1.redux初步使用

import {createStore} from 'redux';  
// 1.建立store  
// 通过reducer建立  
// 根据老的state和action，生成新的state  
// state=0默认(初始值)是0，表示如果没有state就是为0；  
function counter(state=0,action){  
 switch(action.**type**){  
 case '加机关枪':  
 return state+1;  
 case '减机关枪':  
 return state-1;  
 default: //什么行为都没有，默认返回10  
 return 10;  
 }  
}  
  
// 1.新建store  
const store=createStore(counter);  
  
const init=store.getState();  
  
// console.log(init);  
  
// 2.派发事件，传递action  
  
// 第一种写法，冗余  
// store.dispatch({type:'加机关枪'});  
// console.log(store.getState());  
// store.dispatch({type:'加机关枪'});  
// console.log(store.getState());  
// store.dispatch({type:'减机关枪'});  
// console.log(store.getState());  
  
// 我们可以用第二种写法，订阅事件  
function listener(){  
 const current=store.getState();  
 console.log(`现在有机枪${current}把`);  
}  
// 每派发一次事件就触发一次订阅  
store.subscribe(listener);  
store.dispatch({**type**:'加机关枪'});  
store.dispatch({**type**:'加机关枪'});  
store.dispatch({**type**:'减机关枪'});

2.redux+react使用

1)同步

Index.js

import App from './App.js';  
import {createStore} from 'redux';  
import {counter,addGUN,removeGUN} from './index.redux.js';  
// 创建store  
var store=createStore(counter);  
function render(){  
 ReactDom.render(  
 <App  
 store={store}  
 addGUN={addGUN}  
 removeGUN={removeGUN}  
 />,  
 document.getElementById('root')  
 );  
}  
render()  
// 每派发一次事件就重新渲染组件  
store.subscribe(render);

index.redux.js

const ADD\_GUN='加机关枪';  
const REMOVE\_GUN='减机关枪';  
// 1.创建redux  
export function counter(state=0,action){  
 switch(action.**type**){  
 case '加机关枪':  
 return state+1;  
 case '减机关枪':  
 return state-1;  
 default: //什么行为都没有，默认返回10  
 return 10;  
 }  
}  
// 2.通过函数返回action  
export function addGUN(){  
 return {**type**:ADD\_GUN}  
}  
export function removeGUN(){  
 return {**type**:REMOVE\_GUN}  
}

App.js

render(){  
 var store=this.**props**.store;  
 var addGUN =this.**props**.addGUN;  
 var removeGUN= this.**props**.removeGUN;  
 return (  
 <div>  
 <h1>现在有机枪{store.getState()}把</h1>  
 <button onClick={()=>{store.dispatch(addGUN())}}>加机关枪</button>  
 <button onClick={()=>{store.dispatch(removeGUN())}}>上交关枪</button>  
 </div>  
 )  
}

2)异步

Index.js

// applyMiddleware处理异步，需要import thunk from 'redux-thunk';  
var store=createStore(counter,applyMiddleware(thunk));

Index.redux.js

// 异步action  
export function addGUNasync(){

//主动派发行为  
 return dispath=>{  
 setTimeout(()=>{  
 dispath(addGUN());  
 },3000)  
 }  
}

App.js

var addGUNasync=this.**props**.addGUNasync;

<button onClick={store.dispatch(addGUNasync())}>拖两天再交</button>

3.在谷歌浏览器中启用redux DEVTools

Index.js

import {createStore,applyMiddleware,compose} from 'redux';

var store=createStore(counter,compose(  
 applyMiddleware(thunk),  
 window.devToolsExtension?window.devToolsExtension():f=>f  
));

4.react-redux使用

与上面的差不多，需要改变的地方有

Index.js

import {**Provider**} from 'react-redux';

ReactDom.render(  
 <Provider store={store}>  
 <App/>  
 </Provider>,  
 document.getElementById('root')  
 );

App.js

import { addGUN,addGUNasync,removeGUN} from './index.redux.js';  
import {**connect**} from 'react-redux'  
  
class App extends React.Component {  
 // 使用react-redux  
 render(){  
 return (  
 <div>  
 <h1>现在有机枪{this.**props**.**num**}把</h1>  
 <button onClick={()=>{this.**props**.**addGUN**()}}>加机关枪</button>  
 <button onClick={()=>{this.**props**.**removeGUN**()}}>上交关枪</button>  
 <button onClick={()=>{this.**props**.**addGUNasync**()}}>拖两天再交</button>  
 </div>  
 )  
 }  
};  
  
