Redux store action reducer(纯函数---执行动作之后的状态)

State.js

//默认数据

export default{

pageTitle:"首页",

infoList:[]

}

Reducer.js

//创建reducer

import { combineReducers } from 'redux'

import defaultState from './state'

//创建reducer函数

function pageTitle(state = defaultState.pageTitle, action) {

switch (action.type) {

case 'SET\_PAGE\_TITLE':

return action.data;

default:

return state;

}

}

function infoList(state = defaultState.infoList, action) {

switch (action.type) {

case 'SET\_INFO\_LIST':

return action.data

default:

return state

}

}

//导出reducer

export default combineReducers({

pageTitle,

infoList

});

Actions.js

//编写action

export function setPageTitle(data) {

return (dispatch, getState) => {

dispatch({ type: 'SET\_PAGE\_TITLE', data: data })

}

}

Index.js

//创建实例store

import { createStore, applyMiddleware } from 'redux'

// 引入reducer

import reducers from './reducer'

// 中间件，作用：如果不使用该中间件，当我们dispatch一个action时，需要给dispatch函数传入action对象；但如果我们使用了这个中间件，那么就可以传入一个函数，这个函数接收两个参数:dispatch和getState。这个dispatch可以在将来的异步请求完成后使用，对于异步action很有用

import thunk from 'redux-thunk'

// 创建store实例

let store = createStore(

reducers,

applyMiddleware(thunk)

)

export default store;

main.js

import React from 'react';

import ReactDOM from 'react-dom';

import './Css/index.css';

import App from './main/App';

import \* as serviceWorker from './serviceWorker';

// Provider是react-redux两个核心工具之一，作用：将store传递到每个项目中的组件中

import { Provider } from 'react-redux'

// 引入创建好的store实例

import store from './store/index.js'

ReactDOM.render(

<Provider store={store}>

<App />

</Provider>

, document.getElementById('root'));

// If you want your app to work offline and load faster, you can change

// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();

组件

import React, { Component } from 'react';

// connect方法的作用：将额外的props传递给组件，并返回新的组件，组件在该过程中不会受到影响

import { connect } from 'react-redux'

// 引入action

import { setPageTitle } from '../store/actions.js'

class Test extends Component {

constructor(props) {

super(props);

this.state = {};

}

componentDidMount() {

let { setPageTitle } = this.props

// 触发setPageTitle action

setPageTitle('新的标题');

console.log(this.props);

}

render() {

// 从props中解构store

let { pageTitle } = this.props

return (

<div>

测试组件

<div>{pageTitle}</div>

</div>

);

}

}

// mapStateToProps：将state映射到组件的props中

const mapStateToProps = (state) => {

return {

pageTitle: state.pageTitle,

}

}

// mapDispatchToProps：将dispatch映射到组件的props中

const mapDispatchToProps = (dispatch, ownProps) => {

return {

setPageTitle(data) {

// 如果不懂这里的逻辑可查看前面对redux-thunk的介绍

dispatch(setPageTitle(data))

// 执行setPageTitle会返回一个函数

// 这正是redux-thunk的所用之处:异步action

// 上行代码相当于

/\*dispatch((dispatch, getState) => {

dispatch({ type: 'SET\_PAGE\_TITLE', data: data })

)\*/

}

}

}

export default connect(mapStateToProps, mapDispatchToProps)(Test)