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=118 sentences=336, words=2978

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 做具体的工作，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

```
function* rootSaga() {

    yield { type: 'PUT', action: { type: "ADD" } };

    yield new Promise(resolve => setTimeout(resolve, 3000))
    yield { type: 'PUT', action: { type: "MINUS" } };
}
function runSaga(saga) {

    const it = saga();
    function next() {
        const { done, value: effect } = it.next();
        if (!done) {
            if (effect instanceof Promise) {
                effect.then(next);
            } else if (effect.type === 'PUT') {
                console.log(`向仓库派发一个动作${JSON.stringify(effect.action)}`);
                next();
            } else {
                next();
            }
        }
    }
    next();
}
runSaga(rootSaga);
```

```
function * gen(){
    yield 1;
    yield 2;
    yield 3;
}
let it = gen();
console.log(it[Symbol.iterator]);
let r1 = it.next();
console.log(r1);

let r2 = it.return();
console.log(r2);
let r3 = it.next();
console.log(r3);
let r4 = it.next();
console.log(r4);
```

```
let EventEmitter = require('events');
let e = new EventEmitter();
e.once('click',(data)=>{
    console.log('clicked',data);
});
e.emit('click','data');
e.emit('click','data');
```

4. 计数器

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\rootSaga.js

```
import {put,take} from 'redux-saga/effects';
import * as types from './action-types';
function delay(ms) {
    return new Promise((resolve) => {
        setTimeout(resolve, ms);
    });
}
function * workerSaga(){
  yield delay(1000);
  yield put({type:actionTypes.ADD});
}
function * watcherSaga(){
  yield take(actionTypes.ASYNC_ADD);
  yield workerSaga();
}

export default function* rootSaga() {
  yield watcherSaga();
}
```

src/components/Counter.js

```
import React from 'react';
import * as actionTypes from '../store/action-types';
import { useSelector, useDispatch } from 'react-redux';
function Counter() {
    const number = useSelector(state => state.number);
    const dispatch = useDispatch();
    return (
        <div>
            <p>{number}p>
            <button onClick={() => dispatch({ type: actionTypes.ASYNC_ADD })}>+button>
        div>
    )
}
export default Counter;
```

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);
window.store=store;
export default store;
```

src/store/action-types.js

```
export const ASYNC_ADD='ASYNC_ADD';
export const ADD='ADD';
```

src/store/reducer.js

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

5. 实现take

src\redux-saga\effectTypes.js

```
export const TAKE = 'TAKE';
export const PUT = 'PUT';
```

src\redux-saga\effects.js

```
import * as effectTypes from './effectTypes'
export function take(actionType) {
    return { type: effectTypes.TAKE, actionType}
}

export function put(action) {
    return { type: effectTypes.PUT, action }
}
```

src\redux-saga\runSaga.js

```
import * as effectTypes from './effectTypes'
export default function runSaga(env, saga) {
    let { channel, dispatch } = env;
    let it = typeof saga === 'function'?saga():saga;
    function next(value) {
        let {value:effect,done} = it.next(value);
        if (!done) {
            if(typeof effect[Symbol.iterator] === 'function'){
                runSaga(env,effect);
                next();
            }else if (effect instanceof Promise) {
                effect.then(next);
            }else{
                switch (effect.type) {
                    case effectTypes.TAKE:
                        channel.once(effect.actionType,next);
                        break;
                    case effectTypes.PUT:
                        dispatch(effect.action);
                        next();
                        break;
                    default:
                        break;
                }
            }

        }
    }
    next();
}
```

redux-saga/index.js

```
import EventEmitter from 'events';
import runSaga from './runSaga';
export default function createSagaMiddleware() {
    let channel = new EventEmitter();
    let boundRunSaga;
    function sagaMiddleware({getState,dispatch}) {
        boundRunSaga=runSaga.bind(null,{channel,dispatch,getState});
        return function (next) {
            return function (action) {
                const result = next(action);
                channel.emit(action.type, action);
                return result;
            }
        }
    }
    sagaMiddleware.run = (saga)=>boundRunSaga(saga);
    return sagaMiddleware;
}
```

6. 支持fork

