**redux个人笔记（最终版，无过程）**

是什么？ 是一个状态管理器

作用：集中性管理多个组件的共享状态

原则： 单一数据源（多个组件公用一个状态源）、数据只读（只有getState、设置要通过dispatch间接完成）、使用纯函数更新数据（同样的输入同样的输出、dispatch和dispatch里的reducers）

|  |
| --- |
|  |

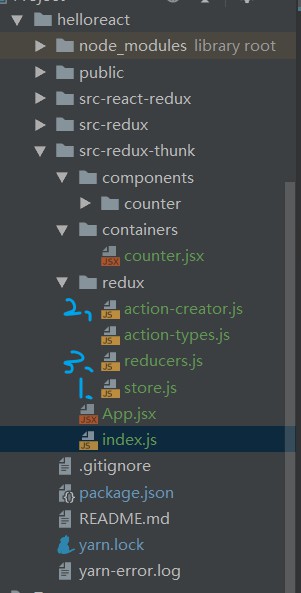
|  |
| --- |
|  |

|  |
| --- |
|  |

多个组件使用同一个redux

|  |
| --- |
|  |

1、上脚手架



|  |
| --- |
| **index.js**   1. /\*\* 2. \* react-redux 3. \*   1.合并创建action和dispatch的代码 4. \*   2.本文件Provider 代替 store.subscribe(render) 5. \*    自动检测状态变化，重新渲染 6. \* / 8. import React from 'react'; 9. import ReactDOM from 'react-dom'; 10. import { Provider } from 'react-redux' 12. import App from './App'; 13. import store from './redux/store' 15. ReactDOM.render(<Provider store={store}><App /></Provider>, document.getElementById('root')); |

|  |
| --- |
| App.jsx   1. import React,{Component} from 'react'; 3. import Counter from './containers/counter'  // 使用react-redux的容器组件 5. export default class App extends Component{ 6. render() { 7. return <Counter/> 8. } 9. } |

|  |
| --- |
| 普通组件Counter   1. import React,{ Component, Fragment } from 'react'; 2. import PropTypes from 'prop-types' 4. /\*\* 5. \* 使用了react-redux的容器组件后 6. \*  1. 状态num在props上 7. \*  2. 创建action和dispatch合并在一起，合并后的方法也在props上 8. \*  （合并后直接使用 ：this.props.increment(this.state.value);） 9. \* / 11. export default class Counter extends Component{ 12. static propTypes = { 13. num: PropTypes.number.isRequired,  // 接受状态数据 14. increment: PropTypes.func.isRequired,  // 接受action方法 15. decrement: PropTypes.func.isRequired, 16. }; 18. state = { 19. value: 1 20. }; 22. handleChange = (e) => { 23. this.setState({ 24. value: +e.target.value 25. }); 26. }; 28. increment = () => { 29. this.props.increment(this.state.value); 30. }; 32. incrementAsync = () => { 33. this.props.incrementAsync(this.state.value); 34. }; 36. render() { 37. const {num} = this.props; // 在props拿状态num 39. return <Fragment> 40. <h2>click {num} times</h2> 41. <select onChange={this.handleChange}> 42. <option value="1">1</option> 43. <option value="2">2</option> 44. <option value="3">3</option> 45. </select> 46. <button onClick={this.increment}>+</button> 47. <button onClick={this.incrementAsync}>increment async</button> 48. </Fragment>; 49. } 50. } |

|  |
| --- |
| Store.js   1. /\*\* 2. \* 创建store对象 3. \* const store = createStore(reducers, composeWithDevTools(applyMiddleware(thunk))); 4. \* 参数：功能实现reducers、异步thunk 5. \* / 7. import { createStore, applyMiddleware } from 'redux' 8. import { composeWithDevTools } from 'redux-devtools-extension' 10. import reducers from './reducers' 11. import thunk from 'redux-thunk' 13. /\*\* 14. \* redux-thunk 15. \*   1.执行异步代码 16. \*      thunk放在createStore第二个参数，用应用中间件applyMiddleware包裹 17. \* 18. \* redux-devtools-extension 19. \*   1.异步代码检测（上线需要删除改组件） 20. \* / 22. const store = createStore(reducers, composeWithDevTools(applyMiddleware(thunk))); 23. export default store; |

|  |
| --- |
| **action-creator：（创建action）(间接修改数据、异步修改数据放这)**   1. import { INCREMENT, DECREMENT, ERROR } from './action-types' 2. // 根据state的功能来定义action，目前只有加减的操作 4. // 同步 返回action对象 5. export const increment = (value) => ({ type:INCREMENT, data:value }); 6. export const decrement = (value) => ({ type:DECREMENT, data:value }); 7. export const error = (value) => ({ type:ERROR, data:value }); 9. // 异步 返回函数 10. export const incrementAsync = (value) => { 11. return (dispatch) => { 12. setTimeout(() => { 13. // 模拟发送请求 14. // 成功 15. dispatch(increment(value)); 16. // 失败 17. // dispatch(error('请求失败')) 18. },1000) 19. } 20. }; |

|  |
| --- |
| **reducers.js（返回newState）（直接修改数据放这）**   1. /\*\* 2. \* 根据action.type实现state的具体操作功能 3. \*   不存在的action.type返回匹配default 4. \* / 5. import {INCREMENT, DECREMENT, ERROR} from './action-types'; // 引入action-types用于判断action.type 6. import { combineReducers } from 'redux'; 7. function num(prevState = 0, action) { 8. switch (action.type) {  // 判断action的类型，(action的类型是引action-types的, 这里也引) 9. case INCREMENT : 10. return prevState + action.data; 11. case DECREMENT : 12. return prevState - action.data; 13. case ERROR: 14. console.log(action.data); 15. return prevState; 16. default : 17. return prevState; 18. } 19. } 21. function add(prevState = [], action){} 22. export default num; // 暴露num这个reducer函数 23. // 暴露多个 (当组件仅需要获取某个状态的时候 使用 结构赋值获取，const {num} = this.props.getState()) 24. // export default combineReducers（{ // 暴露的还是reducer函数 25. // num； 26. // add; 27. // yyy 28. // }） |

|  |
| --- |
| 容器组件（将React与redux关联起来）（多个UI组件共用state就创建多个容器组件，传对应的普通(UI)组件）   1. import { connect } from 'react-redux'; 3. import Counter from '../components/counter'; 4. import { increment, decrement, incrementAsync, error } from '../redux/action-creator' // 导入各个action工厂函数 6. /\*\* 7. \* react-redux 8. \*   1.高阶组件connect封装action和dispatch到一起，传到Counter普通组件props中使用 9. \* / 10. export default connect( 11. (state) => ({num: state }),  // 暴露一个    // 暴露多个  // (state) => ({num: state.num, add: state.add}),   1. { increment, decrement, incrementAsync, error } 2. )(Counter)              // 注意：这里不是传组件 |