## 一、安装redux

## 二、Redux 概念图展示工作流程

## 三、结合antd实例演示redux原理：

1. 新建一个TodoList.js文件；
2. 在文件中引入react和antd框架，并定义一个组件TodoList.js并导出；
3. 组件的render中使用antd里面的组件定义一个list列表
4. 在TodoList组件中提前定义好constructor方法；
5. 提前定义一个data变量，变量中放的一些加载列表的数据：

const data = [

'Racing car sprays burning fuel into crowd.',

'Japanese princess to wed commoner.',

'Australian walks 100km after outback crash.',

'Man charged over missing wedding girl.',

'Los Angeles battles huge wildfires.',

];

1. 将这个data设置为List组件的dataSource属性的值（dataSource属性是antd框架中List组件用来设置数据源的属性）：此时列表中会展示出这些data数据

<List

style={{marginLeft:'10px',width:'300px'} header={<div>Header</div>}

footer={<div>Footer</div>} bordered

dataSource={data} //使用data

renderItem={item => (<List.Item>{item}</List.Item>)} />

## 四、创建redux中的store

1. 在src中创建store用于放置redux相关配置脚本
2. 编写一个index.js文件用于创建redux;

2.1，在index.js中引入 createStore ：

import {createStore} from 'redux'

2.2 实例化createStore方法，并赋值给 常量 store:

const store=createStore(reducer) //创建store

2.3、导出常量store：

export default store

3、创建一个redux用于编写reducer数据

3.1在新建的reducer文件中 新建一个对象，对象里面保存的是存储的初始化数据：

const deafultState={ //初始化state

inputValue:'', //这里就是要存储的数据

List:[]

}

3.2、导出一个函数，函数有两个参数，第一个参数：state、第二个参数：action;

export default(state=deafultState,action)=>{ //函数传递两个参数：state、action；并设置state为上面定义的对象

return state //store所有信息

}

4、**访问store存储的数据**；

4.1 ，在TodoList.js文件中引入store

import store from './store/index'

4.2、在TodoList.js中，写一个内置的constructor函数；

constructor(props){

super(props)

}

4.3、在函数内部可以通过store.getState()方法访问到store中reducer里面编写的数据

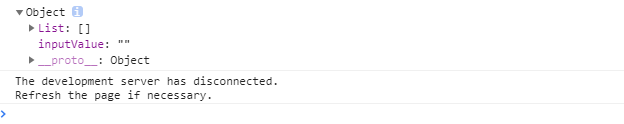
constructor(props){

super(props)

console.log(store.getState())

}

页面打印如下：



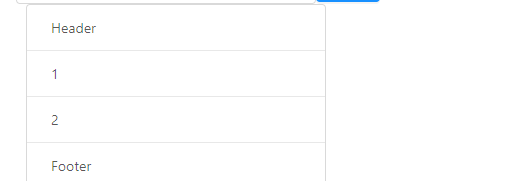
从这里可以看到已经获取到store中reducer的数据

5**、将store中的数据设置为组件内部的数据（在组件中获取store的数据）：**

5.1，在组件内部的constructor中可以通过store.getState()方法获取到store数据，则可以将这个数据设置为组件内state的数据：

dataSource={this.state.List} //使用data

此时页面就会展示出store中的数据：



## 五、使用redux-devTools:

安装chrom插件

## 六、Action和Reducer的编写（修改store数据）

### 1、改变store中的数据（dispatch()）：

1.1：给input绑定onChange事件，在事件函数中定义一个对象，action(也可以较其他名字，按照规范就用action)，里面两条数据：type和value

const action={

type:'change\_input\_value',

value:e.target.value

}

上面的对象是react的规范，一般都要按照这种 规范编写，也就是传递一个type和一个value

**1.2 、执行store.dispatch()方法将新的数据传递到store。** store.dispatch（）方法里面的参数就是上面定义的对象

store.dispatch(action)

**1.3、Store对数据的处理：**

Store收到组件传递的数据后，会将store原本存储的数据和收到的数据传递（转发）到reducer进行查询，这一步是自动执行的

**1.4、Reducer对数据的接受和处理：**

Reducer接收到store传递的数据后可以在reducer.js里面的函数中获取到：

export default(state=deafultState,action)=>{ //函数传递两个参数：state、action；并设置state为上面定义的对象

console.log("state",state)

console.log("action",action)

return state //store所有信息

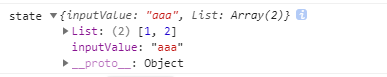
}

在reducsr.js的你函数中，执行console.log方法可以获取到state和action

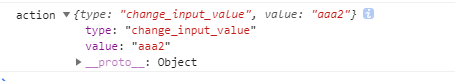
当页面打开的时候，reducer第一次会自动执行里面的console.log() 这第一次打印的东西不用关心。