```
+import { put, takeEvery } from '../redux-saga/effects';
import * as actionTypes from './action-types';
+const delay = (ms)=>{
+    return new Promise(resolve=>{
+        setTimeout(resolve,ms);
+    });
+}
+export function* workerSaga() {
+    yield delay(1000);
+    yield put({ type: actionTypes.ADD });
+}
+function * watcherSaga(){
+    const action = yield take(actionTypes.ASYNC_ADD);
+    yield fork(workerSaga);
+}
+export default function* rootSaga() {
+   yield watcherSaga();
+}
```

src\redux-saga\effectTypes.js

```
export const TAKE = 'TAKE';
export const PUT = 'PUT';
+export const FORK = 'FORK';
```

src\redux-saga\effects.js

```
import * as effectTypes from './effectTypes'
export function take(actionType) {
    return { type: effectTypes.TAKE, actionType }
}

export function put(action) {
    return { type: effectTypes.PUT, action }
}

+export function fork(saga) {
+    return { type: effectTypes.FORK, saga };
+}
```

src\redux-saga\runSaga.js

```
import * as effectTypes from './effectTypes'
export default function runSaga(env, saga) {
    let { channel, dispatch } = env;
    let it = typeof saga == 'function' ? saga() : saga;
    function next(value) {
        let { value: effect, done } = it.next(value);
        if (!done) {
            if (typeof effect[Symbol.iterator] == 'function') {
                runSaga(env,effect);
                next();
            } else {
                switch (effect.type) {
                    case effectTypes.TAKE:
                        channel.take(effect.actionType, next);
                        break;
                    case effectTypes.PUT:
                        dispatch(effect.action);
                        next();
                        break;
+                    case effectTypes.FORK:
+                        runSaga(env,effect.saga);
+                        next();
+                        break;
                    default:
                        break;
                }
            }

        }
    }
    next();
}
```

7. 支持takeEvery

- 一个 task 就像是一个在后台运行的进程，在基于redux-saga的应用程序中，可以同时运行多个task
- 通过 fork 函数来创建 task

```
+import { put, takeEvery } from '../redux-saga/effects';
import * as actionTypes from './action-types';
export function* add() {
    yield put({ type: actionTypes.ADD });
}
export default function* rootSaga() {
+    yield takeEvery(actionTypes.ASYNC_ADD,add);
}
```

src\redux-saga\effects.js

```
import * as effectTypes from './effectTypes'
export function take(actionType) {
    return { type: effectTypes.TAKE, actionType }
}

export function put(action) {
    return { type: effectTypes.PUT, action }
}

export function fork(saga) {
    return { type: effectTypes.FORK, saga };
}

+export function takeEvery(actionType, saga) {
+    function* takeEveryHelper() {
+        while (true) {
+            yield take(actionType);
+            yield fork(saga);
+        }
+    }
+    return fork(takeEveryHelper);
+}
```

8. 支持 call

src\store\sagas.js

```
+import { put, takeEvery,call } from '../redux-saga/effects';
import * as actionTypes from './action-types';
const delay = ms => new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve();
    }, ms);
});
export function* add() {
+    yield call(delay,1000);
    yield put({ type: actionTypes.ADD });
}
export default function* rootSaga() {
    yield takeEvery(actionTypes.ASYNC_ADD, add);
}
```

src\redux-saga\effectTypes.js

```
export const TAKE = 'TAKE';
export const PUT = 'PUT';
export const FORK = 'FORK';
+export const CALL = 'CALL';
```

src\redux-saga\effects.js

```
import * as effectTypes from './effectTypes'
export function take(actionType) {
    return { type: effectTypes.TAKE, actionType }
}

export function put(action) {
    return { type: effectTypes.PUT, action }
}

export function fork(saga) {
    return { type: effectTypes.FORK, saga };
}

export function takeEvery(pattern, saga) {
    function* takeEveryHelper() {
        while (true) {
            yield take(pattern);
            yield fork(saga);
        }
    }
    return fork(takeEveryHelper);
}
+export function call(fn, ...args) {
+    return { type: effectTypes.CALL, fn, args };
+}
```

src\redux-saga\runSaga.js

