# 基于react+ redux的todo例子笔记

## 1 环境搭建

使用淘宝NPM镜像定制的 cnpm 命令行工具代替默认的 npm

$ npm install -g cnpm --registry=https://registry.npm.taobao.org

使用 create-react-app 快速构建 React 开发环境

create-react-app 是来自于 Facebook，通过该命令我们无需配置就能快速构建 React 开发环境。

create-react-app 自动创建的项目是基于 Webpack + ES6 。

执行以下命令创建项目：

$ cnpm install -g create-react-app

$ create-react-app my-app

$ cd my-app

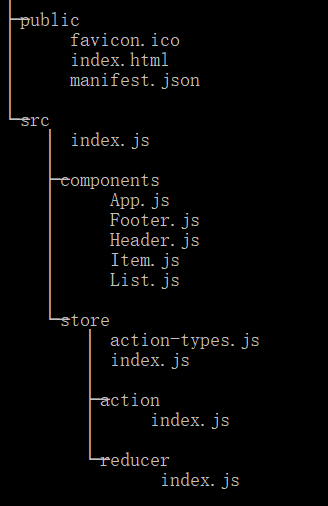
$ npm start

本次需要用到bootstrap redux react-redux；因此也需要安装到本地，执行如下命令：

cnpm install bootstrap redux react-redux --save

## 2结构目录

`win+r ->cmd -> f: ->进入对应的文件夹 -> tree/f 打印目录树如下 `



目录说明：public里面存放的是首页index.html

src里面存放的是入口js文件index.js、components组件文件夹和store，

components包括用到的组件，App.js作为index.js的入口，包括Header、List和Footer。

List组件中的每一项又可以抽取出一个Item组件。如果复用，建议抽取出来，也可以不抽取。

store放的是状态管理，action-type.js中主要是放的action.type。store中的index是导出store。

action中主要是放的是dispatch时候，传递给dispatch的对象。

reducer是一个函数，返回的是state的状态，会作为参数传递到createStore中。它主要有两个参数，一个是state（此时可以设置初始值），另一个是action对象，里面包括对state的操作。最终reducer函数返回一个新对象赋值给state。

## 3 redux

### 3.1 概念

createStore，创建容器store来保存状态，参数为reducer函数。返回{getState,}

getState是获取状态。

dispatch派发，参数为action。内部会调用reducer和订阅的事件。

subscribe 订阅事件，当dispatch执行的时候，会触发。返回的是一个函数，执行为解绑监听的订阅时间。一般放this.setState。把订阅放在componentDidMount.

combineReducers 合并状态。将多个reducer合并成一个。参数{reducer1,reducer2}，合并之后的状态{reducer1:reducer1(),reducer2:reducer2} reducer执行后的结果返回是操作后的状态。

### 3.2 react-redux

定义常量，一般放在action-type.js中

定义reducer，创建store的时候传递进去

定义actions对象，每一个action都是一个函数，这个函数返回的是需要进行的操作。

在redux中调用，例如：store.dispacth(actions.add(1))；

在react-redux中，Provider组件为内置组件，包裹在最外层，并传递属性store={store}，这样被包裹的组件都可以使用store中的状态，但是组件必须使用connect方法。

connect方法处理后的新组件this.props中就可以访问。

新组建=connect(mapStateToProps,mapDispatchToProps)(组件)；

mapStateToProps将状态映射成新组建的属性

let mapStateToProps=(state)=>{

//state 默认是store.getState()后的

return {…state}

}

mapDispatchToProps 将dispatch方法映射成他的属性。可以直接传递actions,，redux会默认调用bindActionCreators，变成dispatch。

mapDispatchToProps=(dispatch)=>{

}

组件中使用this.props.xxx

action-types.js存储action.type的对应类型。

export const ADD\_TODO ='ADD\_TODO'; // 添加

export const CHANGE\_SELECTED="CHANGE\_SELECTED"; //修改选中状态

export const DELETE\_TODO="DELETE\_TODO"; //删除

export const CHANGE\_TYPE="CHANGE\_TYPE"; //切换显示

actions中保存的是对状态的操作传递的参数。每一个action是一个对象，通过函数返回，它中的每一个对象中都包括一个type属性，用于对应reducer中的switch中对应的操作。

//actions

import \* as Types from '../action-types';

let actions ={

addTodo(todo){

return {type:Types.ADD\_TODO,todo}

},

changeSelected(id){

return {type:Types.CHANGE\_SELECTED,id}

},

deleteTodo(id){

return {type:Types.DELETE\_TODO,id}

},

changeType(typeVal){

return {type:Types.CHANGE\_TYPE,typeVal}

}

}

export default actions;

reducer是负责，根据action.type来对数据进行处理，处理的过程要与action对应。

import \* as Types from '../action-types';

let initState={

type: 'all',// all finish unfinish

todoList:[

{isSelected: true,title: '睡觉',id:1},

{isSelected: false,title: '吃饭',id:2}

]

