react 项目安装插件：

“dependencies”: {
“react”: “^16.6.3”,
“react-dom”: “^16.6.3”,
“react-scripts”: “2.1.1”
},
“devDependencies”: {
“antd”: “^3.10.9”,//ui组件库
“jsonp”: “^0.2.1”,
“react-redux”: “^5.1.1”,
“redux”: “^4.0.1”
}

react项目结构

一，按照类型

component（展示组件）加载初始页面要显示的组件

- Todo组件 设置组件结构 触发action里的函数让其派发指令， 通过this.props.的方式获取数据

```
import React,{Component} from "react";
import {connect} from "react-redux";
import actions from "../store/actions/todo";
class Todo extends Component{
    render(){
        return <div>
            <input type="text" onKeyDown={(e)=>{
                if(e.keyCode===13){
     this.props.todo(e.target.value);
                    e.target.value="";
                }}}/>
            <ul>
           {this.props.todos.map((item,index)=>{
                        return <li key={index}>{item}</li>
                    })}
            </ul>
        </div>
    }
}
export default connect((state)=>{
```

```
    return {todos:state.todo}
},actions)(Todo);
```

container（容器组件）

store文件下是redux状态管理当中有，action，reducer

- actions文件夹 ：组件触发这里的方法执行派发指令（告诉reducer我要改数据）
  - todo.js 设置Todo组件中用到的action

```
import * as actionTypes from "../action-types";
let actions = {
    todo(text){
        return {type:actionTypes.ADDTODO,text:text}
    }
}
export default actions;
```

- reducer文件夹 ：把所有组件用到的reducer整合在一起
  - todo.js 是Todo组件的reducer设置组件需要的初始数据，根据action类型判断哪个组件要改数据及改哪一条数据，获取旧值合并新值返回合并后的值

```
import * as actionTypes from "../action-types";
function todo(state=[],action){
    switch (action.type){
        case actionTypes.ADDTODO:
            return [...state,action.text]
    }
    return [...state];
}
export default todo;
```

- index.js 导出store 到根index.js

```
//导出store
import {createStore,combineReducers} from "redux";
import counter from "./reducer/counter";
import todo from "./reducer/todo";
let reducer = combineReducers({    //{counter:{number:0},todo:[]}
    counter,
    todo
});
export default createStore(reducer);
```

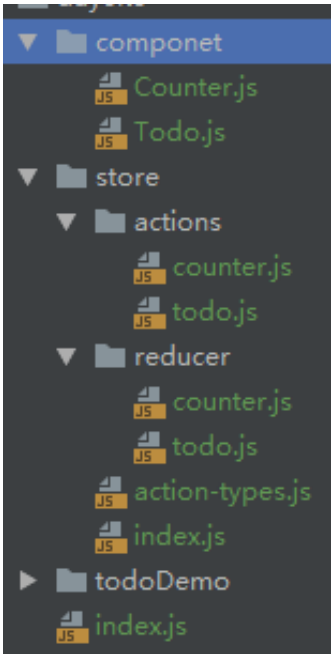
action-types.js : 宏 定义指令类型

```
export const ADD = "ADD";
export const MINUS = "MINUS";
export const ADDTODO ="ADDTODO";
//1.设置派发指令的类型
//2.设置派发的指令（action {type:"ADD",text:1}）
//3.reducer ，获取更改后的数据（是我们要的状态的数据）
//4.各个组件里获取数据，进行的操作
```

index.js入口文件（设置渲染）

引入react/react-dom/需要渲染的组件/store文件redux状态管理

```
import React,{Component} from "react";
import ReactDOM from "react-dom";
import Counter from "./componet/Counter";
import  Todo from "./componet/Todo";
//引入store
import store from "./store/index";//{getState,dispatch,subscribe}
import {Provider} from "react-redux";
ReactDOM.render(
    <Provider store={store}>
        <div>
            <Counter/>
            <Todo/>
        </div>
    </Provider>
,window.root);
```



react-redux 整理

三大原则

1.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。

2.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。

3.使用纯函数来执行修改

为了描述 action 如何改变 state tree ，需要编写 reducers。
Reducer 只是一些纯函数，它接收先前的 state 和 action，并返回新的 state

Action

Action 是把数据从应用传到 store 的有效载荷。它是 store 数据的唯一来源
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。

- Action创建函数 (ADD_TODO是指令类型)

```
export function addTodo(text) {
return { type: ADD_TODO, text }
}
```

Reducer

Reducers 指定应用状态的变化通过指令类型响应 actions 并发送到 store 的，actions 只是描述有事情发生，没有描述应用如何更新 state。

```
function todoApp(state = initialState, action) {
  switch (action.type) {
    case SET_VISIBILITY_FILTER:
      return Object.assign({}, state, {
        visibilityFilter: action.filter
      })
    case ADD_TODO:
      return Object.assign({}, state, {
```

```
          todos: todos(state.todos, action)
        })
      case TOGGLE_TODO:
        return Object.assign({}, state, {
          todos: todos(state.todos, action)
        })
      default:
        return state
    }
  }
```

Store

使用 action 描述“指令类型，不同指令渲染不同组件”，使用 reducers 根据 action 指令类型更新 state

- Store 是把action、reducers 联系到一起的对象。Store 有以下职责：
  - 维持应用的 state；
  - 提供 getState() 方法获取 state；
  - 提供 dispatch(action) 方法更新 state；
  - 通过 subscribe(listener) 注册监听器;
  - 通过 subscribe(listener) 返回的函数注销监听器。

```
import { createStore } from 'redux'
import todoApp from './reducers'
let store = createStore(todoApp)
```