

# React全家桶

---

## 资源

---

1. [redux](#)
2. [react-redux](#)
3. [react-router](#)

## 起步

---

### 安装redux

```
npm install redux --save
```

### redux上手

创建store, src/store.js

```
import {createStore} from 'redux'

const counterReducer = (state = 0, action) => {
  switch (action.type) {
    case 'add':
      return state + 1
    case 'minus':
      return state - 1
    default:
      return state
  }
}

const store = createStore(counterReducer)

export default store
```

创建ReduxTest.js, components/ReduxTest.js

```
import React, { Component } from "react";
import store from "../store";

export default class ReduxTest extends Component {
  render() {
    return (
```

```

    <div>
      <p>{store.getState()}</p>
      <div>
        <button onClick={() => store.dispatch({ type: "add" })}>+</button>
        <button onClick={() => store.dispatch({ type: "minus" })}>-</button>
      </div>
    </div>
  );
}
}

```

订阅状态变更, index.js

```

import store from './store'
const render = ()=>{

  ReactDOM.render(
    <App/>,
    document.querySelector('#root')
  )
}
render()

store.subscribe(render)

```

## react-redux

将redux整合到react中, 需要react-redux的支持

```
npm install react-redux --save
```

全局提供store, index.js

```

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.querySelector('#root')
)

```

获取状态数据, ReduxTest.js

```
// import store from "../store";
import { connect } from "react-redux";

@connect(
  state => ({ num: state }), // 状态映射
  {
    add: () => ({ type: "add" }), // action creator
    minus: () => ({ type: "minus" }) // action creator
  }
)
class ReduxTest extends Component {
  render() {
    return (
      <div>
        <p>{this.props.num}</p>
        <div>
          <button onClick={this.props.add}>+</button>
          <button onClick={this.props.minus}>-</button>
        </div>
      </div>
    );
  }
}
export default ReduxTest;
```

## 异步操作

react默认只支持同步，实现异步任务需要中间件的支持

```
npm install redux-thunk redux-logger --save
```

应用中间件, store.js

```
import { createStore, applyMiddleware } from "redux";
import logger from "redux-logger";
import thunk from "redux-thunk";

const store = createStore(fruitReducer, applyMiddleware(logger, thunk));
```

使用异步操作时的变化, ReduxTest.js

```

@connect(
  state => ({ num: state }),
  {
    ...,
    asyncAdd: () => dispatch => {
      setTimeout(() => {
        // 异步结束后, 手动执行dispatch
        dispatch({ type: "add" });
      }, 1000);
    }
  }
)

```

## 代码优化

抽离reducer和action, 创建store/counter.js

```

export const counterReducer = (state = 0, action) => {};

export const add = num => ({ type: "add", payload: num });
export const minus = num => ({ type: "minus", payload: num });
export const asyncAdd = num => dispatch => {};

```

移动action.js并重命名为index.js

```

import { counterReducer } from './counter'
const store = createStore(counterReducer, applyMiddleware(logger, thunk));
export default store;

```

ReduxTest.js

```

import { add, minus, asyncAdd } from "../store/counter";
@connect(
  state => ({ num: state }),
  { add, minus, asyncAdd }
)

```

## 模块化

store/index.js

```
import { combineReducers } from "redux";

const store = createStore(
  combineReducers({counter: counterReducer}),
  applyMiddleware(logger, thunk)
);
```

ReduxTest.js

```
@connect(
  state => ({ num: state.counter }), // 添加一个counter
  { add, minus, asyncAdd }
)
```

## redux原理

### 核心功能实现

store/redux.js

```
export function createStore(reducer, enhancer){
  if (enhancer) {
    return enhancer(createStore)(reducer)
  }
  let currentState = undefined;
  let currentListeners = [];

  function getState(){
    return currentState
  }
  function subscribe(listener){
    currentListeners.push(listener)
  }
  function dispatch(action){
    currentState = reducer(currentState, action)
    currentListeners.forEach(v=>v())
    return action
  }
  dispatch({type: '@IMOOO/KKB-REDUX'})
  return { getState, subscribe, dispatch}
}
```

