[Action 2](#_Toc27952)

[Reducer 2](#_Toc14499)

[设计 State 结构 2](#_Toc28764)

[Action 处理 2](#_Toc13803)

[处理多个 action 2](#_Toc11839)

[Store 2](#_Toc31917)

# 安装

npm install --save redux

附加包

npm install --save react-redux

npm install --save-dev redux-devtools

# Action

把数据从应用传到 store 的有效载荷。

action 内必须使用一个字符串类型的 type 字段来表示将要执行的动作。应用规模极大时，建议使用单独的模块或文件来存放 action。

Action的type可用单独的文件存放，而后通过常量引用。

尽量减少在 action 中传递的数据。

# Reducer

指定了应用状态的变化如何响应 actions 并发送到 store 。

Reducer为纯函数，相同输入必然有相同输出。

组合多个reducer：combineReducers({ reducer1,reducer2...});

Redux中用户操作后发出action，store自动调用dispatch方法处理action。

## 设计 State 结构

在 Redux 应用中，所有的 state 都被保存在一个单一对象中。

尽可能地把 state 范式化，不存在嵌套。每个数据以 ID 为主键，不同实体或列表间通过 ID 相互引用数据。

## Action 处理

不要修改 state。 使用 Object.assign() 新建一个副本。

Object.assign({}, state, {

visibilityFilter: action.filter //将state，action改变值复制进空对象

});

在 default 情况下返回旧的 state。

## 处理多个 action

同一个函数中封装多个reducer，action传入后根据type判定调用哪个reducer；对state需先克隆后增加或更新，Object.assign({},state,reducer)。增加时，通过...解构state参数获取参数集合；修改时，通过map遍历集合而后修改。

组合多个reducer处理多个action：combineReducers({ visibilityFilter,todos});

# Store

Redux 应用只有一个单一的 store。当需要拆分数据处理逻辑时，你应该使用 reducer 组合 而不是创建多个 store。

创建store：

createStore(reducer(,initState)); //initState参数可选，设置初始state

store方法：

getState();获取当前state

dispatch(action);使用getState()、action调用reducer。如何匹配到对应reducer，注册？Type匹配？逐一传入action调用reducer，而后拼接所有reducer返回结果。

subscribe(listener);listener为监听到变化后调用的函数；返回停止监听的函数。

unSubscrebe=subscribe(listener); unSubscribe();停止监听

# 数据流

严格的单向数据流，父到子。数据范式化，通过ID引用。

数据库范式：1NF-属性不可再分，2NF-消除部分依赖(主键最小化)，3NF-消除传递依赖

数据生命周期：

1，调用store.dispatch(action);

2，逐个执行reducer。

3，组合reducer返回结果。

4，在listenner中更新state。

# 搭配React

通过connect()获取容器组件，可使用Redux的性能优化。