# 上一节课笔记：[react全家桶学习笔记7：路由传参](react全家桶学习笔记7-路由传参.docx)

# 学习笔记8 状态机初始化配置：还是React-fam-bucketdemo项目

状态机主要用来保存公共的数据和公共的方法

## 1.安装，需要安装下面2个插件

## yarn：我们定位到项目根目录，然后执行下面的命令

|  |
| --- |
| yarn add redux react-redux |

## 2.状态机的初始化

### 1）创建状态机的配置文件

在src文件夹下面新建一个store文件夹，然后在文件夹里面新建一个index.js文件，如图

|  |
| --- |
|  |

### 2）创建仓库对象

|  |
| --- |
| **import** {*legacy\_createStore*} **from "redux"**;  **let *store*** = *legacy\_createStore*();  **export default *store*** |

### 3）全局注入仓库对象，需要在main.jsx引入Provider，然后用它把App组件包起来

|  |
| --- |
| **import *React* from 'react' import** ReactDOM **from 'react-dom/client' import** *App* **from './App.jsx'** *// import './index.css'* **import** {*Provider*} **from "react-redux"**;  ReactDOM.*createRoot*(***document***.getElementById(**'root'**)).render(  <**Provider**>  <**App** />  </**Provider**>   ) |

### 4）在main.jsx中引入我们创建的仓库对象，然后把它设置为Provider标签的store属性

|  |
| --- |
| **import *React* from 'react' import** ReactDOM **from 'react-dom/client' import** *App* **from './App.jsx'** *// import './index.css'* **import** {*Provider*} **from "react-redux"**; **import *store* from "./store"**;   ReactDOM.*createRoot*(***document***.getElementById(**'root'**)).render(  <**Provider store=**{***store***}>  <**App** />  </**Provider**>  ) |

#### 注意：redux的底层就是用Context来实现的

### 此时运行程序会报错，因为创建仓库需要reducer函数，下面我们来实现reducer并且设置到store

## 3.配置状态机

### 以角色数据为例，老师的接口

|  |
| --- |
|  |

### 1）在store文件夹下面新建一个roles文件夹，然后新建一个叫做reducers.js的文件，内容如下

#### store/roles/reducers.js

|  |
| --- |
| **export let** *rolesReducer* = (state=[])=>{  **return** state *//返回的数据会保存到仓库* } |

### 2)需要把reducer传递给仓库，注意：我们可能需要使用到很多仓库，所以我们需要使用redux提供的combineReducers方法

#### store/index.js

|  |
| --- |
| **import** {*legacy\_createStore*,*combineReducers*} **from "redux"**; **import** {*rolesReducer*} **from "./roles/reducers"**;   *//合并所有模块的reducer函数* **let** allReducer = *combineReducers*({  roles:*rolesReducer* }); **let *store*** = *legacy\_createStore*(allReducer);  **export default *store*** |

## 4.组件获取状态机数据

### 我们来做角色管理模块，打开src/pages/index/roles/RolesPage.jsx,这里用到一个新的hook，是react-redux提供的useSelector，然后他会拿到整个仓库，我们根据需要筛选我们的数据

|  |
| --- |
| **import *React* from 'react'**; **import** {***useSelector***} **from "react-redux"**;  **const** *RolesPage* = () => {  *//从仓库里面获取数据* **let** rolesData = ***useSelector***((state)=>{  **return** state.roles  })  **return** (  <**div**>  role page  </**div**>  ); };  **export default** *RolesPage*; |

#### 因为此时仓库里面没有数据，我们得到的是他空数组

## 5.状态机的异步配置

### redux状态机默认不支持异步请求，我们需要安装一个中间件来解决这个问题，其实有两个可以选择一个是redux-thunk（比较简单），另外一个是redux-saga(这个很复杂)

### 1.安装

### yarn

|  |
| --- |
| yarn add redux-thunk |

#### 注意如果你的项目是用npm创建的，你需要使用npm来安装依赖

### 2.配置中间件

需要在store/index.js中配置，我们打开index.js

|  |
| --- |
| **import** {*legacy\_createStore*,*combineReducers*,*applyMiddleware*} **from "redux"**; **import** {*rolesReducer*} **from "./roles/reducers"**; *//引入redux-thunk中间件* **import** {***thunk*** }**from 'redux-thunk'** *//合并所有模块的reducer函数* **let** allReducer = *combineReducers*({  roles:*rolesReducer* });  *//应用中间件* **let** middleware = *applyMiddleware*(***thunk***); **let *store*** = *legacy\_createStore*(allReducer,middleware); **export default *store*** |

### 3.配置异步代码

#### 1）创建配置文件，在store/roles文件夹下面新建一个action.js

|  |
| --- |
|  |

#### 2）配置异步代码,store/roles/action.js

|  |
| --- |
| **import** {*getRolesApi*} **from "../../api/roles"**;  **export let** *getRolesAsync* = () =>{  **return async**()=>{ *//返回一个函数  //调用api，发送异步请求* **let** res = **await** *getRolesApi*()  ***console***.log(res)  } } |

#### 3)组件中调用异步方法，RolesPage.jsx,需要使用一个叫做useDispatch的hook，这个hook的功能被thunk增强了，可以在里面直接调用函数

|  |
| --- |
| **import *React***, {*useEffect*} **from 'react'**; **import** {***useDispatch***, ***useSelector***} **from "react-redux"**; **import** {*getRolesAsync*} **from "../../../store/roles/actions"**;  **const** *RolesPage* = () => {  *//接收一个dispatch函数，用来调用仓库的公共方法* **let** dispatch = ***useDispatch***();  *//从仓库里面获取数据* **let** rolesData = ***useSelector***((state)=>{  **return** state.roles  })  **let** getRoles = ()=>{  *//调用仓库公共方法* dispatch(*getRolesAsync*())  }   *useEffect*(()=>{  getRoles()  },[])  **return** (  <**div**>  role page  </**div**>  ); };  **export default** *RolesPage*; |

### 效果：

|  |
| --- |
|  |

## 6.完善异步方法，把数据保存到仓库

### 1）在异步方法中用dispatch来触发reducer方法，并且把手机传递给reducer

#### action.js

|  |
| --- |
| **import** {*getRolesApi*} **from "../../api/roles"**;  **export let** *getRolesAsync* = () =>{  **return async**(dispath)=>{ *//返回一个函数  //调用api，发送异步请求* **let** res = **await** *getRolesApi*() *//我的接口返回的数据结构和老师的不一样，我的少一层data  //保存数据* **if**(res){  ***console***.log(res)  *//调用仓库的reducer方法把新数据保存到仓库* dispath({**type**:**'SET\_ROLES'**, **payload**:res})  }  } } |

### 2.)在异步方法中判断需要进行的操作，并将数据保存到仓库中，其实reducer方法还有第二个参数action，用来接收dispatch传递的数据并且保存到仓库

#### store/roles/reducers.js

|  |
| --- |
| **export let** *rolesReducer* = (state=[],action)=>{  ***console***.log(**"action:"**,action)  **switch** (action.**type**) {  **case "SET\_ROLES"**:  **return** action.**payload  default**:  **return** state  }  } |

### 拿到数据后，就可以进行列表渲染了