测试代码, components/MyReduxTest.js

```
import React, { Component } from "react";
import { createStore } from "../store/redux";

const counterReducer = (state = 0, action) => {
  switch (action.type) {
```

```

    case "add":
      return state + 1;
    case "minus":
      return state - 1;
    default:
      return state;
  }
};

const store = createStore(counterReducer);

export default class MyReduxTest extends Component {

  componentDidMount() {
    store.subscribe(() => this.forceUpdate());
  }

  render() {
    return (
      <div>
        {store.getState()}
        <button onClick={() => store.dispatch({ type: "add" })}>+</button>
      </div>
    );
  }
}

```

## 中间件实现

核心任务是实现函数序列执行, redux.js

```

export function applyMiddleware(...middlewares){
  return createStore => (...args) => {
    const store = createStore(...args)
    let dispatch = store.dispatch

    const midApi = {
      getState:store.getState,
      dispatch:(...args)=>dispatch(...args)
    }
    const middlewareChain = middlewares.map(middleware => middleware(midApi))
    dispatch = compose(...middlewareChain)(store.dispatch)
    return {
      ...store,
      dispatch
    }
  }
}

export function compose(...funcs){
  if (funcs.length==0) {
    return arg=>arg
  }
  if (funcs.length==1) {
    return funcs[0]
  }
  return (...args) => {
    let result = funcs[funcs.length-1](...args)
    for (let i = funcs.length-2; i >= 0; i--) {
      result = funcs[i](result, ...args)
    }
    return result
  }
}

```

```

    return funcs[0]
  }
  return funcs.reduce((left,right) => (...args) => right(left(...args)))
}

```

测试代码, MyReduxTest.js

```

import { applyMiddleware } from "../store/redux";

function logger({dispatch, getState}) {
  return dispatch => action => {
    // 中间件任务
    console.log(action.type + '执行了! ! ');
    // 下一个中间件
    return dispatch(action);
  }
}

const store = createStore(counterReducer, applyMiddleware(logger));

```

## redux-thunk原理

```

const thunk = ({dispatch, getState}) => dispatch => action => {
  if (typeof action == 'function') {
    return action(dispatch, getState)
  }
  return dispatch(action)
}
export default thunk

```

测试代码

```

const store = createStore(counterReducer, applyMiddleware(logger,thunk));

<button onClick={() => store.dispatch(() => {
  setTimeout(() => {
    store.dispatch({ type: "add" })
  }, 1000);
})}></button>

```

## react-redux原理

### 实现kreact-redux

核心任务: 属性映射、变更检测和刷新; 实现一个Provider组件可以传递store

```

import React from 'react'

```

```

import PropTypes from 'prop-types'
import {bindActionCreators} from './kkb-redux'

export const connect = (mapStateToProps = state=>state, mapDispatchToProps = {}) =>
(WrapComponent)=>{
  return class ConnectComponent extends React.Component{
    static contextTypes = {
      store: PropTypes.object
    }
    constructor(props, context){
      super(props, context)
      this.state = {
        props:{}
      }
    }
    componentDidMount(){
      const {store} = this.context
      store.subscribe(()=>this.update())
      this.update()
    }
    update(){
      const {store} = this.context
      const stateProps = mapStateToProps(store.getState())
      const dispatchProps = bindActionCreators(mapDispatchToProps,
store.dispatch)
      this.setState({
        props:{
          ...this.state.props,
          ...stateProps,
          ...dispatchProps
        }
      })
    }
    render(){
      return <WrapComponent {...this.state.props}></WrapComponent>
    }
  }
}

export class Provider extends React.Component{
  static childContextTypes = {
    store: PropTypes.object
  }
  getChildContext() {
    return { store: this.store }
  }
  constructor(props, context) {
    super(props, context)
    this.store = props.store
  }
  render() {
    return this.props.children
  }
}

```



## 实现bindActionCreators

```
function bindActionCreator(creator, dispatch){
  return (...args) => dispatch(creator(...args))
}

export function bindActionCreators(creators, dispatch){
  return Object.keys(creators).reduce((ret, item) => {
    ret[item] = bindActionCreator(creators[item], dispatch)
    return ret
  }, {})
}
```

- 安装: `npm install --save react-router-dom`

- 基本使用:

范例：把水果市场转换为多页面， ReduxTest.js

开课吧web全栈架构师

注意render和component有竞争关系，component优先于render，二选一即可，[原因在这里](#)

