https://redux-saga-in-chinese.js.org/

（一）自述

（1）

redux-saga是一个redux中间件，管理副作用

（二）初级教程

（1）

yield delay(1000)返回Promise

yield call(delay, 1000)返回plain Object

（2）

export function\* needTest() {

yield call(delay, 1000)

yield put({ type: 'typeName' })

}

import test from 'tape';

test('needTest Saga test', (assert) => {

const gen = needTest()

assert.deepEqual(

gen.next().value,

call(delay, 1000),

'description'

)

assert.deepEqual(

gen.next().value,

put({type: 'typeName'}),

'description'

)

assert.deepEqual(

gen.next(),

{ done: true, value: undefined },

'description'

)

assert.end()

});

（三）Saga 概念的背景

（四）使用 Saga Helpers

（1）takeEvery

yield takeEvery('typeName', otherReduxSagaFunction)

（2）takeLatest

yield takeLatest('typeName', otherReduxSagaFunction)

（五）声明式 Effects

yield call(fn, ...arguments)

yield apply(fn, [...arguments])

yield call([obj, obj.method], ...arguments)

yield apply(obj, obj.method, [...arguments])

注：call和appl作用相同，改变this指向。

function add(c, d){

return this.a + this.b + c + d;

}

var objectWithoutAdd = {a:1, b:2};

add.call(objectWithoutAdd, 3, 4); // 1 + 2 + 3 + 4 = 10

add.apply(objectWithoutAdd, [3, 4]); // 1 + 2 + 3 + 4 = 10

（六）dispatch action

put

（七）错误处理

（1.1）try catch

function\* fetchFunction() {

try {

const fetchData = yield call(fetchApi, 'fetchUrl')

yield put({ type: 'fetchSuccess', fetchData })

}

catch(error) {

yield put({ type: 'fetchFailed', error })

}

}

（1.2）测试

const iterator = fetchFunction()

assert.deepEqual(

iterator.next().value,

call(fetchApi, 'fetchUrl'),

'description'

)

const error = {}

assert.deepEqual(

iterator.throw(error).value,

yield put({ type: 'fetchFailed', error }),

'description'

)

（2）错误标识

function\* fetchFunction() {

const { fetchData, error } = yield call(fetchApi, 'fetchUrl')

if (fetchData){

yield put({ type: 'fetchSuccess', fetchData })

}else{

yield put({ type: 'fetchFailed', error })

}

}

（八）一个常见的抽象概念: Effect

（九）监听未来的 action

const action = yield take('\*')

const state = yield select()

（十）无阻塞调用

const action = yield take([...typeName])

const forkName = yield fork(otherReduxSagaFunction, action)

yield cancel(forkName)

try {

} catch(error) {

} finally {

if(yield cancelled()){

}

}

（十一）同时执行多个任务

const [name1, name2] = yield [

call(otherReduxSagaFunction1, ...arguments1),

call(otherReduxSagaFunction2, ...arguments2))

]

generator会被阻塞直到所有的effect都执行完毕，或一个effect被拒绝

（十二）在多个 Effects 之间启动 race

const {name1, name2} = yield race({

name1:call(otherReduxSagaFunction1, ...arguments1),

name2:call(otherReduxSagaFunction2, ...arguments2))

})

（十三）使用 yield\* 对 Sagas 进行排序

yield\* otherReduxSagaFunction()

（十四）组合 Sagas

（十五）取消任务

取消当前Effect任务，也将同时取消阻塞在当前Effect任务中的任务。

caller调用

callee被调用

（十六）redux-saga 的 fork model

fork

用来创建 attached forks

parent会等待fork终止。

取消parent会自动取消fork

fork抛出的未捕获的错误会冒泡到 parent

spawn

用来创建 detached forks

parent不会等待fork终止。

取消parent不会自动取消fork

fork抛出的未捕获的错误不会冒泡到 parent

（十七）并发

takeEvery实现

import {fork, take} from "redux-saga/effects"

const takeEvery = (pattern, saga, ...args) => fork(function\*() {

while (true) {

const action = yield take(pattern)

yield fork(saga, ...args.concat(action))

}

})

takeLatest实现

import {cancel, fork, take} from "redux-saga/effects"

const takeLatest = (pattern, saga, ...args) => fork(function\*() {

let lastTask

while (true) {

const action = yield take(pattern)

if (lastTask) {

yield cancel(lastTask) // 如果任务已经结束，则 cancel 为空操作

}

lastTask = yield fork(saga, ...args.concat(action))

}

})

（十八）测试 Sagas

（十九）连接 Sagas 至外部输入和输出

（二十）使用 Channels

（1）actionChannel缓存action

const chan = yield actionChannel('typeName')

const action = yield take(chan)

yield call(otherReduxSagaFunction, action)

（2）eventChannel连接外部事件

function eventChannelOuterFunction(argument) {

return eventChannel(emitter => {

//subscriber函数，里面emitter(argument)或emitter(END)

//return subscriber函数

})

}

const chan = yield call(eventChannelOuterFunction, argument)

const action = yield take(chan)

yield call(otherReduxSagaFunction, action)

（3）channel在Sagas之间沟通

const chan = yield call(channel)

const action = yield take(chan)

yield put(chan, action)

yield fork(otherReduxSagaFunction, chan)

（二十一）技巧

（1）节流，毫秒内不会再一次执行

import { throttle } from 'redux-saga/effects'

yield throttle(毫秒, 'typeName', otherReduxSagaFunction)

（2）防抖动

function\* handleInput({ input }) {

yield delay(500)

}

function\* watchInput() {

yield takeLatest('INPUT\_CHANGED', handleInput);

}

（3）ajax 重试

（4）ajax 撤销