****Reducers**** 指定了应用状态的变化如何响应 [actions](https://www.redux.org.cn/docs/basics/Actions.html) 并更新 state。

## **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。

以 todo 应用为例

{

visibilityFilter: 'SHOW\_ALL',

todos: [

{

text: 'Consider using Redux',

completed: true,

},

{

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

completed: false

}

]

}

## **Action 处理**

reducer 就是一个纯函数，接收旧的 state 和 action，返回新的 state。

(previousState, action) => newState

永远不要在 reducer 里做这些操作：

* 修改传入参数；
* 执行有副作用的操作，如 API 请求和路由跳转；
* 调用非纯函数，如 Date.now() 或 Math.random()。

Redux 首次执行时，传入的state 为 undefined，此时我们可借机设置并返回应用的初始 state。

import { VisibilityFilters } from './actions'

const initialState = {

visibilityFilter: VisibilityFilters.SHOW\_ALL,

todos: []

};

// 如果 state 为undefined则默认值为initialState

function todoApp(state = initialState, action) {

switch (action.type) {

case SET\_VISIBILITY\_FILTER:

// 如果更改state，则必须返回一个新的state

return Object.assign({}, state, {

visibilityFilter: action.filter

})

default:

// 如果没有更改state，则返回原对象

return state

}}

## **处理多个 action**

import {

ADD\_TODO,

TOGGLE\_TODO,

SET\_VISIBILITY\_FILTER,

VisibilityFilters} from './actions'

...

function todoApp(state = initialState, action) {

switch (action.type) {

// 如果动作的类型是 SET\_VISIBILITY\_FILTER

case SET\_VISIBILITY\_FILTER:

return Object.assign({}, state, {

visibilityFilter: action.filter

})

// 如果动作的类型是 ADD\_TODO

case ADD\_TODO:

return Object.assign({}, state, {

todos: [

...state.todos,

{

text: action.text,

completed: false

}

]

})

default:

return state

}}

## **拆分 Reducer**

目前的代码看起来有些冗长：

function todoApp(state = initialState, action) {

switch (action.type) {

case SET\_VISIBILITY\_FILTER:

return Object.assign({}, state, {

visibilityFilter: action.filter

})

case ADD\_TODO:

return Object.assign({}, state, {

todos: [

...state.todos,

{

text: action.text,

completed: false

}

]

})

case TOGGLE\_TODO:

return Object.assign({}, state, {

todos: state.todos.map((todo, index) => {

if (index === action.index) {

return Object.assign({}, todo, {

completed: !todo.completed

})

}

return todo

})

})

default:

return state

}}

如下：我们将上面的reducer拆分为下面两个

function todos(state = [], action) {

switch (action.type) {

case ADD\_TODO:

return [

...state,

{

text: action.text,

completed: false

}

]

case TOGGLE\_TODO:

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

if (index === action.index) {

return Object.assign({}, todo, {

completed: !todo.completed

})

}

return todo

})

default:

return state

}}

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

switch (action.type) {

case SET\_VISIBILITY\_FILTER:

return action.filter

default:

return state

}}

function todoApp(state = {}, action) {

return {

visibilityFilter: visibilityFilter(state.visibilityFilter, action),

todos: todos(state.todos, action)

}}

随着应用的膨胀，我们还可以将拆分后的 reducer 放到不同的文件中

最后，Redux 提供了 [combineReducers()](https://www.redux.org.cn/docs/api/combineReducers.html) 工具类来做上面 todoApp 函数做的事情

import { combineReducers } from 'redux'

const todoApp = combineReducers({

visibilityFilter,

Todos

})

export default todoApp

注意上面的写法和下面完全等价：

export default function todoApp(state = {}, action) {

return {

visibilityFilter: visibilityFilter(state.visibilityFilter, action),

todos: todos(state.todos, action)

}}

## **源码**

import { combineReducers } from 'redux'

import {

ADD\_TODO,

TOGGLE\_TODO,

SET\_VISIBILITY\_FILTER,

VisibilityFilters} from './actions'

const { SHOW\_ALL } = VisibilityFilters

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 TOGGLE\_TODO:

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

if (index === action.index) {

return Object.assign({}, todo, {

completed: !todo.completed

})

}

return todo

})

default:

return state

}

}

const todoApp = combineReducers({

visibilityFilter,

todos})

export default todoApp