**Redux三大原则**

Redux 可以用这三个基本原则来描述：

### 单一数据源

**整个应用的**[**state**](http://cn.redux.js.org/docs/Glossary.html#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**](http://cn.redux.js.org/docs/Glossary.html#store)**中。**

这让同构应用开发变得非常容易。来自服务端的 state 可以在无需编写更多代码的情况下被序列化并注入到客户端中。由于是单一的 state tree ，调试也变得非常容易。在开发中，你可以把应用的 state 保存在本地，从而加快开发速度。此外，受益于单一的 state tree ，以前难以实现的如“撤销/重做”这类功能也变得轻而易举。

console.log(store.getState())

/\* Prints

{

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

**惟一改变 state 的方法就是触发**[**action**](http://cn.redux.js.org/docs/Glossary.html#action)**，action 是一个用于描述已发生事件的普通对象。**

这样确保了视图和网络请求都不能直接修改 state，相反它们只能表达想要修改的意图。因为所有的修改都被集中化处理，且严格按照一个接一个的顺序执行，因此不用担心 race condition 的出现。 Action 就是普通对象而已，因此它们可以被日志打印、序列化、储存、后期调试或测试时回放出来。

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

**为了描述 action 如何改变 state tree ，你需要编写**[**reducers**](http://cn.redux.js.org/docs/Glossary.html#reducer)**。**

Reducer 只是一些纯函数，它接收先前的 state 和 action，并返回新的 state。刚开始你可以只有一个 reducer，随着应用变大，你可以把它拆成多个小的 reducers，分别独立地操作 state tree 的不同部分，因为 reducer 只是函数，你可以控制它们被调用的顺序，传入附加数据，甚至编写可复用的 reducer 来处理一些通用任务，如分页器。

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)

就是这样，现在你应该明白 Redux 是怎么回事了。