```
import * as effectTypes from './effectTypes'
export default function runSaga(env, saga) {
    let { channel, dispatch } = env;
    let it = typeof saga == 'function' ? saga() : saga;
    function next(value) {
        let { value: effect, done } = it.next(value);
        if (!done) {
            if (typeof effect[Symbol.iterator] == 'function') {
                runSaga(env,effect);
                next();
            }else if(effect.then){
                effect.then(next);
            } else {
                switch (effect.type) {
                    case effectTypes.TAKE:
                        channel.take(effect.actionType, next);
                        break;
                    case effectTypes.PUT:
                        dispatch(effect.action);
                        next();
                        break;
                    case effectTypes.FORK:
                        runSaga(env,effect.saga);
                        next();
                        break;
+                    case effectTypes.CALL:
+                        effect.fn(...effect.args).then(next);
+                        break;
                    default:
                        break;
                }
            }

        }
    }
    next();
}
```

9. 支持 cps

src\store\sagas.js

```
+import { put, takeEvery,call,cps} from '../redux-saga/effects';
import * as actionTypes from './action-types';
+const delay = (ms,callback)=>{
+    setTimeout(() => {
+        callback(null,'ok');
+    },ms);
+}
export function* add() {
+    let data = yield cps(delay,1000);
+    console.log(data);
    yield put({ type: actionTypes.ADD });
}
export default function* rootSaga() {
    yield takeEvery(actionTypes.ASYNC_ADD, add);
}
```

src\redux-saga\effectTypes.js

```
export const TAKE = 'TAKE';
export const PUT = 'PUT';
export const FORK = 'FORK';
export const CALL = 'CALL';
+export const CPS = 'CPS';
```

src\redux-saga\effects.js

```
import * as effectTypes from './effectTypes'
export function take(actionType) {
    return { type: effectTypes.TAKE, actionType }
}

export function put(action) {
    return { type: effectTypes.PUT, action }
}

export function fork(saga) {
    return { type: effectTypes.FORK, saga };
}

export function takeEvery(pattern, saga) {
    function* takeEveryHelper() {
        while (true) {
            yield take(pattern);
            yield fork(saga);
        }
    }
    return fork(takeEveryHelper);
}
export function call(fn, ...args) {
    return { type: effectTypes.CALL, fn, args };
}
+export function cps(fn, ...args) {
+    return { type: effectTypes.CPS, fn, args };
+}
```

src\redux-saga\runSaga.js

```
import * as effectTypes from './effectTypes'
export default function runSaga(env, saga) {
    let { channel, dispatch } = env;
    let it = typeof saga == 'function' ? saga() : saga;
+    function next(value,isErr) {
+        let result;
+        if (isErr) {
+            result = it.throw(value);
+          } else {
+            result = it.next(value);
+          }
+        let { value: effect, done } = result;
        if (!done) {
            if (typeof effect[Symbol.iterator] == 'function') {
                runSaga(env,effect);
                next();
            }else if(effect.then){
                effect.then(next);
            } else {
                switch (effect.type) {
                    case effectTypes.TAKE:
                        channel.take(effect.actionType, next);
                        break;
                    case effectTypes.PUT:
                        dispatch(effect.action);
                        next();
                        break;
                    case effectTypes.FORK:
                        runSaga(env,effect.saga);
                        next();
                        break;
                    case effectTypes.CALL:
                        effect.fn(...effect.args).then(next);
                        break;
+                    case effectTypes.CPS:
+                        effect.fn(...effect.args,(err,data)=>{
+                            if(err){
+                                next(err,true);
+                            }else{
+                                next(data);
+                            }
+                        });
+                        break;
                    default:
                        break;
                }
            }

        }
    }
    next();
}
```

11. 支持all

src\store\sagas.js

```
import { put, takeEvery, call, cps, take,all } from '../redux-saga/effects';
import * as actionTypes from './action-types';
+export function* add1() {
+    for (let i = 0; i < 1; i++) {
+        yield take(actionTypes.ASYNC_ADD);
+        yield put({ type: actionTypes.ADD });
+    }
+    console.log('add1 done ');
+    return 'add1Result';
+}
+export function* add2() {
+    for (let i = 0; i < 2; i++) {
+        yield take(actionTypes.ASYNC_ADD);
+        yield put({ type: actionTypes.ADD });
+    }
+    console.log('add2 done ');
+    return 'add2Result';
+}
export default function* rootSaga() {
+    let result = yield all([add1(), add2()]);
+    console.log('done', result);
}
```

src\redux-saga\effectTypes.js

