link: null
title: 珠峰架构师成长计划
description: src\index.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=160 sentences=195, words=1898

1. redux-saga

- redux-saga (https://redux-saga-in-chinese.js.org/) 是一个 redux 的中间件，而中间件的作用是为 redux 提供额外的功能。
- 在 reducers 中的所有操作都是同步的并且是纯粹的，即 reducer 都是纯函数，纯函数是指一个函数的返回结果只依赖于它的参数，并且在执行过程中不会对外部产生副作用，即给它传什么，就吐出什么。
- 但是在实际的应用开发中，我们希望做一些异步的（如Ajax请求）且不纯粹的操作（如改变外部的状态），这些在函数式编程范式中被称为"副作用"。

> redux-saga 就是用来处理上述副作用（异步任务）的一个中间件。它是一个接收事件，并可能触发新事件的过程管理者，为你的应用管理复杂的流程。

2. redux-saga工作原理

- sages 采用 Generator 函数来 yield Effects（包含指令的文本对象）
- Generator 函数的作用是可以暂停执行，再次执行的时候从上次暂停的地方继续执行
- Effect 是一个简单的对象，该对象包含了一些给 middleware 解释执行的信息。
- 你可以通过使用 effects API 如 fork，call，take，put，cancel 等来创建 Effect。

3. redux-saga分类

- worker saga 做实际的工作，如调用API，进行异步请求，获取异步封装结果
- watcher saga 监听被dispatch的actions,当接受到action或者知道其被触发时，调用worker执行任务
- root saga 立即启动saga的唯一入口

4. 构建项目

```
cnpm install create-react-app -g
create-react-app zhufeng-saga-start
cd zhufeng-saga-start
cnpm i redux react-redux redux-saga tape --save
```

5. 跑通saga

src\index.js

```
import store from './store';
```

src\store\index.js

```
import {createStore, applyMiddleware} from 'redux';
import reducer from './reducer';
import createSagaMiddleware from 'redux-saga';

import {helloSaga} from './sagas';
let sagaMiddleware = createSagaMiddleware();

let store=applyMiddleware(sagaMiddleware)(createStore)(reducer);
sagaMiddleware.run(helloSaga);
export default store;
```

src\store\reducer.js

```
export default function (state,action) {}
```

src\store\sagas.js

```

export function* helloSaga() {
    console.log('Hello Saga!');
}
```

6. 异步计数器

src/components/Counter.js

```
import React,{Component} from 'react'
import {connect} from 'react-redux';
import actions from '../store/actions';
class Counter extends Component{
    render() {
        return (
            <div>
                <p>{this.props.number}p>
                <button onClick={this.props.incrementAsync}>+button>
            div>
        )
    }
}
export default connect(
    state => state,
    actions
)(Counter);
```

src/index.js

```
import React from 'react'
import ReactDOM from 'react-dom';
import Counter from './components/Counter';
import {Provider} from 'react-redux';
import store from './store';
ReactDOM.render(<Provider store={store}>
  <Counter/>
Provider>,document.querySelector('#root'));
```

src/store/action-types.js

```
export const INCREMENT='INCREMENT';
export const INCREMENT_ASYNC='INCREMENT_ASYNC';
```

src/store/actions.js

```
import * as types from './action-types';
export default {
    incrementAsync() {
        return {type:types.INCREMENT_ASYNC}
    }
}
```

src/store/index.js

```
import {createStore, applyMiddleware} from 'redux';
import reducer from './reducer';
import createSagaMiddleware from 'redux-saga';

import rootSaga from './sagas';
let sagaMiddleware = createSagaMiddleware();

let store=applyMiddleware(sagaMiddleware)(createStore)(reducer);
sagaMiddleware.run(rootSaga);
export default store;
```

src/store/reducer.js

```
import * as types from './action-types';
export default function (state={number:0},action) {
    switch(action.type){
        case types.INCREMENT:
            return {number: state.number+1};
        default:
            return state;
    }
}
```

src/store/sagas.js

```
import { delay,all,put, takeEvery } from 'redux-saga/effects'

export function* incrementAsync() {

    yield delay(1000)

    yield put({ type: 'INCREMENT' })
}

export function* watchIncrementAsync() {
    yield takeEvery('INCREMENT_ASYNC', incrementAsync)
}

export function* helloSaga() {
    console.log('Hello Saga!');
}

export default function* rootSaga() {

    yield all([
      helloSaga(),
      watchIncrementAsync()
    ])
  }
```

