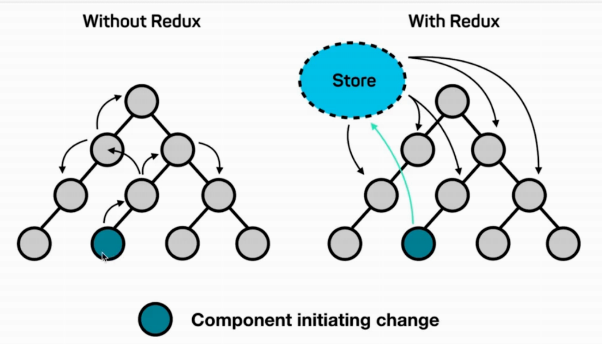
学习笔记

## Redux简单认识:

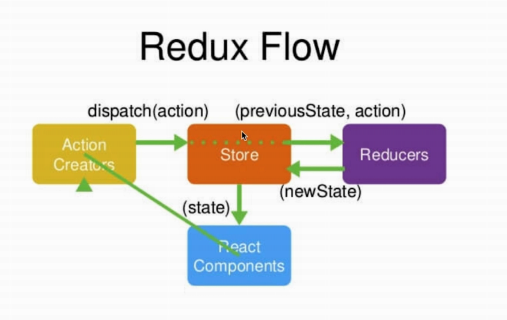


React只是一个视图层的轻量级UI框架；如果需要做一个大型的项目，需要一个数据层的大型框架和他结合起来一起去完成，首选redux。

Redux = Reducer + Flux;

Flux是一开始官方推出的数据管理框架，基本过时了。Redux加入了reducer，是flux的升级版优化版。

## Redux工作流程：



Store：存储数据的公共区域；图书管理员；

React Component：页面组件； 借书的用户；

Reducers：图书记录本，看图书的存放位置；

Action Creators：借书的时候，跟图书管理员说的话；

## Store的创建：

使用redux需要安装该模块：

Cnpm I redux –save

多数情况下，你还需要使用 React 绑定库和开发者工具。

官方文档推荐我们安装附加包：

npm install --save react-redux

npm install --save-dev redux-devtools

然后使用redux里面的createStore创建数据仓库。

import { createStore } from 'redux';

import reducer from './reducer';

*// 创建存储仓库*

const store = createStore(reducer);

export default store;

数据仓库的数据来自于 reduce

const defaultState = {

username:'',

stulist:['A', 'B', 'C']

};

export default (*state* = defaultState, *action*)=>{

return state;

}

获取里面的数据：

import store from './store/index';

store.getState()

## Action和Reducer：

Action执行要执行的动作并触发：

const action = {

type:'del\_stu',

value:this.props.ind

}

store.dispatch(action);

Reduce里面根据动作类型做出响应：

const defaultState = {

username: 'D',

stulist: ['A', 'B', 'C']

};

export default (*state* = defaultState, *action*) => {

console.log(state, action);

let newState={};

switch (action.type) {

case "add\_stu":

*// 不能直接对state进行修改*

newState = JSON.parse(JSON.stringify(state));

newState.username = action.value;

newState.stulist.push(action.value);

return newState;

break;

case "del\_stu":

*// 不能直接对state进行修改*

newState = JSON.parse(JSON.stringify(state));

newState.stulist.splice(action.value, 1);

return newState;

break;

default:

break;

}

return state;

}

组件里面需要订阅store改变动作：

constructor(*props*) {

*super*(props);

this.state = {

stulist:store.getState().stulist,

username:store.getState().username

}

*// 订阅store改变事件*

this.storeChange = this.storeChange.bind(this);

store.subscribe(this.storeChange);

}

storeChange(){

this.setState(store.getState());

}

## ActionTypes的模块化：

模块化的好处是防止出错，同时容易排查错误。

Actiontypes.js:

export const ADD\_STU = 'add\_stu';

export const DEL\_STU = 'del\_stu';

reducer.js:

import {ADD\_STU, DEL\_STU} from './actiontypes';

case ADD\_STU:

*// 不能直接对state进行修改*

newState = JSON.parse(JSON.stringify(state));

newState.username = action.value;

newState.stulist.push(action.value);

return newState;

break;

case DEL\_STU:

*// 不能直接对state进行修改*

newState = JSON.parse(JSON.stringify(state));

newState.stulist.splice(action.value, 1);

return newState;

break;

## actionCreator统一创建action：

在store文件夹下创建actionCreators.js

import {ADD\_STU, DEL\_STU} from './actiontypes';

export const addStuAction = (*value*)=>{

return {

type:ADD\_STU,

value

}

}

export const delStuAction = (*value*)=>{

return {

type:DEL\_STU,

value

}

}

使用：

import {delStuAction} from '../../store/actionCreator';

delStu(){

store.dispatch(delStuAction(this.props.ind));

}

## Redux特征总结：

设计三项基本原则：

1. 单一数据源；
2. State是只读的；
3. 使用纯函数来执行修改；

## Redux核心API：

1. createStore：创建数据仓库；
2. store.dispatch()：用于派发action，执行修改动作；
3. store.getState()：用于获取store里面的数据；
4. store.subscribe()：订阅store的修改，只要store发生改变，回调函数就会被执行；