基础知识

Redux (opens new window)是 JavaScript 状态管理容器，Redux 和 React 之间没有关系。Redux 支持 React、Angular、Ember、jQuery 甚至纯 JavaScript。
在REACT中使用REACT-REDUX (opens new window)库可以更方便的操作REDUX，也是企业REACT开发中使用广泛的REDUX管理库。

#扩展安装

安装redux所需要的扩展包

```
npm i redux react-redux redux-thunk

```

#目录结构

良好的文件结构可以便于业务功能的扩展

```
redux
 - actions/types.js
 - actions/user.js
 - reducers/index.js
 - reducers/user.js
- store.js
- index.js
- App.js

```

#实例操作

下面通过实例讲解REACT-REDUX的使用

1. actions/types.js 放置 action_type的动作名

```
export default {
 //用户管理
 USER_CREATE:'USER_CREATE',
 USER_UPDATE:'USER_UPDATE'
}

```

2. reducers/user 用于状态处理动作

```
import types from './types'
//初始值
const initState=[]

export function user(state=initState,action){
 switch(action.type){
                case types.USER_ADD:
                        return '用户添加';
                case types.USER_UPDATE:
                        return '用户更新';
                default:
                        return state;
 }
}

```

3. reducers/index.js 用于合并reduces

```
//用于合并多个reducer
import {combineReducers} from 'redux'
import user from './user.js'

export default combineReducers({
        user
})

```

4. actions/user.js 用于调用reducers处理动作，简化组件中的实现代码

```
import types from './types'
...
export const add =(user)=>({
 type:types.USER_ADD,
 data:user
});

```

5. store.js 用于创建store对象

```
import { createStore, applyMiddleware } from "redux"
import thunk from "redux-thunk"
import reducers from "./reducers"

export default createStore(reducers, applyMiddleware(thunk))

```

6. index.js入口文件中传递store

```
import React from "react"
import ReactDOM from "react-dom"
import App from "./App"
import store from "./store"
import { Provider } from "react-redux"

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById("root")
)
```

```
```

7. 组件中获取状态

```
import {add} from '../actions/user'
...
//映射STORE-STATE到PROPS
const mapStateToProps = state => {
  return {
    users: state.users
  }
}

//参数一: 把状态映射到PROPS
//参数二: 把状态调度映射到PROPS,即执行this.props.add()会将user.action动作传递
export default connect(mapStateToProps, { add })(Cart)

```

8. App.js组件监听状态改变后通知REDUX

```
...
componentDidMount(){
 this.props.subscribe(this.getState)
}

```

#异步操作

使用react-thunk (opens new window)中间件来进行异步操作非常方便，首先声明中间件来开启异步操作
store.js 用于创建store对象

```
import { createStore, applyMiddleware } from "redux"
import thunk from "redux-thunk"
import reducers from "./reducers"

```

```
export default createStore(reducers, applyMiddleware(thunk))

```

修改 actions/user.js 来实现异步获取用户

```
export const list=id=>dispatch=>{
        axios.get(id).then((response)=>{
                dispatch(response.data)
 })
}
```

REDUX核心

下面来实现REDUX的原理实现

```
<body>
  <button onclick="store.dispatch({type:'SUB',value:2})">-</button>
  <span id="count">10</span>
  <button onclick="store.dispatch({type:'DESC',value:2})">+</button>
</body>
<script>
  const span = document.querySelector("#count")
  const state = {
    count: 5
  }
  const changeState = (state, action) => {
    switch (action.type) {
      case "DESC":
        return {
          ...state,
          count: state.count + action.value
        }
        break
      case "SUB":
        return {
          ...state,
          count: state.count - action.value
        }
      default:
        state
    }
  }
  const createStore = (state, changeState) => {
    const listeners = []
    return {
      getState: () => state,
      dispatch: action => {
        state = changeState(state, action)
        listeners.map(listener => listener())
      },
      subscribe: listener => listeners.push(listener)
    }
  }
  const store = createStore(state, changeState)
  const render = () => {
    span.innerHTML = store.getState().count
  }
  render()
  store.subscribe(render)
</script>
```