7. 单元测试

```
cnpm i @babel/core @babel/node @babel/plugin-transform-modules-commonjs --save-dev
```

```
"scripts": {
    "test": "babel-node src/store/sagas.spec.js --plugins @babel/plugin-transform-modules-commonjs"
}
```

src\utils.js

```
export const   delay = (ms)=>{
    return new Promise(function(resolve){
        setTimeout(()=>{
            resolve();
        },ms);
    });
}
```

src\store\sagas.spec.js

```
import test from 'tape';
import { all,put, takeEvery,call } from 'redux-saga/effects';
import { incrementAsync }  from './sagas';
import {delay} from '../utils';

test('incrementAsync Saga test', (assert) => {
    const gen = incrementAsync();
    assert.deepEqual(
    gen.next().value,
    call(delay,3000),
    'incrementAsync should return a Promise that will resolve after 3 second'
  )

   assert.deepEqual(
    gen.next().value,
    put({type: 'INCREMENT'}),
    'incrementAsync Saga must dispatch an INCREMENT action'
  )
  assert.deepEqual(
    gen.next(),
    {done:true,value:undefined},
    'incrementAsync Saga must be done'
  )

  assert.end()
});
```

8. 声明式effects

- 在 redux-saga 的世界里，Sagas 都用 Generator 函数实现。我们从 Generator 里 yield 纯 JavaScript 对象以表达 Saga 逻辑
- 我们称呼那些对象为 Effect。Effect 是一个简单的对象，这个对象包含了一些给 middleware 解释执行的信息
- 你可以把 Effect 看作是发送给 middleware 的指令以执行某些操作（调用某些异步函数，发起一个 action 到 store，等等）
- `cps(fn, ...args)` 创建一个 Effect 描述信息，用来命令 middleware 以 Node 风格的函数（Node style function）的方式调用 fn
- `call(fn, ...args)` 创建一个 Effect 描述信息，用来命令 middleware 以参数 args 调用函数 fn
- `call([context, fn], ...args)` 类似 call(fn, ...args)，但支持传递 this 上下文给 fn,在调用对象方法时很有用
- `apply(context, fn, [args])` call([context, fn], ...args) 的另一种写法

src/store/sagas.js

```
import {all,put, takeEvery,call,takeLatest,cps,apply } from 'redux-saga/effects'
import {delay,read} from '../utils';

export function* readAsync() {
    let content = yield cps(read,'1.txt');
    console.log('content=',content);
}

export function* incrementAsync() {

    yield call(delay,3000);

    yield put({ type: 'INCREMENT' })
}
```

src/store/sagas.spec.js

```
import test from 'tape';
import { all,put, takeEvery,call,cps,apply } from 'redux-saga/effects';
import { incrementAsync,readAsync }  from './sagas';
import {delay,read} from '../utils';

test('readAsync Saga test', (assert) => {
  const gen = readAsync();
  assert.deepEqual(
    gen.next().value,
    cps(read,'1.txt'),
    'readAsync should be done  after 3 second'
  )
  assert.deepEqual(
    gen.next(),
    {done:true,value:undefined},
    'readAsync Saga must be done'
  )
  assert.end();
});
```

src/utils.js

```
export function read(filename,callback){
    setTimeout(function(){
        console.log('read',filename);
     callback(null,filename);
    },1000);
 }
```

9. 错误处理

- 我们可以使用熟悉的 try/catch 语法在 Saga 中捕获错误

src\store\sagas.js

```
export const delay2=ms => new Promise((resolve,reject) => {
    setTimeout(() => {
        if(Math.random()>.5){
            resolve();
        }else{
            reject();
        }
    },ms);
});
export function* incrementAsync2() {
    try{
        yield call(delay2,3000);
        yield put({ type:'INCREMENT'});
        alert('操作成功');
    }catch(error){
        alert('操作失败');
    }
}

export function* watchIncrementAsync() {

    yield takeLatest('INCREMENT_ASYNC', incrementAsync2);
}
```

- 你也可以让你的 API 服务返回一个正常的含有错误标识的值 src\store\sagas.js

```
export const delay3=ms => new Promise((resolve,reject) => {
  setTimeout(() => {
      let data = Math.random();
      resolve({
          code:data>.5?0:1,
          data
      });
  },ms);
});
export function* incrementAsync3() {
  let {code,data} = yield call(delay3,1000);
  if(code === 0){
      yield put({ type:'INCREMENT'});
      alert('操作成功 data='+data);
  }else{
      alert('操作失败');
  }
}

export function* watchIncrementAsync() {

  yield takeLatest('INCREMENT_ASYNC', incrementAsync3);
}
```