```
export const TAKE = 'TAKE';
export const PUT = 'PUT';
export const FORK = 'FORK';
export const CALL = 'CALL';
export const CPS = 'CPS';
export const ALL = 'ALL';
```

src\redux-saga\effects.js

```
import * as effectTypes from './effectTypes'
export function take(actionType) {
    return { type: effectTypes.TAKE, actionType }
}

export function put(action) {
    return { type: effectTypes.PUT, action }
}

export function fork(saga) {
    return { type: effectTypes.FORK, saga };
}

export function takeEvery(pattern, saga) {
    function* takeEveryHelper() {
        while (true) {
            yield take(pattern);
            yield fork(saga);
        }
    }
    return fork(takeEveryHelper);
}
export function call(fn, ...args) {
    return { type: effectTypes.CALL, fn, args };
}
export function cps(fn, ...args) {
    return { type: effectTypes.CPS, fn, args };
}
+export function all(iterators) {
+    return { type: effectTypes.ALL, iterators };
+}
```

src\redux-saga\runSaga.js

```
import * as effectTypes from './effectTypes'
+export default function runSaga(env, saga,callback) {
    let { channel, dispatch } = env;
    let it = typeof saga == 'function' ? saga() : saga;
    function next(value, isErr) {
        let result;
        if (isErr) {
            result = it.throw(value);
        } else {
            result = it.next(value);
        }
        let { value: effect, done } = result;
        if (!done) {
            if (typeof effect[Symbol.iterator] == 'function') {
                runSaga(env, effect);
                next();
            } else if (effect.then) {
                effect.then(next);
            } else {
                switch (effect.type) {
                    case effectTypes.TAKE:
                        channel.take(effect.actionType, next);
                        break;
                    case effectTypes.PUT:
                        dispatch(effect.action);
                        next();
                        break;
                    case effectTypes.FORK:
                        runSaga(env, effect.saga);
                        next();
                        break;
                    case effectTypes.CALL:
                        effect.fn(...effect.args).then(next);
                        break;
                    case effectTypes.CPS:
                        effect.fn(...effect.args, (err, data) => {
                            if (err) {
                                next(err, true);
                            } else {
                                next(data);
                            }
                        });
                        break;
+                    case effectTypes.ALL:
+                        const { iterators } = effect;
+                        let result = [];
+                        let count = 0;
+                        iterators.forEach((iterator, index) => {
+                        runSaga(env, iterator, (data) => {
+                            result[index] = data;
+                            if (++count === iterators.length) {
+                            next(result);
+                            }
+                        });
+                        });
+                        break;
                    default:
                        break;
                }
            }
        } else {
+            callback && callback(effect);
        }
    }
    next();
}
```

12. 取消任务

src\store\sagas.js

```
+import { put, takeEvery, call, cps, all, take, cancel, fork, delay } from '../redux-saga/effects';
+import * as actionTypes from './action-types';
+export function* add() {
+    while (true) {
+        yield delay(1000);
+        yield put({ type: actionTypes.ADD });
+    }
+}
+export function* addWatcher() {
+    const task = yield fork(add);
+    console.log(task);
+    yield take(actionTypes.STOP_ADD);
+    yield cancel(task);
+}
+function* request(action) {
+    let url = action.payload;
+    let promise = fetch(url).then(res => res.json());;
+    let res = yield promise;
+    console.log(res);
+}
+
+function* requestWatcher() {
+    //action = {type,url}
+    const requestAction = yield take(actionTypes.REQUEST);
+    //开启一个新的子进程发起请求
+    const requestTask = yield fork(request, requestAction);
+    //立刻开始等待停止请求的动作类型
+    const stopAction = yield take(actionTypes.STOP_REQUEST);
+    yield cancel(requestTask);//在axios里，是通过 调用promise的reject方法来实出任务取消
+}
+export default function* rootSaga() {
+    yield addWatcher();
+    yield requestWatcher();
+}
```

src\redux-saga\effectTypes.js

