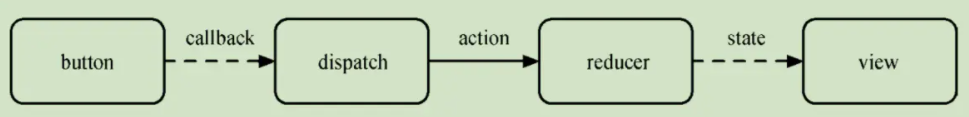
<Provider/>接受一个store作为props,它是整个应用的顶层组件.

connect（）提供了在整个React应用的任意组件中获取store中数据的功能.

### redux中间件

#### 概述

Redux本身就提供了非常强大的数据流管理功能，但这并不是它唯一的强大之处， 它还提供了利用中间件来扩展自身功能，以满足用户的开发需求.

无中间件情况下redux数据流:

增加了中间件 redux数据流:

派发给 redux Store 的 action 对象，会被 Store 上的多个中间件依次处理，如果把 action 和当前的 state 交给 reducer 处理的过程看做默认存在的中间件，那么其实所有的对 action 的处理都可以有中间件组成的。值得注意的是这些中间件会按照指定的顺序依次处理传入的 action，只有排在前面的中间件完成任务之后，后面的中间件才有机会继续处理 action，同样的，每个中间件都有自己的“熔断”处理,当它认为这个 action 不需要后面的中间件进行处理时，后面的中间件也就不能再对这个 action 进行处理了。

而不同的中间件之所以可以组合使用，是因为 Redux 要求所有的中间件必须提供统一的接口，每个中间件的尉氏县逻辑虽然不一样，但只要遵循统一的接口就能和redux以及其他的中间件对话了

#### 理解中间件的机制

一个啥都不干的中间件.

const doNothingMidddleware = ({dispatch, getState}) => next => action => next(action)

上面这个函数接受一个对象作为参数，对象的参数上有两个字段 dispatch 和 getState，分别代表着 Redux Store 上的两个同名函数，但需要注意的是并不是所有的中间件都会用到这两个函数。然后 doNothingMidddleware 返回的函数接受一个 next 类型的参数，这个 next 是一个函数，如果调用了它，就代表着这个中间件完成了自己的职能，并将对 action 控制权交予下一个中间件。但需要注意的是，这个函数还不是处理 action 对象的函数，它所返回的那个以 action 为参数的函数才是。最后以 action 为参数的函数对传入的 action 对象进行处理，在这个地方可以进行操作，比如：

调动dispatch派发一个新 action 对象

调用 getState 获得当前 Redux Store 上的状态

调用 next 告诉 Redux 当前中间件工作完毕，让 Redux 调用下一个中间件

访问 action 对象 action 上的所有数据

#### redux-thunk的使用

redux-thunk允许在action creator中返回一个函数, 我们可以在该函数中实现异步操作.

一个返回值为函数的action creator

function get\_films() {

return (dispatch) => {

axios.get('/gateway?cityId=330100&pageNum=1&pageSize=10&type=1&k=5536279').then((res) => {

let films = res.data.data.films;

dispatch(init\_films\_data(films))

})

}

}

function init\_films\_data(param) {

return { type: "INIT\_FILMS\_DATA", param }

}

应用redux-thunk中间件

const store = createStore( reducer, applyMiddleware( thunk ) );

redux-saga是另外一个管理redux应用异步操作的中间件，功能和redux-thunk类似, 可以用于代替 redux-thunk .

#### redux-logger的使用

redux-logger是一个打印数据操作日志的中间件.

安装redux-logger

npm i redux-logger –save

应用redux-logger中间件

const store = createStore( reducer, applyMiddleware( thunk, logger ) );

注意: redux-logger中间件应该放在所有中间件的最后一个，因为有的action可能不是一个平面对象，可能是一个函数（副作用函数），这时候需要在前面通过一些其它的中间件去处理，不然redux-logger会显示错误.

1. 综合案例

<cdf-会员购> 详情页,加入购物车功能