}

function reducer(state=initState,action){

switch (action.type){

case Types.ADD\_TODO:

return {...state,todoList:[...state.todoList,action.todo]};

case Types.CHANGE\_SELECTED:{

let todoList=state.todoList.map(item=>{

if(item.id==action.id){

item.isSelected=!item.isSelected;

}

return item;

})

return {...state,todoList};

}

case Types.DELETE\_TODO:

let todoList=state.todoList.filter(item=>item.id!=action.id);

return {...state,todoList};

case Types.CHANGE\_TYPE:

return {...state,type:action.typeVal};

default:

return state;

}

}

export default reducer;

store仓库

import {createStore} from "redux";

import reducer from './reducer';

let store = createStore(reducer);

window.$store=store;// 暴露到全局，可以在控制台查看数据

export default store;

## 4 实现步骤

首先新建组件App组件为根组件，包括Header、Footer和List组件

### 4.1 App组件

import React, {Component} from 'react';

import Head from './Header';

import Foot from './Footer';

import List from './List';

export default class App extends Component{

constructor(){

super();

}

render(){

return (

<div>

<Head/>

<List/>

<Foot/>

</div>

);

}

}

### 4.2 Header组件

import React, {Component} from 'react';

import {connect} from 'react-redux';

import actions from '../store/action';

class Header extends Component{

getUnFinish=()=>{

return this.props.todoList.filter(item=>!item.isSelected).length;

}

handleKeyUp=(e)=>{

let {addTodo}=this.props;

if(e.keyCode==13){

addTodo({id:Math.random(),isSelected:false,title:e.target.value});

e.target.value='';

}

}

render(){

let {todoList}=this.props;

return (

<div>

<h1>还有{this.getUnFinish()}件事未完成</h1>

<input type="text" className="form-control" onKeyUp={this.handleKeyUp}/>

</div>

);

}

}

export default connect((state)=>({...state}),actions)(Header);

connect是react-redux中的方法，它主要是把组件映射为新组件的props中。第一个参数中的state对应的是store.getState()中的state,第二个参数是actions对象，默认调用了自己内置的方法bindActionCreators，connect后面第二个括号是要添加prop的react组件，第一个括号中的参数是用来改变该组件prop的方法，第一个括号有两个参数，第一个参数是一个函数，返回一个对象，对象的键是该组件的prop属性，值是该prop的值；第二个参数也是一个函数，返回一个对象，对象的键同样是prop的属性名，值是一个redux的dispatch，当这个prop属性被用于触发时，dispatch会改变redux中state的值。

### 4.3 Footer

import React, {Component} from 'react';

import {connect} from "react-redux";

import actions from '../store/action';

class Footer extends Component{

render(){

return (

<div>

<nav className="nav nav-pills" onClick={(e)=>{

// console.log(e.target.dataset);

let typeVal=e.target.dataset.type;

this.props.changeType(typeVal);

}}>

<button className={this.props.type=="all"?"btn-primary":''} data-type="all">全部</button>

<button data-type="finish" className={this.props.type=="finish"?"btn-primary":''} >已完成</button>

<button data-type="unfinish" className={this.props.type=="unfinish"?"btn-primary":''} >未完成</button>

</nav>

</div>

);

}

}

export default connect((state)=>({...state}),actions)(Footer);

### 4.4 List组件

import React, {Component} from 'react';

import {connect} from 'react-redux';

import actions from '../store/action';

import Item from './Item';

class List extends Component{

filterTodo(){

let todoList=[];

if(this.props.type=="all"){

todoList=this.props.todoList;

}else if(this.props.type=="finish"){

todoList= this.props.todoList.filter((item)=>item.isSelected);

}else if(this.props.type=="unfinish"){

todoList=this.props.todoList.filter((item)=>!item.isSelected);

}

return todoList;

}

render(){

let {todoList,changeSelected,deleteTodo}=this.props;

return (

<div>

<ul className="list-group">

{

this.filterTodo().map((item,index)=>{

return <Item item={item} key={index} changeSelected={()=>{changeSelected(item.id);}} deleteTodo={()=>{deleteTodo(item.id);}}/>

})

}

</ul>

</div>

);

}

}

export default connect((state)=>({...state}),actions)(List);

### 4.5 Item组件

import React from "react";

class Item extends React.Component{

render(){

let {item} = this.props;

return (

<li className="list-group-item">

<input type="checkbox" checked={item.isSelected} onChange={()=>{

this.props.changeSelected();

}}/>

{item.title}

<button className="btn btn-xs pull-right" onClick={()=>{

this.props.deleteTodo();

}}>&times;</button>

</li>

);

}

}

export default Item;

### 4.6 根组件

import React from 'react';

import ReactDOM,{render} from 'react-dom';

import App from './components/App';

import {Provider} from 'react-redux';

import store from './store';

import 'bootstrap/dist/css/bootstrap.css';

render(

<Provider store={store}>

<App/>

</Provider>,

document.getElementById('root')

);