10. take

- takeEvery 只是一个在强大的低阶 API 之上构建的 `wrapper effect`
- take 就像我们更早之前看到的 call 和 put。它创建另一个命令对象，告诉 middleware 等待一个特定的 action

src/store/sagas.js

```
import {all,put,take,select } from 'redux-saga/effects'
import {INCREMENT_ASYNC,INCREMENT} from './action-types';

export function* watchIncrementAsync() {
    for (let i = 0; i < 3; i++) {
        const action = yield take(INCREMENT_ASYNC);
        console.log(action);
        yield put({type:INCREMENT});
    }
    alert('最多只能点三次!');
}
export function* watchAndLog() {

    while(true){
        let action = yield take('*');
        const state = yield select();
        console.log('action', action);
        console.log('state after', state);
    }
}

export default function* rootSaga() {
    yield all([
        watchAndLog(),
        watchIncrementAsync()
    ])
  }
```

11. 登陆流程

src/index.js

```
import React from 'react'
import ReactDOM from 'react-dom';
import Login from './components/Login';
import {Provider} from 'react-redux';
import store from './store';
ReactDOM.render(<Provider store={store}>
  <Login/>
Provider>,document.querySelector('#root'));
```

src/store/action-types.js

```
export const INCREMENT='INCREMENT';
export const INCREMENT_ASYNC='INCREMENT_ASYNC';
export const LOGIN_REQUEST='LOGIN_REQUEST';
export const LOGIN_SUCCESS='LOGIN_SUCCESS';
export const SET_USERNAME='SET_USERNAME';
export const LOGIN_ERROR='LOGIN_ERROR';
export const LOGOUT='LOGOUT';
```

src/store/actions.js

```
import * as types from './action-types';
export default {
    incrementAsync() {
        return {type:types.INCREMENT_ASYNC}
    },
    login(username,password){
        return {type:types.LOGIN_REQUEST,username,password}
    },
    logout(){
        return {type:types.LOGOUT}
    }
}
```

src/store/reducer.js

```
import * as types from './action-types';
export default function (state={number:0,username:null},action) {
    switch(action.type){
        case types.INCREMENT:
            return {number: state.number+1};
        case types.LOGIN_ERROR:
            return {error: action.error};
        case types.SET_USERNAME:
            return {username: action.username};
        default:
            return state;
    }
}
```

src/store/sagas.js

```
import { call, all, put, take } from "redux-saga/effects";
import {LOGIN_ERROR,LOGIN_REQUEST,SET_USERNAME,LOGOUT} from "./action-types";
import Api from "../Api";

function* login(username, password) {
  try {

    const token = yield call(Api.login, username, password);
    return token;
  } catch (error) {
    alert(error);

    yield put({
      type: LOGIN_ERROR,
      error
    });
  }
}

function* loginFlow() {

  while (true) {

    const { username, password } = yield take(LOGIN_REQUEST);
    const token = yield call(login, username, password);

    if (token) {
      yield put({
        type: SET_USERNAME,
        username
      });

      Api.storeItem("token", token);

      yield take(LOGOUT);
      Api.clearItem("token");
      yield put({
        type: SET_USERNAME,
        username: null
      });
    }
  }
}

export default function* rootSaga() {
  yield all([loginFlow()]);
}
```

src/Api.js

```
export default {
    login(username, password){
        return new Promise(function(resolve,reject){
            setTimeout(()=>{
                if(Math.random()>.5){
                    resolve(username+'-'+password);
                }else{
                    reject('登录失败');
                }
            },1000);
        });
    },
    storeItem(key,value){
        localStorage.setItem(key,value);
    },
    clearItem(){
        localStorage.removeItem('token');
    }
}
```

src/components/Login.js

```
import React,{Component} from 'react'
import {connect} from 'react-redux';
import actions from '../store/actions';
class Login extends Component{
    constructor(props){
        super(props);
        this.username=React.createRef();
        this.password=React.createRef();
    }
    login = (event)=>{
        event.preventDefault();
        let username = this.username.current.value;
        let password = this.password.current.value;
        this.props.login(username,password);
    }
    logout = (event)=>{
        event.preventDefault();
        this.props.logout();
    }
    render() {
        let {username} = this.props;
        let loginForm = (
            <form>
                <label>用户名label><input ref={this.username}/><br/>
                <label>密码label><input ref={this.password}/><br/>
                <button onClick={this.login}>登录button>
            form>
        )
        let logoutForm = (
            <form >
                 用户名:{username}<br/>
                <button onClick={this.logout}>退出button>
            form>
        )
        return (
            username?logoutForm:loginForm
      )
  }
}
export default connect(
    state => state,
    actions
)(Login);
```