```
export const TAKE = 'TAKE';
export const PUT = 'PUT';
export const FORK = 'FORK';
export const CALL = 'CALL';
export const CPS = 'CPS';
export const ALL = 'ALL';
+export const CANCEL = 'CANCEL';
```

src\redux-saga\effects.js

```
import * as effectTypes from './effectTypes'
export function take(actionType) {
    return { type: effectTypes.TAKE, actionType }
}

export function put(action) {
    return { type: effectTypes.PUT, action }
}

export function fork(saga) {
    return { type: effectTypes.FORK, saga };
}

export function takeEvery(pattern, saga) {
    function* takeEveryHelper() {
        while (true) {
            yield take(pattern);
            yield fork(saga);
        }
    }
    return fork(takeEveryHelper);
}
export function call(fn, ...args) {
    return { type: effectTypes.CALL, fn, args };
}
export function cps(fn, ...args) {
    return { type: effectTypes.CPS, fn, args };
}
export function all(effects) {
    return { type: effectTypes.ALL, effects };
}
+const delayFn = (ms) => {
+    return new Promise(resolve => {
+        setTimeout(resolve, ms);
+    })
+}
+export function delay(...args) {
+    return call(delayFn, ...args);
+}
+export function cancel(task) {
+    return { type: effectTypes.CANCEL, task };
+}
```

src\redux-saga\runSaga.js

```
import * as effectTypes from './effectTypes';
const CANCEL_TASK = 'CANCEL_TASK';
export default function runSaga(env, saga,callback) {
+    let task = {cancel:()=>next(TASK_CANCEL)};
    let { channel, dispatch } = env;
    let it = typeof saga == 'function' ? saga() : saga;
    function next(value, isErr) {
        let result;
        if (isErr) {
            result = it.throw(value);
+        }else if(value === TASK_CANCEL){
+            result = it.return(value);
        } else {
            result = it.next(value);
        }
        let { value: effect, done } = result;
        if (!done) {
            if (typeof effect[Symbol.iterator] == 'function') {
                runSaga(env, effect);
                next();
            } else if (effect.then) {
                effect.then(next);
            } else {
                switch (effect.type) {
                    case effectTypes.TAKE:
                        channel.take(effect.actionType, next);
                        break;
                    case effectTypes.PUT:
                        dispatch(effect.action);
                        next();
                        break;
                    case effectTypes.FORK:
+                        let forkTask = runSaga(env, effect.saga);
+                        next(forkTask);
                        break;
                    case effectTypes.CALL:
                        effect.fn(...effect.args).then(next);
                        break;
                    case effectTypes.CPS:
                        effect.fn(...effect.args, (err, data) => {
                            if (err) {
                                next(err, true);
                            } else {
                                next(data);
                            }
                        });
                        break;
                    case effectTypes.ALL:
                        const { iterators } = effect;
                        let result = [];
                        let count = 0;
                        iterators.forEach((iterator, index) => {
                            runSaga(env, iterator, (data) => {
                                result[index] = data;
                                if (++count
                                next(result);
                                }
                            });
                        });
                        break;
+                    case effectTypes.CANCEL:
+                        effect.task.cancel();
+                        next();
+                        break;
                    default:
                        break;
                }
            }
        } else {
            callback && callback(effect);
        }
    }
    next();
+    return task;
}
```

src\store\action-types.js

```
export const ASYNC_ADD='ASYNC_ADD';
export const ADD='ADD';
+export const STOP='STOP';

+export const REQUEST = 'REQUEST';
+export const STOP_REQUEST = 'STOP_REQUEST';
```

src\components\Counter.js

```
import React from 'react';
import * as actionTypes from '../store/action-types';
import { useSelector, useDispatch } from 'react-redux';
function Counter() {
    const number = useSelector(state => state.number);
    const dispatch = useDispatch();
    return (

            {number}
+             dispatch({ type: actionTypes.ASYNC_ADD })}>+
+             dispatch({ type: actionTypes.STOP_ADD })}>stop
+             dispatch({ type: actionTypes.REQUEST, payload: '/users.json' })}>request
+             dispatch({ type: actionTypes.STOP_REQUEST })}>stopRequest

    )
}
export default Counter;
```