## 阅读本篇文章注意区分 action 和 action 创建函数

## **Action**

一个异步 API 请求一般使用3个action标识（非必须）

**1 请求开始的 action。**

2 请求成功的 action。

3 请求失败的 action。

为了区分这三种 action，你可以在 action 里添加一个专门的 status 字段作为标记位

{ type: 'FETCH\_POSTS' }

{ type: 'FETCH\_POSTS', status: 'error', error: 'Oops' }

{ type: 'FETCH\_POSTS', status: 'success', response: { ... } }

又或者为它们定义不同的 type：

{ type: 'FETCH\_POSTS\_REQUEST' }

{ type: 'FETCH\_POSTS\_FAILURE', error: 'Oops' }

{ type: 'FETCH\_POSTS\_SUCCESS', response: { ... } }

在本教程中，我们将使用不同的 type 来做。

## **同步 Action 创建函数（Action Creator）**

// 在请求发送前，我们向store发送的action

export const REQUEST\_POSTS = 'REQUEST\_POSTS'

export function requestPosts(subreddit) {

    return {

        type: REQUEST\_POSTS,

        subreddit

    }

}

// 在收到请求响应时，我们向store发送的action

export const RECEIVE\_POSTS = 'RECEIVE\_POSTS'

export function receivePosts(subreddit, json) {

    return {

        type: RECEIVE\_POSTS,

        subreddit,

        posts: json.data.children.map(child => child.data),

        receivedAt: Date.now()

    }

}

## **设计 state 结构**

"Reddit 头条" 应用会长这个样子：

let state = {

    selectedsubreddit: 'frontend',

    postsBySubreddit: {

        // reactjs 子版块

        reactjs: {

            // 是否正在请求

            isFetching: false,

            // 数据是否已经失效

            didInvalidate: false,

            // 最后更新时间

            lastUpdated: 1439478405547,

            // 实际应用可能还需要 fetchedPageCount 和 nextPageUrl 这样分页相关的字段

            // 子版块的文章列表

            items: [

                {

                    id: 42,

                    title: 'Confusion about Flux and Relay'

                },

                {

                    id: 500,

                    title: 'Creating a Simple Application Using React JS and Flux Architecture'

                }

            ]

        }

    }

}

## **Reducer设计**

import { combineReducers } from 'redux'

import {

    SELECT\_SUBREDDIT,

    INVALIDATE\_SUBREDDIT,

    REQUEST\_POSTS,

    RECEIVE\_POSTS

} from '../actions'

...

// posts Reducer

function posts(

    state = {

        isFetching: false,

        didInvalidate: false,

        items: []

    },

    action) {

    switch (action.type) {

        // 文章失效

        case INVALIDATE\_SUBREDDIT:

            return Object.assign({}, state, {

                didInvalidate: true

            })

        // 请求文章

        case REQUEST\_POSTS:

            return Object.assign({}, state, {

                isFetching: true,

                didInvalidate: false

            })

        // 接收文章

        case RECEIVE\_POSTS:

            return Object.assign({}, state, {

                isFetching: false,

                didInvalidate: false,

                items: action.posts,

                lastUpdated: action.receivedAt

            })

        default:

            return state

    }

}

// postsBySubreddit Reducer

function postsBySubreddit(state = {}, action) {

    switch (action.type) {

        case INVALIDATE\_SUBREDDIT:

        // 文章请求时

        case RECEIVE\_POSTS:

        // 文章接收时

        case REQUEST\_POSTS:

            return Object.assign({}, state, {

                posts: posts(state.posts, action)

            })

        default:

            return state

    }

}

// 根 reducer

const rootReducer = combineReducers({

    postsBySubreddit,

    ...

})

export default rootReducer

## **异步 action 创建函数**

以上结构设计完毕，那么我们的执行逻辑如下：

1. 调用 dispatch()，发送一个异步 action
2. 异步 action调用 dispatch()，发送REQUEST\_POSTS action，告知store开始请求文章数据，reducer处理REQUEST\_POSTS
3. 异步 action调用 fetch() 执行异步请求
4. fetch() 接收成功后调用 dispatch()，发送 RECEIVE\_POSTS action，告知store文章数据已接收，reducer处理RECEIVE\_POSTS

以上2，3，4个步骤均包含在我们的“异步 action”中，我们如何实现“异步 action 创建函数”？

第一步：

我们需要使用Thunk middleware中间件

通过使用指定的 Thunk middleware，action 创建函数除了返回 action 对象外还可以返回函数。

当 action 创建函数返回函数时，这个函数会被 Redux Thunk middleware 执行

如下：我们在index.js中引用

// 引用 ThunkMiddleware 中间件

import thunkMiddleware from 'redux-thunk'

import { createLogger } from 'redux-logger'

import { createStore, applyMiddleware } from 'redux'

import { selectSubreddit, fetchPosts } from './actions'

import rootReducer from './reducers'

const loggerMiddleware = createLogger()

const store = createStore(

    rootReducer,

    // 在创建 store 时指定中间件

    applyMiddleware(

        thunkMiddleware, // 这里添加了一个thunk中间件，他会处理 thunk action

        loggerMiddleware // 一个很便捷的 middleware，用来打印 action 日志

    )

)

// dispatch 普通动作

store.dispatch(selectSubreddit('reactjs'))

// dispatch thunk action（异步动作）

// thunk action 没有对应的 Reducer

// thunk action会被thunk中间件执行

store.dispatch(fetchPosts('reactjs'))

    .then(() => console.log(store.getState()))

第二步：

编写异步action创建函数代码

import fetch from 'cross-fetch'

...

// 来看一下我们写的第一个（异步） thunk action 创建函数！

// 注：thunk action 没有 Reducer

export function fetchPosts(subreddit) {

    // 返回 thunk action（这个动作是一个函数），Thunk middleware 会执行这个函数

return function (dispatch) {

        // 发送 REQUEST\_POSTS action

        dispatch(requestPosts(subreddit))

        // 调用 fetch() 执行异步请求，返回一个等待处理的 promise。

        return fetch(`http://www.subreddit.com/r/${subreddit}.json`)

            .then(

                response => response.json(),

                error => console.log('An error occurred.', error)

            )

            .then(json =>

                // fetch() 接收成功后调用 dispatch()，发送 RECEIVE\_POSTS action

                dispatch(receivePosts(subreddit, json))

            )

    }

}

**异步动作（thunk action）参数**

(dispatch,getState) => {

...

}