在input中输入内容的时候，触发了onchange事件，此时reducer会收到store传递过来的数据

在reducer导出的函数中，打印到的数据中 可以看到：



State参数是之前设置的数据



而action是组件通过store传递过来的数据

**处理数据：**

**Reducer 函数里面收到action后，可以对action数据的type进行判断，如果等于组件中传递过来的action里面的type，则对之前存储的数据的修改：**

**修改过程：**

export default(state=deafultState,action)=>{ //函数传递两个参数：state、action；并设置state为上面定义的对象

//console.log("state",state)//

//console.log("action",action)//

if(action.type==='change\_input\_value'){ //判断是否要处理的这条数据

const newState=JSON.parse(JSON.stringify(state)) //对state进行深拷贝，也就是对store存储的数据进行深拷贝

newState.inputValue=action.value //此时对这个newState的值进行修改，由于是深拷贝，所以store存储的数据也会改变

return newState //固定语法：返回一下，必须返回，newState会返回到store,然后替换掉store里面存储的数据

}

return state //store所有信息

}

**这样就完成了对Store数据的修改**

**1.5、组建中数据的更新：**

**1.5.1在组件的constructor函数中，使用store.subscribe()来订阅消息这个消息用于监听store中数据的改变：**

store.subscribe(this.handleStoreChange.bind(this))

**store.subscribe()方法里面的参数是一个函数，相当于是一个事件函数，也需要绑定this指向**

**1.5.2 、改变页面组件中的数据：**

handleStoreChange(){

this.setState(store.getState())

}

**这个方法是在组件的constructor中监听数据变化的，当数据变化之后执行setState()方法，由于组件内部是是使用state获取的数据，二state的值是store.getState()获取到的，所以当执行setState()的时候,组件内的state值变了，组件内使用的值也就变化了**

## 七、ActionTypes的拆分

、 为了避免action 中的type 的值编写的时候写错了，但是不会报错，所以可以将actionType 写在一个单独的文件中，每个type用常量或者变量定义，然后引入到其他要用到的地方，这样，如果写错的话就会，报错，可以定位到错误的位置

## 八、使用ActionCreator 统一创建 action

## 九、UI组件和容器组件

## 十、无状态组件

## 十一、Redux 中获取发送异步请求获取数据

1、

## 十二、中间件

### 什么是中间件

1. Redux-think

## 2、Redux-saga

#### 2．1 安装

#### 2.2引入redux-saga;在store目录下的index.js文件中：

#### 2.2.1、引入 redux中间件和compose函数：

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

#### 2.2.2、定义一个三目运算符的变量：

const composeEnhancers = window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_COMPOSE\_\_ ? window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_COMPOSE\_\_({}) : compose;

#### 2.2.3、引入redux-saga:

import createSagaMiddleware from 'redux-saga'

#### 2.2.4 实例化 redux-saga中间件：

const sagaMiddleware=createSagaMiddleware()

#### 2.2.5、定义一个变量用于注册第三方中间件

const enhancer = composeEnhancers(applyMiddleware(sagaMiddleware) ), //第三方中间件;

#### 2.2.6、将引入实调用的creatrStore方法中：

const store=createStore(

reducer, enhancer

) //创建store

## 3：创建saga.js文件，用于编写saga中间件内容

1、在store 下面的index.js中引入saga.js文件

import saga from './saga'

1. 在文件最底部执行一个方法：

sagaMiddleware.run(saga)

1. 定义好actionTypes 和actionCreator:

actionType:

export const GET\_INIT\_LIST='get\_init\_list'

actionCreator:

import { GET\_INIT\_LIST } from "./ActionTypes";

export const getInitList=()=>{

type:GET\_INIT\_LIST

}

4. 编写 saga文件 ：

4.1，引入actionCreator:并在组件的 componentDidMount()方法中定义action,并使用store.dispatch()方法执行：

componentDidMount(){ //在生命周期函数中执行

const action=getInitList()

store.dispatch(action)

}

3.2、在新建的saga文件中：

引入 tageEvery:

import {takeEvery} from 'redux-saga/effects'

定义generator函数：

function \* mySaga(){

yield takeEvery(,fetchUser)

}

导出这个generator函数