12. fork

- 当 loginFlow 在 login 中被阻塞了，最终发生在开始调用和收到响应之间的 `LOGOUT` 将会被错过
- 我们需要的是一些非阻塞调用login
- 为了表示无阻塞调用，redux-saga 提供了另一个 Effect： `fork`,当我们 fork 一个 任务，任务会在后台启动，调用者也可以继续它自己的流程，而不用等待被 fork 的任务结束

src/store/sagas.js

```
import {call,all,put,take,fork} from 'redux-saga/effects'
import {LOGIN_ERROR,LOGOUT,LOGIN_REQUEST,LOGIN_SUCCESS,SET_USERNAME} from './action-types';
import Api from '../Api'

function* login(username, password) {
    try {

        const token = yield call(Api.login, username, password);
        yield put({type: LOGIN_SUCCESS, token});
        yield put({type: SET_USERNAME, username});

        Api.storeItem('token',token);
    } catch(error) {

        yield put({type: LOGIN_ERROR, error});
    }
  }

  function* loginFlow() {

    while(true) {

        const {username, password} = yield take(LOGIN_REQUEST);

        yield fork(login, username, password);

        yield take([LOGOUT,LOGIN_ERROR]);
        Api.clearItem('token');
    }
  }
export default function* rootSaga() {
    yield all([
        loginFlow()
    ])
}
```

13. 取消任务

- 如果我们在 API 调用期间收到一个 LOGOUT action，我们必须要 取消 login 处理进程,否则将有 2 个并发的任务， 并且 login 任务将会继续运行，并在成功的响应（或失败的响应）返回后发起一个 LOGIN_SUCCESS action（或一个 LOGIN_ERROR action），而这将导致状态不一致
- cancel Effect 不会粗暴地结束我们的 login 任务，相反它会给予一个机会执行清理的逻辑,在 finally 区块可以处理任何的取消逻辑（以及其他类型的完成逻辑）

src/components/Login.js

```
class Login extends Component{
    render() {
        let {token} = this.props;
        let loginForm = (

                用户名
                密码
                登录
+                 退出

        )
  }
}
```

src/store/sagas.js

```
import {call,all,put,take,fork,cancel,cancelled} from 'redux-saga/effects'
import {LOGIN_ERROR,LOGOUT,LOGIN_REQUEST,LOGIN_SUCCESS} from './action-types';
import Api from '../Api'

function* login(username, password) {
    try {

        Api.storeItem('loading','true');
        const token = yield call(Api.login, username, password);
        yield put({type: LOGIN_SUCCESS, token});

        Api.storeItem('token',token);
        Api.storeItem('loading','xx');
    } catch(error) {

        yield put({type: LOGIN_ERROR, error});
        Api.storeItem('loading','false');
    } finally {
        console.log(cancelled())
        if (yield cancelled()) {

          Api.storeItem('loading','false');
        }
      }
  }

  function* loginFlow() {

    while(true) {

        const {username, password} = yield take(LOGIN_REQUEST);

        const task = yield fork(login, username, password);

        const action = yield take([LOGOUT,LOGIN_ERROR]);

        if(action.type == LOGOUT){
            yield cancel(task);
        }
        Api.clearItem('token');
    }
}
export default function* rootSaga() {
    yield all([
        loginFlow()
    ])
}
```

14. race

- 有时候我们同时启动多个任务，但又不想等待所有任务完成，我们只希望拿到 胜利者：即第一个被 resolve（或 reject）的任务
- race 的另一个有用的功能是，它会自动取消那些失败的 Effects

src/index.js

```
import React from 'react'
import ReactDOM from 'react-dom';
import Login from './components/Login';
import Recorder from './components/Recorder';
import {Provider} from 'react-redux';
import store from './store';
ReactDOM.render(<Provider store={store}>
  <Recorder/>
Provider>,document.querySelector('#root'));
```

src/store/action-types.js

```
export const CANCEL_TASK='CANCEL_TASK';
```

src/store/actions.js

```
stop(){
        return {type:types.CANCEL_TASK}
}
```

src/store/sagas.js

```
import {call,all,put,take,race} from 'redux-saga/effects'
import {INCREMENT,CANCEL_TASK} from './action-types';
import {delay} from '../utils';

function* raceFlow() {
    const {a, b} = yield race({
        a: call(delay, 1000),
        b: call(delay, 2000)
    });
    console.log('a='+a,'b='+b);
}

function* start() {
    while(true){
        yield call(delay,1000);
        yield put({type:INCREMENT});
    }
}

function* recorder() {
    yield race({
        start: call(start),
        stop: take(CANCEL_TASK)
    });
}
export default function* rootSaga() {

    yield all([recorder()])
}
```

src/components/Counter.js

```
import React,{Component} from 'react'
import {connect} from 'react-redux';
import actions from '../store/actions';
class Counter extends Component{
    render() {
        return (
            <div>
                <p>{this.props.number}p>
                <button onClick={this.props.stop}>停止button>
            div>
      )
  }
}
export default connect(
    state => state,
    actions
)(Counter);
```

参考