### **单一数据源**

**整个应用的**[state](https://www.redux.org.cn/docs/Glossary.html" \l "state)**被储存在一棵 object tree 中，并且这个 object tree 只存在于唯一一个**[store](https://www.redux.org.cn/docs/Glossary.html" \l "store)**中。**

console.log(store.getState())

/\* 输出{

visibilityFilter: 'SHOW\_ALL',

todos: [

{

text: 'Consider using Redux',

completed: true,

},

{

text: 'Keep all state in a single tree',

completed: false

}

]}\*／

### **State 是只读的**

**唯一改变 state 的方法就是触发**[action](https://www.redux.org.cn/docs/Glossary.html" \l "action)**，action 是一个用于描述已发生事件的普通对象。**

store.dispatch({

type: 'COMPLETE\_TODO',

index: 1})

store.dispatch({

type: 'SET\_VISIBILITY\_FILTER',

filter: 'SHOW\_COMPLETED'})

### **使用纯函数来执行修改**

**想要改变state，你需要编写**[reducers](https://www.redux.org.cn/docs/Glossary.html" \l "reducer)**。**

Reducer 只是一些纯函数（同样的输入参数就有同样的输出结果），它接收先前的 state 和 action，并返回新的 state。

function visibilityFilter(state = 'SHOW\_ALL', action) {

switch (action.type) {

case 'SET\_VISIBILITY\_FILTER':

return action.filter

default:

return state

}

}

function todos(state = [], action) {

switch (action.type) {

case 'ADD\_TODO':

return [

...state,

{

text: action.text,

completed: false

}

]

case 'COMPLETE\_TODO':

return state.map((todo, index) => {

if (index === action.index) {

return Object.assign({}, todo, {

completed: true

})

}

return todo

})

default:

return state

}

}

import { combineReducers, createStore } from 'redux'

let reducer = combineReducers({ visibilityFilter, todos })

let store = createStore(reducer)