// 下面的2个方法相当于不使用react-redux时，从父组件传递进来  
  
// 需要传递进组件的props的数据，即你需要state什么属性放到props  
const mapStatetoProps =(state)=>{  
 // 返回的数据是一个对象  
 return {  
 **num**:state //相当于把num这个数据传递给App组件的props  
 }  
}  
  
// 组件需要的行为，这些行为也会传递进组件的props属性中  
// 即你需要什么方法放到props中，自动dispatch  
const actionCreators={addGUN,addGUNasync,removeGUN};  
  
App=**connect**(mapStatetoProps,actionCreators)(App);  
  
export default App;

5.例子

创建reducer

// 注册，登录，完善信息成功  
const AUTH\_SUCCESS='AUTH\_SUCCESS';

// 初始状态  
var initState={  
 **username**:'',  
 // 注册或者登录失败的信息  
 **msg**:'',  
 **type**:'',  
 // 注册或者登录成功后需要跳转到的路由  
 **redirectTo**:''  
}

export function user(state=initState,action){  
 switch(action.**type**){  
 case AUTH\_SUCCESS:  
 return {...state,**isAuth**:true,**msg**:'',...action.**data**,**redirectTo**:getRedirectPath(action.**data**)}  
 case LOAD\_DATA:  
 // 返回用户信息  
 return {...state,**isAuth**:true,**msg**:'',...action.**data**,**redirectTo**:getRedirectPath(action.**data**)}  
 case RESET:  
 return {**isAuth**:false,**username**:'',**msg**:'',**type**:'',**redirectTo**:''}  
 case ERR\_MSG:  
 // 返回注册失败的状态  
 return {...state,**isAuth**:false,**msg**:action.**msg**,**username**:'',**type**:''}  
 default:  
 return state;  
 }  
}

// 登录,异步  
export function login({username,password}){  
 if(!username||!password){  
 return errMsg('请输入用户名或者密码')  
 }  
 return dispatch=>{  
 axios.post('/user/login',{username,password}).then(res=>{  
 if(res.status===200 && res.data.**code**===0){ //登陆成功  
 dispatch(authSuccess(res.data.**data**));  
 }else{ //失败  
 dispatch(errMsg('用户名或者密码错误'))  
 }  
 })  
 }  
}

// 注册，登录，完善信息  
function authSuccess(data){  
 return {  
 **type**:AUTH\_SUCCESS,  
 data  
 }  
}

合并reducer

import {combineReducers} from 'redux';

// 合并所有的reducer  
export default combineReducers({  
 user,  
 chatuser,  
 chat  
})

Index.js

import {**Provider**} from 'react-redux';  
import {createStore,applyMiddleware,compose} from 'redux';  
import thunk from 'redux-thunk';

// 创建store  
const store = createStore(reducers,compose(  
 // 处理异步，需要用到thunk中间件  
 applyMiddleware(thunk),  
 // 开启谷歌浏览器中的redux插件  
window.devToolsExtension?window.devToolsExtension():f=>f  
));  
ReactDom.render(  
 (  
 // react-router4的写法，与react-router2的写法不一样  
 <Provider store={store}>  
 <BrowserRouter>  
 <div>  
 <AuthRoute/>  
 {/\*被Switch包裹的路由只能有一个被渲染，即有多个路由被匹配到，只渲染第一个被命中的\*/}  
 <Switch>  
 <Route path='/login' component={Login} />  
 <Route path='/register' component={Register} />  
 {/\*不写path默认匹配所有路由\*/}  
 <Route component={Dashboard}/>  
 </Switch>  
 </div>  
 </BrowserRouter>  
 </Provider>  
 ),  
 document.getElementById('root')  
);

组件内使用

import {**connect**} from 'react-redux';

class Login extends React.Component{  
 constructor(props){  
 super(props);  
 // 引入高阶组件后不需要this.state，由高阶组件传递进该组件的props  
 // this.state={  
 // username:'',  
 // password:''  
 // }  
 this.register=this.register.bind(this);  
 }  
 register(){  
 // 这里重置redux中的状态是因为可能用户点击登陆后会改面redux中的状态，跳转到注册页面后这些状态不是需要的  
 this.**props**.reset();  
 // 进行路由跳转,所有的路有组件都有this.props.history，普通组件没有  
 this.**props**.**history**.push('/register');  
 }  
 handelLogin(){

//触发action  
 this.**props**.**login**(this.**props**.**state**);  
 }  
}  
const mapStateToProps=(state)=>{  
 return {  
 **user**:state.**user** }  
};  
const actionCreators={login,reset}  
Login=**connect**(mapStateToProps,actionCreators)(Login);  
export default Login