我们还发现了children选项（不管路由是否匹配都会渲染）

## 路由参数

和vue一样，试用:id的形式定义参数

添加导航链接，FruitList

```
<Link to={`/${detail}/${f}`}>{f}</Link>
```

定义路由，ReduxTest

```
<Route path="/detail/:fruit" component={Detail} />
```

创建Detail并获取参数

```
function Detail({ match, history, location }) {  
  console.log(match, history, location);  
  return (  
    <div>  
      <h3>{match.params.fruit}的详细信息</h3>  
      <p>...</p>  
      <div>  
        <button onClick={history.goBack}>返回</button>  
      </div>  
    </div>  
  );  
}
```

顺带看一下history和location

## 嵌套

Route组件嵌套在其他页面组件中就产生了嵌套关系

去掉根路由的exact，避免Detail不可见

```
<Link to="/list">水果列表</Link>  
  
<Route  
  path="/list"  
  render={props =>  
    loading ? <div>数据加载中...</div> : <FruitList fruits={fruits} />  
  }  
</Route>
```

还可以加个重定向默认跳转

```
<Redirect to="/list"></Redirect>
```

移动Detail相关路由到FruitList

```
<ul>
  {fruits.map(f => (
    <li key={f} onClick={() => setFruit(f)}>
      {/*地址相对于父路由*/}
      <Link to={`/${list/detail}/${f}`}>{f}</Link>
    </li>
  ))}
</ul>
{/*地址相对于父路由*/}
<Route path="/list/detail/:fruit" component={Detail} />
```

## 404页面

设定一个没有path的路由在路由列表最后面，表示一定匹配

```
{/* 添加Switch表示仅匹配一个 */}
<Switch>
  {/* 首页重定向换成Route方式处理避免影响404 */}
  <Route exact path="/" render={props => <Redirect to="/list" />} />
  {/* <Redirect to="/list"></Redirect> */}
  <Route component={() => <h3>页面不存在</h3>}></Route>
</Switch>
```

## 路由守卫

思路：创建高阶组件包装Route使其具有权限判断功能

创建PrivateRoute

```
function PrivateRoute({ component: Component, isLogin, ...rest }) {
  // 结构props为component和rest
  // rest为传递给Route的属性
  return (
    <Route
      {...rest}
      render={
        // 执行登录判断逻辑从而动态生成组件
        props =>
          isLogin ? (
            <Component {...props} />
          ) : (
```

```

        <Redirect
          to={{
            pathname: "/login",
            state: { redirect: props.location.pathname } // 重定向地址
          }}
        />
      )
    }
  />
);
}

```

创建Login

```

function Login({ location, isLogin, login }) {
  const redirect = location.state.redirect || "/"; // 重定向地址

  if (isLogin) return <Redirect to={redirect} />;

  return (
    <div>
      <p>用户登录</p>
      <hr />
      <button onClick={login}>登录</button>
    </div>
  );
}

```

配置路由, ReduxTest

```

<PrivateRoute path="/add" component={FruitAdd} />
<Route path="/login" component={Login} />

```

整合redux, 获取和设置登录态, 创建./store/user.redux.js

```

const initialState = { isLogin: false, loading: false };
export default (state = initialState, action) => {
  switch (action.type) {
    case "requestLogin":
      return { isLogin: false, loading: true };
    case "loginSuccess":
      return { isLogin: true, loading: false };
    case "loginFailure":
      return { isLogin: false, loading: false };
    default:
      return state;
  }
};

export function login(user) {
  return dispatch => {
    dispatch({ type: "requestLogin" });
  };
}

```

```

    setTimeout(() => {
      if (Date.now() % 2 === 0) {
        dispatch({ type: "loginSuccess" });
      } else {
        dispatch({ type: "loginFailure" });
      }
    }, 1000);
  };
}

```

引入, store/index.js

```

import user from "../user.redux";

const store = createStore(
  combineReducers({ fruit: fruitReducer, user }),
  applyMiddleware(logger, thunk)
);

```

连接状态, ReduxTest.js

```

import { login } from "../store/user.redux";

const PrivateRoute = connect(state => ({
  isLogin: state.user.isLogin
}))(function({ component: Component, isLogin, ...rest }) {})

const Login = connect(
  state => ({
    isLogin: state.user.isLogin
  }),
  { login }
)(function({ location, isLogin, login }) {})

```

## 回顾

---